МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук

«_____»_____ 2020 р.

На правах рукопису
УДК 003.26:004.056.55

ДИПЛОМНА РОБОТА

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
«МАГІСТР»

Тема: Програмний модуль захисту інформації від мережевих атак на веб-додатки

Автор: Коваль Д.В.

Науковий керівник: к.т.н., доц. Гулак Н.К.

Нормоконтролер: к.т.н., доц. Гулак Н.К.

Київ 2020

ВСТУП

Актуальність. Зі збільшенням залежності компаній будь-якого напрямку діяльності від IT технологій, гостро постає питання забезпечення інформаційної безпеки. Одним з ключових заходів в забезпеченні інформаційної безпеки компанії є захист своєї інформації. Це дозволяє упевнитися в надійності захисту від несанкціонованого доступу та інших загроз інформаційної безпеки. [1] Після аналізу повідомлень у засобах масової інформації, та повідомлення команди аварійні події в Україні (CERT-UA). Було зроблено висновок, що упродовж останніх кількох років втручання організованих груп зловмисників в роботу комп'ютерних систем установ та підприємств призвело до блокування роботи, матеріального та репутаційного збитку. BlackEnergy на інформаційно-телекомунікаційні системи Міністерства фінансів, Державної казначейської служби. Ми повинні визнати, що нині в Україні законодавство в сфері захисту інформації застаріло, і не відповідає вимогам сьогодення. В Україні відсутня єдина затверджена методика захисту веб-додатків [2], саме тому, рівень захисту комп'ютерних систем не відповідає існуючим загрозам. Не дивлячись на те, що глобальні організації в сфері кібербезпеки займаються дослідженнями уразливості веб-додатків, та існують стандарти обробки інформації, інструменти та методи виявлення уразливостей. Проте, методи сканування уразливостей є загальними та охоплюють широкий спектр тем. Таким чином, ці методи можуть бути надлишковими, якщо вони застосовуються до конкретних систем. Отже, при пошуку деяких уразливостей може бути витрачено необгрунтовано багато часу. У зв'язку з цим пропонується провести аналіз наявних методик для подальшої розробки нової методики, що підходить для Українських стандартів.

Метою дипломної роботи є удосконалення програмного модуля для підвищення ефективності та швидкості тестування на проникнення в веб-додатки для підвищення рівня захисту інформації.

Об'єкт дослідження: процес захисту інформації за допомогою виявлення вразливостей у веб-додатках.

Предмет дослідження: пошук мережевих атак для захисту інформації.

Галузь застосування: даний модуль може використовуватися у галузі бізнес-середовища. Якщо виключити з розгляду рішення вузьких завдань на зразок файлового обміну, то мова в першу чергу повинна йти про безпеку інформації у компанії.

Новизна одержаних результатів полягає в наступному:

– удасконалено модуль захисту інформації з використанням нових технологій, що дало можливість підключення його до будь-якого веб-додатяку за рахунок розробки нового програного продукту мовою програмування JavaScript на платформі VueJS, HTML, CSS, Express.

Практична цінність у створенні удосконаленого програмного модуля мовою програмування JavaScript, NodeJS та на платформі VueJS, HTML, CSS, Express, який може використовуватися у компаніях, яким буде потрібен модуль для захисту інформації від мережевих атак у веб-додатках.

Апробація. Основні положення роботи доповідалися та обговорювалися на таких конференціях:

Коваль Д.В. Програмний модуль захисту інформації. Матеріали XVI науково-практичної конференції «Наука и іновація», 07.10.2020 р. -15.10.2020 р., Польща.

Розділ 1. АНАЛІЗ ВРАЗЛИВОСТЕЙ ВЕБ-ДОДАТКІВ

1.1. Аналіз можливих атак на веб-додатки

Веб-додатки стають жертвами різноманітних атак, які порушують різні властивості безпеки, що повинні застосовуватися додатком. Потрібно звернути увагу, що це не стосується атак, які можуть стосуватися інфраструктури (наприклад, щодо веб-сервера та баз даних) або роботи мережі (наприклад, щодо маршрутизаторів та брандмауерів) на атаки, які намагаються спонукати веб-програму поводитися непередбачувано (і небажаними) способами розкривати конфіденційну інформацію або виконувати команди від імені зловмисника. Багато веб-додатків пропонують послуги, доступні лише зареєстрованим користувачам, наприклад, «преміум» функціональність або персоналізований вміст. Ці служби вимагають наявності певного механізму автентифікації для встановлення ідентичності користувачів. Помилки в коді автентифікації або логіці можуть бути використані щоб обійти автентифікацію або блокування законних користувачів. Наприклад, облікові дані користувачів, передані в явному вигляді в програму, можуть бути викрадені підслуховуванням мережевим підключенням та слабкі механізми автентифікації можуть бути порушені грубою силою чи атаками зловмисника [3]. Після автентифікації користувача програма повинна застосовувати політику, яка встановлює, які ресурси доступні користувачеві. Порушена авторизація може призвести до підвищення привілеїв, розголошення конфіденційних даних та відпустки даних. Механізми авторизації особливо важливі, коли веб-програми обробляють конфіденційні дані, такі як фінансова інформація або інформація про стан здоров'я. Веб-додатки, як правило, являють собою великі, неоднорідні, складні додатки, конфігурація яких далеко не тривіальна. Проблеми з конфігурацією можуть впливати як на інфраструктуру (наприклад, обліковий запис, під яким працює веб-сервер) так і на конфігурацію внутрішньої бази даних, а також на сам веб-додаток (наприклад, де програма зберігає свої тимчасові файли). Помилки конфігурації можуть дозволити зловмисникові обійти ефективні механізми автентифікації та авторизації. Наприклад, неправильна конфігурація була

використана для отримання несанкціонованого доступу до адміністративних функцій або отримання конфіденційної інформації, наприклад секретів, що зберігаються у вигляді простого тексту у файлах конфігурації, таких як паролі сервера баз даних. Атаки, які використовують погано розроблену автентифікацію, несправні механізми авторизації або конфігурації, є причиною серйозних компромісів. Однак в даний час більшість атак на веб-додатки можна віднести до одного класу вразливостей: неправильної перевірки вводу. Більшість розробників веб-додатків припускають, що вони можуть отримувати від своїх користувачів неправильне введення або в результаті помилки, або зловмисного наміру.

Перевірка введення - це захисна техніка програмування, яка гарантує, що всі введені користувачем дані мають очікуваний формат і не містять небезпечного вмісту. Хоча в принципі проста, але правильна та повна перевірка всіх вхідних даних є трудомістким завданням, яке вимагає значних знань. Тому цей тип недоліків занадто поширений у сучасних веб-додатках. Решта цього розділу досліджує різні типи атак, які використовують переваги неправильної або відсутньої перевірки вводу.

Було проведено аналіз вразливостей веб-додатків та показано на рис 1.1.

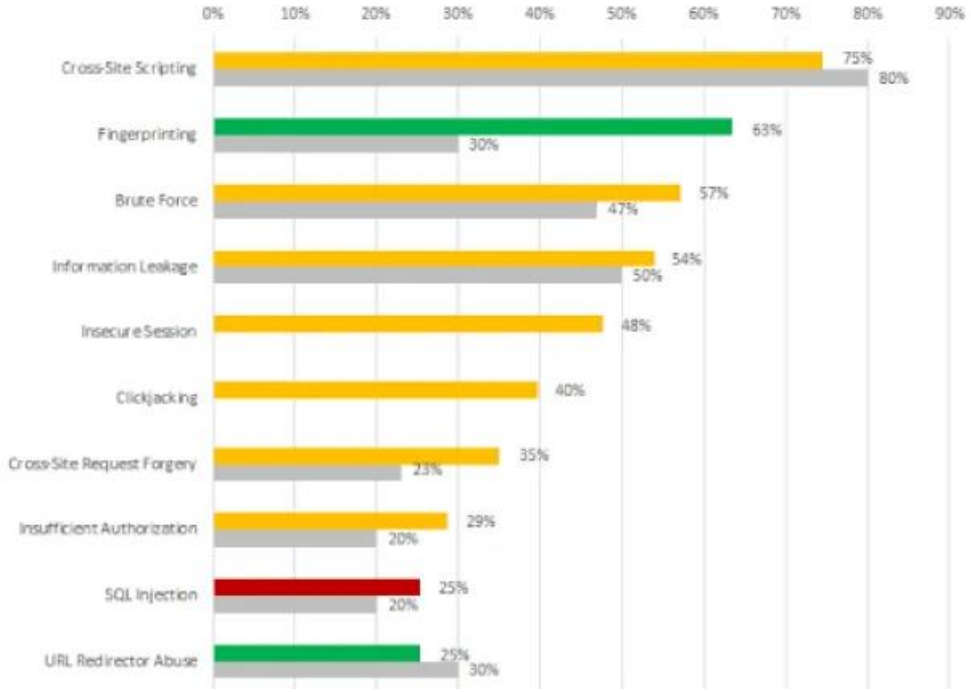

Рис. 1.1. Аналіз вразливостей веб-додатків

1.1.1. Ін’єкція перекладача

Багато динамічних мов включають функції для динамічного складання та інтерпретації коду. Наприклад, мова PHP забезпечує функцію eval, яка приймає рядок як параметр і обчислює його як PHP-код. Якщо не позначений користувацький вхід використовується для складання рядка, що підлягає оцінці, - цей веб-додаток вразливий до довільного виконання коду. Наприклад, введення перекладача, який був присутній у Double Choco Latte (версія 0.9.4.3 і раніше) Веб-додаток PHP, що забезпечує базову функціональність управління проектами [4]. URL-адреса атаки має вигляд:

http: // [ціль] / [каталог dcl] /

Частини URL-адреси запиту, що містять рядки menuAction потрібні, щоб уникнути помилок. Довільний код є частиною між цими двома рядками і, у наведеному вище прикладі, відповідає до частини, що містить систему (id). Уразливість міститься в такому фрагменті коду:

```
if ($ g_oSec -> ValidateMenuAction () == true)
{
list ($ class, $ method) = explode (".", $ menuAction);
$ obj = CreateObject (‘dcl.’. $ class);
eval ("\ $ obj -> $ method ();");
}
else
{
commonHeader ();
PrintPermissionDenied ();
}
```

Як видно з наведеного вище коду, змінні класу та методу, отримані з контрольованої користувачем змінної menuAction, ніколи не перевіряються. Отже, можна вставити команду, яка буде виконана, у рядок, яка представляє значення змінної. Після запиту URL-адреси атаки виклик eval стає:

eval ("$ obj-> show; system (id); ob_start ();");

Таким чином, окрім виконання методу show на об'єкті obj, інтерпретатор також виконає команду, визначену зловмисником. У цьому, наприклад, буде виконана команда id UNIX та буде надрукований ідентифікатор користувача, під яким виконується команда. Звичайно, довільні (і більш зловмисні) команди можуть бути виконані.

Однією з труднощів запобігання атакам введення перекладача є такі популярні мови, які пропонують багато векторних атак. У PHP eval та preg replace можуть використовувати для інтерпретації PHP-коду. Крім того, система функцій, passthru, backticks, і shell exec передає команду системній оболонці. Нарешті, exec, pcntl exec, popen і proc open можуть бути використані для виконання зовнішніх програм.

Деякі мови пропонують примітивні дезінфікуючі примітиви, які забезпечують цей зловмисний ввід користувача належним чином видаляється перед використанням. Наприклад, у PHP, escapeshellarg та escapeshellcmd можуть використовуватися для виходу з лапок або інших спеціальних символів, які можна вставити, щоб обдурити виконання програми.

1.1.2. Ін’єкція імені файлу

Більшість мов, що використовуються при розробці веб-додатків, дозволяють програмістам динамічно включати файли або для інтерпретації їх вмісту, або презентувати їх користувачеві. Ця функція використовується, наприклад, для модуляції додатку, розділяючи загальні функції на різні файли або генерувати різний вміст сторінки, залежно від уподобань користувача, наприклад, для цілей інтернаціоналізації. Якщо вибором файлу для включення можна маніпулювати користувачем, тоді може виникнути низка непередбачених наслідків. Погіршують ситуацію, деякі мови, зокрема PHP, навіть підтримують включення файлів з віддалених сайтів. Наступний фрагмент коду ілюструє вразливість імені файлу у txtForum - програма для створення форумів [5]. У txtForum сторінки є розділені на частини, наприклад, верхній, нижній колонтитул, перегляд форуму тощо, і можуть бути налаштовані використовуючи

різні «скіни», які відрізняються поєднанням кольорів, шрифтів та іншими параметрами презентації. Наприклад, код, який визначає заголовок є наступним:

DEFINE ("skin", "$ skin");

...

function

t_header ($ h_title, $ pre_skin = ’’, $ admin_bgcolor = ’’) {

...

include (SKIN. ’/ header .tpl’);

}

Під час виконання кожна сторінка складається, просто викликаючи функції, які відповідають за створення різних частин, наприклад, заголовок t ("заголовок сторінки"). На жаль, шкірною змінною може керувати зловмисник, який може налаштувати його на включення та оцінку довільного змісту. Оскільки PHP дозволяє включати віддалені файли, доданий код до програми може розміщуватися на веб-сайті під контролем зловмисника. Наприклад, запит сторінки login.php та передача обкладинки параметра зі значенням http: // [сайт зловмисника] веде до виконання коду на *http: // [сайт зловмисника] /header.tpl.* Для цього типу проблем PHP не пропонує жодних методів санітарії. Тому відповідні спеціальні перевірки повинні бути проведені розробниками.

1.1.3. Міжсайтові сценарії

При атаці міжсайтового сценарію (XSS) зловмисник змушує клієнта, як правило, запустити веб-браузер для виконання виконуваного коду, що надається зловмисником за допомогою, як правило, мови JavaScript, яка працює в контексті надійного веб-сайту [6]. Ця атака дозволяє зловмисникові обійти політику того самого походження, яку застосовують браузери, під час виконання коду на стороні клієнта.

Політика того самого походження стверджує, що сценарії або документи, завантажені з одного веб-сайту, не можуть отримати або встановити властивості документів з різних веб-сайтів (тобто різні “походження”). Це запобігає, наприклад, шкідливим веб-додаткам від викрадення конфіденційної інформації,

наприклад файлів cookie, що містять автентифікацію, пов'язана з іншими програмами, що працюють на різних сайтах. Однак політику того самого походження можна обійти, за певних умов, коли програма не виконує правильну перевірку вводу. У цих випадках вразливу програму можна обдурити на зберігання зловмисних програм коду від зловмисника, а потім представити цей зловмисний код користувачам. Так це буде виконано за припущення, що воно походить від вразливих програм, а не від зловмисника.

Існують різні форми XSS-атак, залежно від того, наскільки шкідливі подаються до вразливої програми та пізніше лунає з додатка для своїх користувачів. У непостійних (або відображених) атаках користувач має відвідати спеціально створене посилання, яке вказує на вразливу програму та вбудовує шкідливий код (наприклад, як значення параметра або назва ресурсу). Коли посилання активоване, вразлива мережа додаток негайно відображає код для користувача (наприклад, як частина помилки повідомлення). Потім код виконується в контексті вразливого сайту і має доступ до всієї інформації, пов’язаної з атакованим додатком, такі як файли cookie для автентифікації або інформація про сеанс. При постійних (або збережених) атаках шкідливий код спочатку зберігається вразливий додаток, а потім, пізніше, він представляється своїм користувачам. У цьому випадку безпека користувача порушується кожного разу, коли він / вона відвідує сторінку, вміст якої визначається за допомогою збереженого шкідливого коду. Типовими прикладами вразливих програм є програми гостьових книг або блог системи. Якщо вони дозволяють користувачам подавати записи, що містять скриптовий код, тоді вони вразливі до постійних XSS-атак.

Можлива і третя форма XSS-атак, яка називається DOM-заснованою. У цьому випадку вразлива програма представляє користувачам HTML-сторінку, яка використовує дані з частин його об'єктної моделі документа (DOM) небезпечними способами. DOM - це структура даних, яка дозволяє коду сценаріїв на стороні клієнта динамічно отримувати доступ та змінювати вміст, структуру та стиль HTML-документів. Деякі його властивості заповнюються

браузером на основі параметрів запиту, а не на основі характеристик самого документа. Наприклад, встановлено властивості document.URL та document.location на URL-адресу документа браузером. Якщо сторінка HTML містить код, що динамічно змінює зовнішній вигляд сторінки, використовуючи вміст document.URL (наприклад, щоб показати користувачеві URL-адресу, пов'язану зі сторінкою), можна використовувати шкідливо створену URL-адресу для виконання шкідливих сценаріїв код. Прикладом коду, вразливого до непостійних атак XSS, може бути знайдено в програмі PHP Advanced Transfer Manager (версія 1.30 та раніше). Уразливість міститься в наступному фрагменті коду.

$ font = $ _GET ['font'];

...

exo "<font face = \" $ font \ "color = \" $ normalfontcolor \ "

size = \ "1 \"> \ n ";

Змінна $ font знаходиться під контролем зловмисника, оскільки вона витягується із параметрів запиту та використовується для створення веб-сторінки повернутої користувачеві без будь-якої дезінфікуючої перевірки. Для використання цієї вразливості зловмисник може запитати таку URL-адресу:

http: // [target] / [path] /viewers/txt.php?font=

\% 22 \% 3E \% 3Cscript \% 3Ealert (document.cookie) \% 3C / script \% 3E

Як наслідок, вразливий додаток генерує таку веб сторінку:

<font face = ""> <script> попередження (document.cookie) </script>

При інтерпретації браузером виконується і сценарій коду у спливаючому вікні покаже файли cookie, пов'язані з поточною сторінкою. Очевидно, що справжня атака, наприклад, надсилає файли cookie зловмиснику.

1.1.4. Ін'єкція SQL

Веб-додаток має вразливість SQL-ін'єкції, коли вона використовує несаніфіковані дані користувача для складання запитів, які пізніше передаються реляційній базі даних для оцінки. Це може призвести до довільних запитів, що

виконуються в базі даних із привілеями вразливої програми. Наприклад, розглянемо такий фрагмент коду:

$ activate = $ _GET ["activate"];

*$ result = dbquery ("SELECT * FROM new_users".*

"WHERE user_code = '$ activate'");

if ($ result) {

...

}

Функція dbquery використовується для виконання запиту до внутрішньої бази даних і повернення результатів до програми. Запит складається динамічно шляхом порівняння статичного рядка з параметром, що надається користувачем. У цьому випадку активаційна змінна встановлюється на вміст омонімічного параметра запиту. Цільове використання змінної полягає в тому, щоб містити особистий код користувача динамічно складати вміст сторінки. Однак, якщо зловмисник подає а запит, де для параметра активації встановлено рядок ’АБО 1 = 1 – the Запит поверне вміст усієї нової таблиці користувачів. Якщо результат пізніше запит використовується як вміст сторінки, це відкриє особисту інформацію. Інші атаки, такі як видалення таблиць бази даних або додавання нових користувачі, також можливі.

1.1.5. Викрадення сесії

Більшість веб-додатків використовують протокол HTTP як протокол зв'язку. HTTP є протоколом без громадянства, тобто немає вбудованого механізму, який дозволяє додатку підтримувати стан протягом серії запитів, виданих одним і тим же користувачем. Однак практично всі нетривіальні програми потребують способу співвіднесення поточного запиту з історією попередніх запитів, тобто їм потрібен «сеансовий» погляд на їх взаємодію з користувачами. Наприклад, на веб-сайтах електронної комерції користувач додає до кошика товари, які він / вона має намір придбати, а згодом переходить до перевірки. Навіть якщо ці операції виконуються в окремих запитах, додаток повинен зберігати стан кошика користувача через усі запити, так щоб кошик міг

бути показаний користувачеві під час оформлення замовлення. Отже, було запроваджено низку механізмів забезпечення додатку з абстракцією сесій. Деякі мови забезпечують сесійні механізми на мовному рівні, інші покладаються на спеціальні бібліотеки. У інших у випадках, управління сесіями має здійснюватися на рівні програми.

Стан сесії може підтримуватися різними способами. Його можна закодувати в документі, переданому користувачеві таким чином, що він гарантуватиме, що інформація надсилається назад як частина пізніших запитів. Наприклад, HTML прихований Для цього можна використовувати поля форми. Ці поля не показуються користувач, але коли користувач подає форму, приховані змінні відправляються назад до програми як частини даних форми. У нашому прикладі електронної комерції додаток може зберегти поточний підсумок транзакції в прихованому вигляді поле. Коли користувач вибирає спосіб доставки, поле повертається до заявки та використовується для розрахунку остаточної загальної вартості. Штат може зберігатися в файлах cookie, що надсилаються в браузер користувача та автоматично повторно переглядається додатком при наступних відвідуваннях. Файли cookie можуть містити товари, в даний час вставлені в кошик користувача. Додаток, впродовж замовлення, шукає ціну кожного товару та представляє загальну вартість для користувачеві. Усі вищезазначені методи вимагають від клієнта співпраці з додатком для збереження стану сеансу. Інший підхід полягає в зберігання стану всіх сеансів на сервері. Тому кожному користувачеві призначено унікальний ідентифікатор сеансу, і це єдина інформація, яка надсилається назад і між додатком та користувачем, наприклад, за допомогою файлів cookie або подібного механізму, який переписує всі URL-адреси на сторінці, додаючи ідентифікатор сеансу як параметр. Як наслідок, кожен наступний запит буде включити ідентифікатор сеансу як параметр. Потім, щоразу, коли користувач подає запит на сайт, наприклад, щоб додати товар у кошик, отримує програма ідентифікатор сеансу, шукає відповідний сеанс у його сховищі та оновлює дані сесії відповідно до запиту. Було розроблено ряд атак на управління станом сесії механізми. Підходи,

що вимагають від клієнтів збереження стану, припускають, що клієнт не змінить стан сеансу, наприклад, змінивши приховане поле (або файл cookie), що зберігає поточну загальну суму для зниження ціни на предмет. Контрзаходи включають використання криптографічних прийомів для підписання параметрів та файлів cookie, щоб зробити їх стійкими до втручання. Більш загальною атакою є фіксація сеансу. Фіксація сеансу змушує користувача ідентифікатор сеансу з явним значенням на вибір зловмисника. Напад вимагає три кроки. Спочатку зловмисник налаштовує сеанс із цільовим додатком і отримує ідентифікатор сеансу. Потім зловмисник заманює жертву на доступ до цільової програми з використанням фіксованого ідентифікатора сеансу. Нарешті, зловмисник чекає поки жертва успішно не виконала всі необхідні аутентифікації та авторизаційних операцій, а потім видає себе за жертву, використовуючи фіксований ідентифікатор сеансу. Залежно від характеристик цільових веб-програм, для виправлення ідентифікатора сеансу можна використовувати різні методи. У найпростішому випадку зловмисник може просто заманити користувачів на вибір посилання, яке містить запит до програми з параметром, який визначає ідентифікатор сеансу, наприклад:

<a href = "http: // [target] /login.php?sessionid=1234">.

1.1.6. Розбиття відповіді

Розбиття відповіді HTTP - це атака, в якій зловмисник може встановити значення поля заголовка HTTP таке, що отриманий потік відповіді інтерпретується ціллю атаки як дві відповіді, а не одна. Розбиття відповідей є прикладом більш загальної категорії атак скористатися розбіжностями при синтаксичному аналізі, коли два або більше пристроїв або об'єктів обробляти потік даних між сервером і клієнтом. Щоб виконати розділення відповідей, зловмисник повинен мати змогу вводити дані містить символи закінчення заголовка та початок секунди заголовок. Зазвичай це можливо, коли дані користувача використовуються (несаніфіковані) для визначення значення заголовка HTTP. Ці умови зазвичай виконуються у ситуаціях, коли веб-програмам потрібно перенаправляти користувачів, наприклад, після входу в

систему процесу. Фактично, переспрямування здійснюється шляхом надсилання користувачеві відповідь із відповідним чином встановленими заголовками Location або Refresh.

Наступний приклад показує частину сторінки JSP, яка вразлива для атаки розбиття відповіді:

<%

answer. sendRedirect ("/ by_lang .jsp? lang =" +

param. getParameter ("lang"));

%>

Коли викликається сторінка, для визначення використовується параметр запиту lang ціль переспрямування. У звичайному випадку користувач передасть рядок, що представляє улюблена мова, скажімо en US. У цьому випадку генерується додаток JSP відповідь, що містить заголовок: Місцезнаходження: *http://vulvable.com/by_lang.jsp?lang=en_US*. Однак розглянемо випадок, коли зловмисник подає запит, де lang встановлено наступний рядок:

манекен% 0d% 0a

Довжина вмісту:% 200

% 0d% 0a% 0d% 0a

HTTP / 1,1% 20200% 20OK% 0d% 0a

Тип вмісту:% 20text / html% 0d% 0a

Довжина вмісту:% 2019% 0d% 0a% 0d% 0a

<html> Новий документ </html>

Сформована відповідь тепер міститиме кілька копій заголовків Content-Length і Content-Type, а саме ті, які вводить зловмисник на ті, що вставлені додатком. Як наслідок, залежно від деталей реалізації, проміжні сервери та клієнти можуть інтерпретувати відповідь, що містить два документи: оригінальний та підроблений зловмисником. У випадках нападу найчастіше згадується отруєння веб-кешем. Фактично, якщо кешуючий проксі-сервер інтерпретує потік відповідей, який містить два документи та пов'язує другий, підроблений зловмисником, з вихідний запит, тоді зловмисник зможе вставити в

кеш файлу проксі-сторінку за його вибором у поєднанні з URL-адресою вразливого застосування. Нещодавно підтримку розбиття контрастної реакції було запроваджено в деякі мови, особливо PHP. В інших випадках програміст відповідає за належне очищення даних, що використовуються для побудови заголовків відповідей.

1.2. Способи захисту веб-додатків від атак

1.2.1. Захист від SQL-ін'єкцій

Оновлювати всі програмні компоненти веб-додатків, включаючи бібліотеки, плагіни, платформи, програмне забезпечення веб-сервера і програмне забезпечення сервера бази даних, використовуючи останні доступні оновлення для системи.

Використовувати принцип найменших привілеїв у зовнішніх посиланнях при підготовці облікових записів, які використовуються для підключення до бази даних SQL. Наприклад, якщо веб-сайту потрібно тільки витягти веб-контент з бази даних за допомогою операторів SELECT, не треба надавати цьому обліковому запису інші привілеї, такі як INSERT, UPDATE або DELETE. У багатьох випадках цими привілеями можна управляти, використовуючи відповідні ролі бази даних для облікових записів. І ніколи не дозволяти веб-додатку підключатися до бази даних з правами адміністратора (наприклад, під обліковим записом «sa» на Microsoft SQL Server).

Налаштувати належні звіти про помилки та їх обробку на веб-сервері і в коді так, щоб повідомлення про помилки баз даних ніколи не відправлялися в веб-браузер клієнта. Тому що зловмисники можуть використовувати технічні деталі в повідомлень про помилки, щоб скорегувати свої запити для успішної експлуатації.

Використовувати escape-символи, щоб спеціальні символи ігнорувалися. Escape-символи – це просто засіб повідомити MySQL, що це не одиночна лапка, яка завершує рядок, а це частина самого рядка. Щоб MySQL знав, що це безпечно, це додати до нього символ зворотної косої межі. Використовувати брандмауер веб-додатків (WAF) для веб-додатків, які звертаються до баз даних.

Це може допомогти ідентифікувати спроби впровадження SQL, а іноді і запобігти попаданню спроб впровадження SQL на додаток.

1.2.2. Захист від XSS-атак

Перший метод це уникнення вводу користувача. Витік даних означає отримання даних, отриманих додатком, і забезпечення їх безпеки перед тим, як надавати їх кінцевому користувачеві. Уникаючи введення користувачем, ключових символів даних, отриманих веб-сторінкою, не зможуть інтерпретувати будь-який зловмисний спосіб. По суті, цензуруєте дані, які отримує ваша веб-сторінка, таким чином, що забороняє візуалізацію символів - особливо <та> символів, що в іншому випадку може завдати шкоди програмі та / або користувачам.

Якщо сторінка не дозволяє користувачам додавати свій власний код на сторінку, хорошим принциповим правилом є уникнення всіх сутностей HTML, URL та JavaScript. Однак, якщо веб-сторінка дозволяє користувачам додавати розширений текст, наприклад на форумах або розміщувати коментарі, є кілька варіантів вибору. Потрібно буде або ретельно вибрати, які HTML-сутності вдасться уникнути, а які ні, або використовуючи замінний формат для вихідного HTML-коду, такий як Markdown, що, своєю чергою, дозволить продовжувати екранувати весь HTML.

Все, що походить поза системою і не має абсолютного контролю над тим, що включає дані форми, рядки запитів, файли cookie, інші заголовки запитів, дані з інших систем (тобто веб-сервісів).

Перевірка введення - це процес забезпечення того, щоб програма надавала правильні дані та запобігала шкідливим даним завдавати шкоди сайту, базі даних та користувачам. Хоча білий список та перевірка вхідних даних частіше пов’язані з введенням SQL, їх також можна використовувати як додатковий метод запобігання XSS. Тоді як додавання до чорного списку або заборона певних, заздалегідь визначених символів у введенні користувачем забороняє лише відомі погані символи, білий список дозволяє лише відомі хороші символи і є кращим методом запобігання атакам XSS, а також іншим.

Перевірка вводу особливо корисна і хороша у запобіганні XSS у формах, оскільки заважає користувачеві додавати спеціальні символи в поля, а не відмовляти в запиті. Однак перевірка введення не є основним методом запобігання таким вразливим місцям, як введення XSS та SQL, а натомість допомагає зменшити ефекти, якщо зловмисник виявить таку вразливість.

Третій спосіб запобігти міжсайтовим атакам сценаріїв - це санітарна робота користувачів. Дезінфекція даних є надійним захистом, але не повинна використовуватися окремо для боротьби з атаками XSS. Цілком можливо, необхідність використовувати всі три методи запобігання, працюючи в напрямку більш безпечного застосування. Дезінфекція вводу даних особливо корисна на сайтах, які дозволяють розмітку HTML, щоб гарантувати, що отримані дані не можуть завдати шкоди користувачам, а також вашій базі даних, очищаючи дані від потенційно шкідливої розмітки, змінюючи неприйнятний ввід користувача на прийнятний формат.

1.2.3. Захист від CSRF-атаки

Найефективнішим методом захисту від CSRF є використання *токенів анти-CSRF*. Розробник повинен додати такі маркери до всіх форм, що дозволяють користувачам виконувати будь-які операції, що змінюють стан. Після подання операції веб-додаток повинен перевірити наявність правильного маркера. Іншим ефективним методом є використання атрибута SameSite для файлів cookie, але ще не всі браузери підтримують цей метод, приклад CSRF дивитись на рис. 1.2.

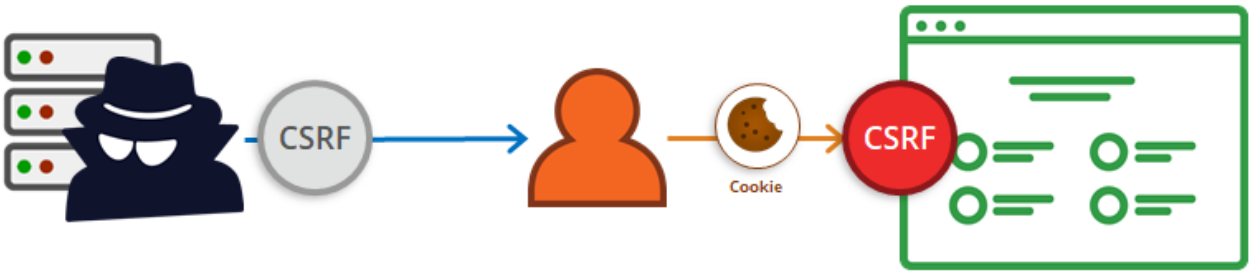

Рис. 1.2 Схема CSRF атаки

Файли cookie є внутрішньо вразливими до CSRF, оскільки вони автоматично надсилаються з кожним запитом. Це дозволяє зловмисникам легко створювати зловмисні запити, що ведуть до CSRF [26].

1.2.4. Захист від Broken authentication атаки

Там, де це можливо, впроваджуйте багатофакторну автентифікацію, щоб запобігти автоматизованим, заповненням облікових даних, грубою силою та викраденими атаками повторного використання облікових даних. Не постачайте та не розгортайте жодні облікові дані за замовчуванням, особливо для користувачів адміністратора.

Впровадити перевірку слабких паролів, наприклад, тестування нових або змінених паролів у списку з 10000 найгірших паролів. Поєднайте політику довжини, складності та ротації паролів із вимогами NIST 800-63

Переконайтеся, що реєстрація, відновлення облікових даних та шляхи API захищені від атак перерахування облікових записів, використовуючи однакові повідомлення для всіх результатів.

Обмежте або все частіше затримуйте невдалі спроби входу. Записуйте всі помилки та попереджуйте адміністраторів, коли виявляються набивання облікових даних, груба сила чи інші атаки.

Використовуйте захищений вбудований менеджер сеансів на стороні сервера, який генерує новий ідентифікатор випадкової сесії з високою ентропією після входу.

Ідентифікатори сеансів не повинні знаходитись в URL-адресі, надійно зберігатись та втратити силу після виходу з режиму очікування, простою та абсолютного часу очікування.

1.2.5. Захист від атаки неправильної конфігурації

Щоб забезпечити такий захист потрібно:

- Зменшити поверхню уразливості за допомогою повторюваного процесу;
- Тримати програмне забезпечення в актуальному стані;
- Вимикати всі облікові записи за промовчанням і регулярно змінювати;
- Розробити сильну архітектуру додатків і шифрувати дані, які мають конфіденційну інформацію;

- Переконатися, що параметри безпеки в рамках і бібліотеках встановлено на захищені значення;
- Виконувати регулярні перевірки та виконуйте інструменти для ідентифікації отворів у системі;
- Використовувати ту ж конфігурацію для виробництва, розробки і постановки, оскільки невідповідності відкривають доступ для багатьох неправильних конфігурацій;
- Автоматизувати систему, де це можливо, щоб уникнути людських помилок;
- Переконатися, що програмне забезпечення - включаючи операційні системи, програми, системи керування базами даних та веб- або сервери додатків – оновлені;
- Перевірити, чи немає жодних облікових записів за промовчанням і чи були змінені їхні паролі. Облікові записи за промовчанням обов'язково викликають проблеми;
- Перевірити чи встановлено непотрібні або потенційно небезпечні функції. Наприклад, програма може мати функцію налагодження, яка може дозволити зловмисникам обійти аутентифікацію для доступу до конфіденційної інформації.
- Проводити планові сканування уразливостей та перевірки безпеки, щоб своєчасно зловити неправильне конфігурування.

1.2.6. Захист від атаки витоку критичних даних

Забезпечувати шифрування даних та впроваджувати перевірену техніку шифрування. Шифрувати дані та визначати доступність: дуже важливо шифрувати дані, інформацію в формі, що зберігається, або в транзиті. Зберігати конфіденційні дані зашифрованими увесь час. Тому що такі дані не повинні зберігатися і не передаватися в чистому текстовому форматі. Будь-які дані у вигляді відкритого тексту є прямим запрошенням для нападників. Визначення даних, які потребують додаткового захисту, і обмеження доступності лише до

групи законних користувачів лише шляхом застосування шифрування на основі ключів [27].

Приклад № 1: Шифрування кредитної картки додаток шифрує номери кредитних карт у базі даних за допомогою автоматичного шифрування баз даних. Однак це означає, що він також розшифровує ці дані автоматично після вилучення, дозволяючи недоліку ін'єкції SQL отримувати номери кредитних карт в прозорому тексті. Тому система повинна мати зашифровані номери кредитних карток, використовуючи відкритий ключ, і дозволятиме розшифровувати їх за допомогою закритого ключа. Використовувати шлюзи безпечної аутентифікації. Захистити сайт за допомогою захищеного протоколу HTTPS (SSL / TLS), щоб переконатися, що всі дані, що передаються між браузером і веб-сервером, зашифровані і залишаються приватними. SSL використовує пару публічних / приватних ключів для передачі даних.

Приклад № 2: SSL не використовується для всіх сторінок, які пройшли аутентифікацію зловмисник просто відстежує мережевий трафік (наприклад, відкриту бездротову мережу) і викрадає cookie сесії користувача. Після цього зловмисник повторює цей файл cookie і захоплює сеанс користувача, отримуючи доступ до особистих даних користувача. Потрібно впровадити сильний алгоритм хешування паролів. Зловмисники можуть використовувати слабкість алгоритму хешування паролів для крадіжки конфіденційної інформації, що зберігається на веб-сервері або сервері додатків. Для реалізації хешування паролів слід використовувати лише криптографічні хеш-функції. [7]

Приклад №3: База даних паролів використовує несолідовані хеши для зберігання паролів кожного користувача Пошкодження файлу дозволяє зловмисникам отримувати файл паролів. Всі несолодкі хеші можуть бути виставлені за допомогою веселки з попередньо розрахованими хешами. Важливо відзначити, що навіть шифрування має свої недоліки, тому використання застарілих або слабких криптографічних алгоритмів не є можливим, оскільки це лише надає помилкове почуття безпеки. Те ж саме стосується простих хешей, які можна повернути. Дуже важливо забезпечити сучасні і сильні стандартні

алгоритми, протоколи і ключі, які використовуються, а також належним чином керуватися керуванням ключами.

1.2.7. Захист від атаки відсутності контролю рівня доступу на функціональному рівні

Програма повинна мати послідовний і простий для аналізу модуль авторизації, який викликається з усіх бізнес-функцій. Часто такий захист забезпечується одним або декількома компонентами, зовнішніми до коду програми. Рекомендується завжди застосовувати правило заборони за замовчуванням. За замовчуванням заборонити доступ до всіх функцій програми, а потім дозволити доступ тільки тим користувачам та іншим частинам програми, які цього потребують. Навіть коли дозволений доступ до функцій в веб-додатку, кожен запит повинен бути перевірений під час доступу. Потрібно перевірити, що запити від дійсних авторизованих користувачів. Використовувати списки контролю доступу і перевірку автентичності на основі ролей, щоб забезпечити дотримання вищесказаного. Використовувати принцип найменших привілеїв. Тобто надавати доступ до функцій тільки при необхідності. Не намагатись надавати загальний доступ, а потім видаляти права доступу у користувачів, у яких його не повинно бути. Не намагатися покладатися на безпеку через неясність, просто приховуючи кнопки і посилання на функції в інтерфейсі. Хтось наткнеться на спосіб доступу до прихованих функцій. Крім того, люди, які мають намір атакувати ваш веб-додаток, будуть ігнорувати призначений для користувача інтерфейс і відправляти запити безпосередньо, щоб спробувати отримати відповіді від внутрішнього застосування і механізмів баз даних. Потрібно перевіряти всі URL, кнопки та інші способи доступу до функцій в вашому веб-додатку, використовуючи обліковий запис з низькими привілеями. Варто подивитися, чи можуть ці облікові записи отримати доступ до функцій, які вони не повинні. Є інструменти, які можуть допомогти з цим завданням. Вони порівнюють перегляди ваших веб-додатків при аутентифікації як користувача з правами адміністратора і зі скануванням при аутентифікації як звичайних користувачів. Потім вони виділяють частини програми, які доступні звичайним

користувачам, коли вони не повинні бути. Більшість веб-додатків не відображає посилання та кнопки для несанкціонованих функцій, але цей "контроль доступу до презентаційного рівня" фактично не забезпечує захисту. Необхідно також здійснювати перевірки контролера або бізнес-логіки. Різні типи невизначеності ускладнюють розвідувальне співтовариство для ефективного аналізу мінливий характер загрози та ступінь ризику. І ці невизначеності пов’язані з властивими кіберзагрозами ─ характеристиками, якими вони поділяються з цілим набором «нових» загроз безпеці. [8]

1.3. Кіберзагрози та їх наслідки

Здатність урядів оцінювати загрози критично важливим інфраструктурам традиційно обумовлюється їх здатністю оцінити намір зловмисного актора та здатність цього актора здійснити навмисне дії. Це було значно легше під час "холодної війни", коли влада була просто стурбована безпекою фізичних конструкцій. Через глобальний характер інформаційних мереж атаки можуть розпочати з будь-якої точки світу, і виявлення походження атак залишається важливою справою труднощі, якщо, дійсно, їх взагалі виявляють. У порівнянні з традиційним аналізом загроз безпеці, який складається з аналізу акторів, їхніх намірів та їх можливостей, кіберзагрози мають різні особливості, що ускладнюють такі напади, аналіз та протидію:

Анонімність суб’єктів: проблема ідентифікації суб’єктів є особливо складною в області де зберегти анонімність легко і де між тим, що відбувається, є проміжки часу зловмисник приймає, саме вторгнення та наслідки вторгнення. Крім того, постійне розповсюдження сучасних комп’ютерних технологій серед основних потоків населення ускладнює ідентифікацію суб'єктів.

Відсутність меж: зловмисні комп’ютерні атаки не обмежуються політичними або географічні межі. Напади можуть відбуватися з будь-якої точки світу і з багатьох локацій одночасно. Розслідування, що слідують за низкою свідомо побудованих помилкових потенційні клієнти можуть зайняти багато часу та вимагати ресурсів. Швидкість розвитку: Технологія розвивається

надзвичайно швидко. Час між виявлення нової вразливості та поява нового інструменту чи техніки, що використовуються що вразливість стає коротшою.

Низька вартість інструментів: технологія, що використовується в таких атаках, проста у використанні, недорога і широко доступна. Інструменти та техніки для вторгнення в комп'ютери доступні на комп'ютері дошки оголошень та різних веб-сайтів, а також інструменти шифрування та анонімності.

Автоматизовані методи: все частіше методи атаки стають автоматизованими та складнішими, що призводить до більшої шкоди від однієї атаки. Ці характеристики значно перешкоджають спроможності прогнозувати певні несприятливі сценарії майбутнього. Різні типи невизначеності ускладнюють розвідувальне співтовариство для ефективного аналізу мінливий характер загрози та ступінь ризику. І ці невизначеності пов'язані і властиві кіберзагрозам ─ характеристики, якими вони поділяються з цілим набором «нових» загроз безпеці. Кінець "холодної війни" означав не тільки кінець відносно стабільного біполярного світового порядку, але й кінець обмеженості загроз. Після розпаду Радянського Союзу різноманітні "Нові", а часто і невійськові загрози, такі як міграція, тероризм, розповсюдження та ін., Були переміщені на порядок денний політики безпеки. Хоча ярлик "новий" у більшості випадків не виправданий, багато хто ці загрози суттєво відрізняються від погроз безпеці холодної війни. і полягає в тому, що головний з них новими загрозами є ті, що виходять від недержавних суб'єктів, що використовують невійськові засоби. Будь-яка комбінація загрози за участю або невійськових - або асиметричних - засобів та / або недержавних суб'єктів значні труднощі для традиційних підходів до збору розвідувальних даних: зв'язок здатності до намірів добре працює лише тоді, коли зловмисників чітко помітні, а спецслужби можуть зосередитись колективні зусилля, щоб визначити, якими можливостями вони володіють або намагаються набути їх. Хоча напад іншої держави з нетрадиційними засобами та принаймні чітко призначеним агентством робить можливі військові варіанти, недержавні актори повністю грають поза "коробкою" Вестфалії державне замовлення. Невизначеність оточує особу та цілі

цих потенційних супротивників, часові рамки в межах яких можуть виникнути загрози та непередбачені ситуації, які можуть бути покладені на державу та інші. Крім того, існує невизначеність щодо можливостей, проти яких треба готуватися, а також про те, до якого типу конфлікту готуватися. Експерти не можуть передбачити, якою ймовірно, кібер-атака насправді є. З іншого боку, попри те, що кіберзагрози ще не матеріалізувались як “реальні”, тривають дебати створює значний тиск для тих, хто приймає рішення. Для багатьох урядів ці рішення дійсно були прямолінійними. Вони вважають загрозу національній безпеці реальною і, отже, склали чи навіть здійснили низку кроків для протидії цьому. Однак необґрунтованість кіберзагроз означає, що дискусія стосується не лише прогнозування майбутнього, але й того, як підготуватися до можливих непередбачених ситуацій в даний час. Оскільки для серйозних руйнівних атак на кіберрівні рішення повинні прийматися на основі різних сценаріїв. Різні актори - від державних установ, технологічного співтовариства до страхових компаній - мають різні інтереси і конкурують між собою за допомогою побудованих версій майбутнє.

Тоді вибір політики в значній мірі залежить від двох факторів: один – рівномірна доступність до ресурсів різних груп. Інший фактор - результат культурно-правовових норм, оскільки вони обмежують кількість потенційних стратегій, доступних для відбору. Зосередження уваги на сприйняття загрози може показати нам, чому певні контрзаходи вважаються більш придатними, ніж інші. Наприклад, ми бачимо, що типи інституційних рішень різняться залежно від того, чи держава вважає, що інші держави, а точніше різні недержавні суб'єкти представляють суб'єкт загроз. Якщо суб'єкт загрози сприймається державними акторами, держава, як правило, зосереджується на стратегічних питаннях відповіді військових. Коли загроза сприймається головним чином від міждежавних суб'єктів, воля держави, як правило, зосереджуються насамперед на реакції правоохоронних органів. Ми звернемося до цих контрзаходів у наступному розділі.

1.4. Висновки до розділу

Захист веб-додатків від зловмисників залежить від технологій і компонентів, що використовуються при створенні веб-додатків, а також від можливих уразливостей цих компонентів. Існують різні класифікації вразливостей, кожна атака через вразливість має свої особливості, але причиною вразливостей є помилки в розробці, реалізації та застосуванні компонентів веб-додатків, звідси необхідність пошуку уразливостей і реагування на інформацію про їх розташування. Як в Україні, так і в інших країнах світу організовуються групи з реагування на надзвичайні ситуації, які складаються з експертів і дослідників. Та деякі міжнародні стандарти регулюють процес розкриття уразливості. Широкий спектр інструментів дозволяє здійснювати пошук уразливостей, але ефективність їх використання залежить від алгоритму дій, які необхідно здійснити цим пошуком. Алгоритми дій можуть бути представлені у вигляді таких спеціальних методів, що охоплюють широке коло питань кібербезпеки, такі як тестування безпеки фізичного середовища, операційних систем, бездротових мереж, тобто потрібен додатковий час для аналізу існуючих методів і вибору таких компонентів. Тому існує потреба в розробці такої методології тестування проникнення, яка б враховувала міжнародні досягнення у тестуванні веб-додатків і містила б перелік можливих інструментів тестування.

Розділ 2. СУЧАСНІ МЕТОДИ ШИФРУВАННЯ ДЛЯ РОЗРОБКИ МОДУЛЮ ЗАХИСТУ ІНФОРМАЦІЇ ВІД МЕРЕЖЕВИХ АТАК НА ВЕБ-ДОДАТКИ

2.1 Нормативні документи із захисту інформації

У цьому розділі наводиться список законодавчих актів, нормативно-правових актів та нормативних актів щодо інформаційної безпеки стосовно захисту інформації в Україні. Нині загальна кількість нормативно-правових актів, що регламентують діяльність в сфері технічного захисту інформації в нашій державі, складає близько 150 одиниць.

Основу державного регулювання суспільних відносин в сфері ТЗІ становить Конституція України. До спеціального законодавства, що врегульовує дану галузь, належать Закони України: «Про інформацію», «Про державну таємницю», «Про захист інформації в автоматизованих системах», «Про Національну систему конфіденційного зв'язку», «Про Концепцію Національної програми інформатизації», «Про науково- технічну інформацію» та інші. У тому числі у сфері правоохоронної діяльності — такі, що визначають компетенцію та функції окремих державних органів влади: Служби безпеки України, поліції, прокуратури тощо (закони України «Про Службу безпеки України», «Про поліцію» та інші). Деякі вимоги щодо технічного захисту окремих видів інформації містяться у законодавстві України про оперативно-розшукову діяльність, про організаційно-правові основи боротьби з організованою злочинністю. [9]

Державна політика України у сфері захисту інформації від її витоку технічними каналами знаходить відображення у системі підзаконних нормативно-правових актів органів державної влади, що видаються останніми відповідно до їх компетенції, функцій, прав і обов'язків. Дану систему складають Укази та Розпорядження Президента України, нормативно-правові акти Кабінету Міністрів України, нормативні акти Служби безпеки України, інших міністерств і відомств. Згідно з Положенням про технічний захист інформації в Україні в КС, де обробляється інформація, яка є власністю держави або захист якої гарантується державою, повинні використовуватись засоби ТЗІ, які мають

документ, що засвідчує їх відповідність вимогам нормативних документів з питань технічного захисту інформації (експертний висновок та/або сертифікат відповідності). Склад засобів ТЗІ, що використовуються під час створення комплексу засобів захисту (КЗЗ) інформації, визначають власники КС, де обробляється інформація, яка підлягає захисту, або уповноважені ними суб'єкти системи ТЗІ, з урахуванням того, що ці засоби повинні мати рівень гарантій коректності реалізації послуг безпеки (НД ТЗІ 2.5-004-99) не нижчий від рівня гарантій створюваного КЗЗ. [9]

Дозволяється в АС класів ";1"; та ";2"; (НД ТЗІ 2.5-005-99) використання засобів ТЗІ з рівнем гарантій на один нижче від рівня гарантій створюваного КЗЗ, за умов реалізації в цих АС необхідного обсягу організаційних заходів. Обсяг цих заходів визначається моделями загроз та порушника, умовами експлуатації АС тощо. [9]

Засоби криптографічних перетворень, які є складовою частиною засобів ТЗІ, повинні відповідати вимогам нормативних документів з питань криптографічного захисту інформації.

Державна політика України у сфері захисту інформації від її витоку технічними каналами знаходить відображення у системі підзакон- них нормативно-правових актів органів державної влади, що видаються останніми відповідно до їх компетенції, функцій, прав і обов'язків. Дану систему складають Укази та Розпорядження Президента України, нормативно-правові акти Кабінету Міністрів України, нормативні акти Служби безпеки України, інших міністерств і відомств.

1. Закони України:
    - Закон України «Про інформацію» від 02.10.1992 № 2657-XII
    - Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» від 05.07.1994 № 80/94-ВР
2. Постанови КМУ:
    - Постанова Кабінету міністрів України «Про затвердження Інструкції про порядок обліку, зберігання і використання документів, справ,

видань та інших матеріальних носіїв інформації, які містять службову інформацію» від 27 листопада 1998 р. №1893

3. Нормативні документи в галузі технічного захисту інформації та державні стандарти України (ДСТУ) стосовно створення і функціонування КСЗІ:
    - НД ТЗІ 3.7-003-05 Порядок проведення робіт із створення комплексної системи захисту інформації в інформаційно-телекомунікаційній системі
    - Державний стандарт України. Захист інформації. Технічний захист інформації. Порядок проведення робіт. ДСТУ 3396.1-96
    - НД ТЗІ 1.4-001-2000 Типове положення про службу захисту інформації в автоматизованій системі

2.2 Шифрування методом RSA

У попередньому розділі було розглянуто симетричне шифрування з ключем, тобто один і той же ключ повинен бути відомий як відправнику, так і одержувачу, шифрувати і дешифрувати повідомлення відповідно. Було зроблено висновок, що він повинен бути відомий обом з них перед відправкою першого зашифрованого повідомлення. У минулому, таким чином, перед тим, як увійти в зашифрований текст, вони повинні були зустрітися і особисто погодитися з паролем або методом шифрування. Але що робити коли людина не може задовольнити або надійно погодитися з паролем? Рішенням є система з асиметричними ключами, в яких під час шифрування та дешифрування використовується інший ключ. Зазвичай така система має два типи ключів: відкритий ключ, який робить його доступним для всіх бажаючи, наприклад, на вашій веб-сторінці, а інший приватний, хто зберігає його в таємниці. Використовуючи відкритий ключ, бажаємо надіслати людині, з якою ми хочеми співпрацювати. Власник приватного ключа може з його допомогою розшифрувати повідомлення. Весь секрет полягає в тому, що відкритий ключ не може розшифрувати зашифроване повідомлення, можна зробити це тільки знаючи приватний ключ. Прикладом такої системи є RSA. Офіційно

представлена в 1979 році це робота трьох людей: Рональда Рівеста, Аді Шаміра і Леонарда Адлемана - його назва походить від перших букв імен її творців.

Шифрування RSA - це система, яка вирішує те, що колись було однією з найбільших проблем у криптографії: як можна надіслати комусь повідомлення, якщо не має можливості раніше поділитися кодом з іншими користувачами? Припустимо, ви хочете повідомити своєму другові секрет. Якщо ви прямо поруч з ними, ви можете просто прошепотіти. Якщо ви знаходитесь в протилежних частинах країни, це, очевидно, не буде працювати. Ви можете записати їх і надіслати їм поштою, або скористатися телефоном, але кожен з цих каналів зв'язку є небезпечним і кожен, хто має достатньо сильну мотивацію, може легко перехопити повідомлення. Якби секрет був достатньо важливим, ви не ризикували б писати його належним чином - шпигуни або працівник шахрайських поштових служб могли б переглядати вашу пошту. Крім того, хтось може торкатися вашого телефону без вашого відома і реєструвати кожен виклик. Одним з рішень, що запобігають доступу до вмісту повідомлень, є шифрування. Це в основному означає додавання коду до повідомлення, яке змінює його на змішаний безлад. Якщо ваш код достатньо складний, то єдині люди, які зможуть отримати доступ до початкового повідомлення - це ті, хто має доступ до коду. Якщо ви мали нагоду поділитися кодом з вашим другом заздалегідь, то будь-який з вас може відправити зашифроване повідомлення у будь-який час, знаючи, що ви є єдиними, хто має можливість читати вміст повідомлення. Але що, якщо у вас не було можливості поділитися кодом заздалегідь? Це одна з фундаментальних проблем криптографії, яку розглядали схеми шифрування публічних ключів (також відомі як асиметричне шифрування), як RSA. Під шифруванням RSA повідомлення шифруються кодом, який називається відкритим ключем, який може бути відкритим. Через певні математичні властивості алгоритму RSA, коли повідомлення було зашифровано відкритим ключем, його можна розшифрувати лише іншим ключем, відомим як приватний ключ. Кожен користувач RSA має пару ключів, що складається з їхніх публічних і приватних ключів. Як випливає з назви, приватний ключ повинен зберігатися в

таємниці. Схеми шифрування відкритих ключів відрізняються від шифрування симетричного ключа, коли процес шифрування та дешифрування використовує той же закритий ключ. Ці відмінності роблять шифрування відкритим ключем, подібним до RSA, корисним для спілкування в ситуаціях, коли не було можливості безпечно поширювати ключі заздалегідь. Алгоритми симетричного ключа мають свої власні програми, такі як шифрування даних для особистого користування, або коли існують захищені канали, через які приватні ключі можуть спільно використовуватись. (див. рис. 2.1.)

Рис. 2.1. Схема роботи методу RSA

Шифрування RSA часто використовується в поєднанні з іншими схемами шифрування, або для цифрових підписів, які можуть довести автентичність і цілісність повідомлення. Зазвичай він не використовується для шифрування цілих повідомлень або файлів, оскільки він менш ефективний і більш ресурсомісткий, ніж шифрування симетричного ключа. Щоб зробити речі більш ефективними, файл зазвичай шифрується алгоритмом симетричного ключа, а потім симетричний ключ шифрується за допомогою шифрування RSA. Відповідно до цього процесу, лише об'єкт, який має доступ до закритого ключа RSA, зможе розшифрувати симетричний ключ.

Не маючи доступу до симетричного ключа, оригінальний файл не можна розшифрувати. Цей метод може використовуватися для збереження надійності повідомлень і файлів, без зайвого затримки або зайвого використання численних обчислювальних ресурсів. Шифрування RSA може використовуватися в ряді

різних систем. Він може бути реалізований в OpenSSL, wolfCrypt, cryptlib і ряді інших криптографічних бібліотек. Як одна з перших широко використовуваних схем шифрування з відкритим ключем, RSA заклала основу для більшої частини наших безпечних комунікацій. Він традиційно використовувався в TLS і був також оригінальним алгоритмом, який використовувався в шифруванні PGP . RSA все ще спостерігається в ряді веб-браузерів, електронної пошти, VPN, чату та інших каналів зв'язку. RSA також часто використовується для забезпечення безпечного з'єднання між клієнтами VPN і VPN серверами. У протоколах, таких як OpenVPN, TLS рукостискання може використовувати алгоритм RSA для обміну ключами та встановлення захищеного каналу. [10]

Перед шифруванням із публічним ключем, важко безпечно спілкуватися, якщо раніше не було можливості безпечно обміняти ключі. Якщо не було можливості поділитися кодом заздалегідь, або безпечним каналом, за допомогою якого ключі могли бути розподілені, не було можливості спілкуватися без загрози перехоплення і доступу до вмісту повідомлення.

Перший серйозний розвиток до того, що ми зараз називаємо криптографією з відкритим ключем, був опублікований на початку десятиліття Джеймсом Еллісом. Елліс не знайшов способу реалізувати свою роботу, але його колега Кліффорд Кокс розширилася, щоб стати тим, що ми тепер знаємо як шифрування RSA. Остаточна частина головоломки - це те, що ми зараз називаємо обміном ключами Діффі-Хеллмана. Ще один співробітник Malcolm J. Williamson з'ясував схему, яка дозволила двом учасникам поділитися ключем шифрування, навіть якщо канал контролювався противниками. Вся ця робота була проведена в британській спецслужбі, штаб-квартирі уряду (GCHQ), який зберігав класифікацію відкриттів. Частково через технологічні обмеження, GCHQ не міг бачити використання для криптографії публічних ключів у той час, так що розробка неквапливо стояла на полиці, збираючи пил. Тільки до 1997 року робота була розсекречена, і первісні винахідники RSA були визнані. Кілька років по тому подібні концепції почали розвиватися в публічній сфері. Ральф Меркле створив ранню форму криптографії з відкритим ключем, що вплинуло на Вітфілд

Діффі та Мартін Хеллман у розробці обміну ключами Діффі-Хеллмана. У ідеях Діффі та Хеллмана не вистачає одного важливого аспекту, який зробив би їх роботу фундаментом криптографії відкритих ключів. Це була одностороння функція, яку було б важко інвертувати. У 1977 році Рон Рівест, Аді Шамір і Леонард Адлеман, чиї прізвища утворюють абревіатуру RSA, вирішили після року роботи над проблемою. Науковці на базі MIT зробили свій прорив після Пасхальної партії в 1977 році. Після ночі пиття Рівест пішов додому, але замість того, щоб спати, він гарячково списував папір, який формалізував його ідею про необхідність односторонньої функції. Ідея була запатентована в 1983 році компанією MIT, але алгоритм RSA почав бачити широке впровадження як важливий інструмент безпеки ще до початку інтернету. Щоб утримати математику від надмірної втрати руки, ми спростимо деякі концепції та використаємо набагато менші числа. Насправді, шифрування RSA використовує прості числа, які набагато більші за величиною, і є кілька інших складностей. Існує кілька різних концепцій, які вам доведеться обдурити, перш ніж ми зможемо пояснити, як все це збігається. До них відносяться дверні функції, генерування простих чисел, функція коефіцієнту Кармайкла і окремі процеси, пов'язані з обчисленням публічних і приватних ключів, що використовуються в процесах шифрування і дешифрування.

Можливість шифрувати номер 4 не здається особливо корисним, тому ви можете ставити собі питання, як можна шифрувати більш складний набір даних, наприклад, симетричний ключ (який є найбільш поширеним використанням RSA), або навіть повідомлення.

Деякі люди можуть бути збентежені тим, як ключ, як "n38cb29fkbjh138g7fqijnf3kaj84f8b9f ..." або повідомлення, як "купити мені бутерброд", може бути зашифрований алгоритмом, як RSA, який займається номерами, а не літерами. Реальність така, що вся інформація про те, що процес нашого комп'ютера зберігається в двійкових (1s і 0s), і ми використовуємо стандарти кодування, такі як ASCII або Unicode, щоб представляти їх так, як люди можуть розуміти (букви).

Це означає, що ключі, такі як «n38cb29fkbjh138g7fqijnf3kaj84f8b9f…» і повідомлення, такі як «купити мені бутерброд», вже існують у вигляді чисел, які можна легко обчислити в алгоритмі RSA. Цифри, якими вони представлені, нам набагато більше і важче керувати, тому ми вважаємо за краще мати справу з буквено-цифровими символами, а не з двійковою сумішшю. Якщо ви хочете зашифрувати більш тривалий сеансовий ключ або більш складне повідомлення з RSA, це просто передбачає набагато більшу кількість.

Вищезгадані функції дверцят утворюють основу того, як працюють схеми шифрування публічних і приватних ключів. Їх властивості дозволяють спільно використовувати відкриті ключі, не піддаючи небезпеці повідомлення або розкриваючи закритий ключ. Вони також дозволяють шифрувати дані одним ключем таким чином, що їх можна розшифрувати лише іншим ключем від пари. Першим кроком шифрування повідомлення за допомогою RSA є генерування ключів. Для цього потрібні два простих числа (p і q), які вибираються за допомогою тесту простоти.

Прості числа в RSA повинні бути дуже великими, а також відносно далеко один від одного. Цифри, які є невеликими або ближче один до одного, набагато легше зламати. Незважаючи на це, приклад буде використовувати менші номери, щоб полегшити слідування та обчислення.

Припустимо, що тест простоти дає нам прості числа, 907 і 773. Наступним кроком є виявлення модуля (n), використовуючи наступну формулу:

$n = p$ x q

Де p = 907 і q = 773

Тому:

n = 907 x 773

n = 701,111

Коли ми маємо n, будемо використовувати функцію коефіцієнта Кармайкла

$\lambda (n) = lcm (p - 1, q - 1)$

Якщо пройшов деякий час, вищезазначене може виглядати трохи жахливо. Значення λ (n) являє собою коефіцієнт Кармайкла для n , тоді як lcm означає

найменший загальний кратний , який є найменшим числом, на яке і p і q можуть розділити.

λ (701,111) = lcm (907 - 1, 773 - 1)

λ (701,111) = lcm (906, 772)

Настав час, щоб з'ясувати відкритий ключ. У RSA відкриті ключі складаються з простого числа e, а також n. Числом e може бути щось між 1 і значенням λ (n).

Оскільки відкритий ключ відкритий, для *e* не так важливо, щоб він був випадковим числом. На практиці, *e*, як правило, встановлюється на рівні 65537, оскільки коли набагато більші числа вибираються випадковим чином, це робить шифрування набагато менш ефективним. Для прикладу збережемо цифри, щоб зробити обчислення ефективними. Скажімо:

e = 11

Наші остаточні зашифровані дані називаються зашифрованим текстом (c). Ми виводимо його з нашого текстового повідомлення (m), застосовуючи відкритий ключ за такою формулою:

$c = m^e \bmod n$

Ми вже придумали e, і ми також знаємо n. Єдине, що нам потрібно пояснити, це мод. Це трохи з глибини цієї статті, але це відноситься до операції по модулю, яка по суті означає залишок, що залишився, коли ви розділяєте одну сторону на іншу. Наприклад:

$10 \bmod 3 = 1$

Це відбувається тому, що 3 переходить в 10 разів три, з рештою 1. Повертаємося до рівняння. Для того, щоб зробити речі простими, скажімо, що повідомлення (m), яке ми хочемо зашифрувати та зберегти в таємниці, - це лише одне число, 4. Давайте підключаємо все:

$c = m^e \bmod n$

$c = 4^{11} \bmod 701{,}111$

$c = 4{,}194{,}304 \bmod 701{,}111$

Тому, коли ми використовуємо RSA для шифрування нашого повідомлення, 4, з нашим відкритим ключем, це дає нам зашифрований текст 688,749. Попередні кроки, можливо, здавалися занадто важкими, але важливо повторити те, що відбулося.

У нас було повідомлення 4, яке ми хотіли зберегти в таємниці. Ми застосували до неї відкритий ключ, який дав нам зашифрований результат 688,749. Тепер, коли вона зашифрована, ми можемо надійно надіслати номер 688 749 власникові пари ключів. Вони є єдиною людиною, яка зможе розшифрувати її за допомогою приватного ключа. Коли вони розшифрують його, вони побачать повідомлення, яке ми дійсно посилаємо, 4. [11]

У шифруванні RSA, коли дані або повідомлення були перетворені в зашифрований текст з відкритим ключем, його можна розшифрувати лише приватним ключем з тієї ж пари ключів. Приватні ключі складаються з d і n. Ми вже знаємо n, і наступне рівняння використовується для знаходження d:

d = 1 / e mod λ (n)

У розділі Генерація відкритого ключа ми вже вирішили, що в нашому прикладі e буде дорівнювати 11. Аналогічно, ми знаємо, що λ (n) дорівнює 349 716 з нашої попередньої роботи під функцією Кармайкла. Речі стають дещо складнішими, коли ми зустрічаємо цей розділ формули:

1 / e mod

Це рівняння може виглядати так, як ви просите ділити 1 на 11, але це не так. Замість цього, це просто символізує, що нам потрібно обчислити модульну інверсію e (яка в даному випадку становить 11) і λ (n).

d = 1/11 мод 349,716

Щоб виконати цю операцію, просто введіть 11 (або будь-яке значення, яке ви можете мати для e, якщо намагаєтеся це з вашим власним прикладом), де говориться про Integer і 349,716 (або будь-яке значення, яке можете мати для λ (n), якщо намагаєтеся це з вашим власним прикладом)

d = 254, 339

Тепер, коли маємо значення d, ми можемо розшифровувати повідомлення, зашифровані нашим відкритим ключем, використовуючи наступну формулу:

$m = c^{d} \bmod n$

Тепер можемо повернутися до зашифрованого тексту, який шифрували під розділом Генерування закритого ключа. Коли ми шифрували повідомлення відкритим ключем, це дало нам значення для 688,749. З вищесказаного відомо, що d дорівнює 254,339. Відомо також, що n дорівнює 701,111. Це дає нам:

$m = 688{,}749^{254{,}339}$ мод 701,111.

Спроби взяти номер до 254 ,339-й потужності можуть бути трохи більшими для більшості звичайних калькуляторів. Якщо ви хочете скористатися іншим методом, ви повинні застосувати повноваження, як це було б зазвичай, і виконуєте операцію модуля таким же чином, як ми це зробили в розділі " Генерація відкритого ключа ".

RSA шифрування працює за умови, що алгоритм легко обчислити в одному напрямку, але практично неможливо в зворотному напрямку. Як приклад, якщо вам сказали, що 701.111 є продуктом двох простих чисел, ви могли б з'ясувати, що таке ці два числа? Навіть з калькулятором або комп'ютером, більшість з нас не буде мати ніякої ідеї, з чого почати, не кажучи вже про можливість з'ясувати відповідь. Але якщо ми обертаємося навколо, то стає набагато простіше. Який результат:

907 x 773

Якби було досить нудно, ви могли б витягти телефон або розрахувати його в голові, щоб виявити, що відповідь - це згадане раніше 701,111. Це 907 і 773 є простими числами, які відповідають нашому першому питанню, що показує нам, що певні рівняння можуть бути легко зрозуміти одним способом, але, здавалося б, неможливим у зворотному напрямку. Іншим цікавим аспектом цього рівняння є те, що просто зрозуміти одне з простих чисел, якщо у вас вже є інший, а також продукт. Якщо вам сказали, що результат 701111 є результатом 907, помноженого на інше просте число, ви можете зрозуміти його іншим простим за допомогою наступного рівняння:

701,111 7 907 = 773

Оскільки взаємозв'язок між цими числами простий для обчислення в одному напрямку, але неймовірно важко в зворотному напрямку, рівняння відоме як функція дверей пастки. Майте на увазі, що хоча вищевказаний приклад важко зрозуміти людям, комп'ютери можуть виконувати операцію тривіально. RSA можна використовувати не тільки для шифрування даних. Його властивості також роблять його корисною системою для підтвердження того, що повідомлення було надіслано суб'єктом, який стверджує, що він надіслав його, а також довів, що повідомлення не було змінено або було. Коли хтось хоче довести автентичність свого повідомлення, вони можуть обчислити хеш (функцію, яка приймає дані довільного розміру і перетворює її у значення фіксованої довжини) відкритого тексту, а потім підписує його своїм закритим ключем. Вони підписують хеш, застосовуючи ту ж формулу, що й у розшифровці ($m = c\ d\ mod\ n$). Після підписання повідомлення вони надсилають цифровий підпис одержувачу поруч із повідомленням. Якщо одержувач отримує повідомлення з цифровим підписом, вони можуть використовувати підпис, щоб перевірити, чи автентичне повідомлення було підписано приватним ключем особи, яка стверджує, що вона надіслана. Вони також можуть переконатися, що повідомлення було змінено зловмисниками після його надсилання. Щоб перевірити цифровий підпис, одержувач спочатку використовує ту ж саму функцію, щоб знайти значення хешу для отриманого повідомлення. Одержувач потім застосовує відкритий ключ відправника до цифрового підпису, використовуючи формулу шифрування ($c = m\ e\ mod\ n$), щоб надати їм хеш цифрового підпису. Порівнюючи хеш повідомлення, яке було отримано поряд з хешем із зашифрованого цифрового підпису, одержувач може визначити, чи є повідомлення автентичним. Якщо ці два значення однакові, повідомлення не було змінено, оскільки було підписано оригінальним відправником. Якщо повідомлення було змінено навіть одним символом, значення хешу було б зовсім іншим. Як і більшість криптосистем, безпека RSA залежить від того, як вона реалізується і використовується. Важливим фактором є розмір ключа. Чим

більше число бітів в ключі (по суті, скільки часу), тим важче пробитися для атаки, такі як грубий примус і факторинг. Оскільки алгоритми асиметричного ключа, такі як RSA, можуть бути розбиті факторизацією цілих чисел, тоді як алгоритми симетричного ключа, такі як AES, не можуть. Ключі RSA повинні бути набагато довші, щоб досягти того ж рівня безпеки. В даний час найбільший розмір ключа, який було враховано, становить 768 біт. Це було зроблено командою вчених протягом двох років, використовуючи сотні машин. Оскільки факторинг був завершений до кінця 2009 року, а обчислювальна потужність значно зросла з того часу, можна припустити, що спроба подібної інтенсивності тепер може вплинути на набагато більший ключ RSA.

Незважаючи на це, час і ресурси, необхідні для такого роду нападу, ставлять його поза досяжність більшості хакерів і в область національних держав. Найкраща довжина ключа для використання залежить від вашої індивідуальної моделі загрози. Національний інститут стандартів і технологій рекомендує мінімальний розмір ключа 2048-біт, але 4096-бітові ключі також використовуються в деяких ситуаціях, коли рівень загрози вище.

Факторинг - це лише один із способів розриву RSA. Ряд інших атак мають потенціал для розриву шифрування з меншою кількістю ресурсів, але вони залежать від реалізації та інших факторів, не обов'язково самої RSA.

Деякі реалізації RSA використовують генератори слабких випадкових чисел, щоб придумати прості числа. Якщо ці цифри не є достатньо випадковими, це набагато полегшує їх зловмисникам і порушують шифрування. Цю проблему можна уникнути за допомогою криптографічно безпечного генератора псевдовипадкових чисел.

Ключі RSA повинні потрапляти в певні параметри, щоб вони були безпечними. Якщо прості числа p і q знаходяться надто близько один до одного, ключ легко можна виявити. Аналогічно, число d, яке становить частину закритого ключа, не може бути занадто малим. Низьке значення дозволяє легко вирішити. Важливо, щоб ці цифри мали достатню довжину, щоб зберегти ключ у безпеці.

Це тип атаки, який не порушує безпосередньо RSA, а замість цього використовує інформацію з його реалізації, щоб дати нападникам підказки про процес шифрування. Ці атаки можуть включати в себе такі речі, як аналіз кількості енергії, що використовується, або аналіз прогнозування гілок, який використовує вимірювання часу виконання, щоб виявити приватний ключ.

Інший тип нападу бічного каналу відомий як атака за часом. Якщо зловмисник має можливість вимірювати час дешифрування на комп'ютері своєї цілі для ряду різних зашифрованих повідомлень, ця інформація може дозволити зловмиснику визначити приватний ключ цілі.

Більшість реалізацій RSA уникають цієї атаки, додаючи одноразове значення під час процесу шифрування, що усуває цю кореляцію. Цей процес називається криптографічним сліпучим.

Доброю новиною є те, що RSA вважається безпечним у використанні, незважаючи на можливі атаки. Застереження полягає в тому, що воно має бути правильно реалізовано і використовувати ключ, який відповідає правильним параметрам. Як ми вже обговорювали, реалізації, які не використовують прокладки, використовують неадекватні розміри чи інші уразливості, не можна вважати безпечними.

Якщо є потреба використовувати шифрування RSA, потрібно переконатися, що використовується ключ не менше 1024 біт. Завдання, з більш високими моделями загрози повинні дотримуватися ключів 2048 або 4096 біт, якщо вони хочуть використовувати RSA з упевненістю. Поки ви усвідомлюєте недоліки, які RSA має і правильно використовуєте, ви повинні бути впевнені у використанні RSA для спільного використання ключів та інших подібних завдань, які вимагають шифрування відкритим ключем.

2.3 Основні поняття симетричних та асиметричних алгоритмів

На сьогоднішній день існує багато алгоритмів шифрування , які допомагають нам залишити нашу інформацію конфіденційною. У криптографії алгоритми шифрування генерують ключі, які є не більш ніж серією бітів, що

використовуються для шифрування та дешифрування інформації. Спосіб використання цих ключів залежить від обраного методу шифрування.

Алгоритми шифрування використовуються для зміни конфіденційної інформації до такого виду, щоб вона не була зрозуміла для прочитання стороннім особам. Перші шифри використовувалися ще за часів Стародавнього Риму, Стародавнього Єгипту і Стародавньої Греції. Одним з відомих шифрів є шифр Цезаря. Даний алгоритм працював наступним чином: кожна буква має свій порядковий номер у алфавіті, який зміщувався на значення вліво. Сьогодні подібний алгоритм не забезпечує той захист, який давав за часів його використання. Сьогодні розроблено велику кількість алгоритмів шифрування, в тому числі стандартних, які забезпечують надійний захист конфіденційної інформації. Розділяють алгоритми шифрування на симетричні (до них відносяться AES, CAST, ГОСТ, DES, Blowfish) і асиметричні (RSA, El-Gamal).

У чому ж саме полягає різниця між симетричним та асиметрим шифрування? Суть симетричних алгоритмів шифрування полягає в перетворенні відкритого тексту в секретний текст шляхом застосування підстановок і повторів. Обидві сторони використовують один і той же ключ, що забезпечує повний захист секретної інформації. Це є і недоліком, і перевагою, оскільки відповідальність за безпеку таємниці лежить на обох сторонах. Тому виникла потреба в розробці методів шифрування, де секретність інформації залежить тільки від однієї сторони. Методи, які використовують інший ключ для збереження секретності повідомлень від розсекречення. Це асиметричні методи шифрування, які характеризуються наступним:

- Ключі шифрування та дешифрування повинні бути різними;
- Зашифроване повідомлення є одним з ключів, що є парою з ключем для розшифрування. Повідомлення, зашифроване за допомогою даного ключа, не може бути розшифровано з ним;
- Знання однієї з ключів не дозволить вам вгадати іншу;

Прикладами сучасних симетричних шифрів є алгоритми, які використовуються для конфіденційності інформації:

- Алгоритм DES, де текст ділиться на восьмибайтові блоки даних, які є аргументами для операцій зсувів і замін в залежності від використовуваного ключа шифрування,
- Алгоритм AES - він також працює на блоках даних, але 16-байт, і шифрування захищено ключами, довжина яких може становити 128, 192 або 256 біт.

Існує декілька прикладів асиметричного шифрування. Основою для комунікації за допомогою цього методу є генерування пари ключів. На відміну від симетричних алгоритмів шифрування, де будь-який номер може бути ключем шифрування, створення ключів шифрування RSA є досить складною операцією. В даний час використовуються ключі шифрування з мінімумом 1024 біта. Ключ, який було опубліковано, називається відкритим ключем. Він може бути використаний для шифрування або дешифрування інформації, отриманої від людини, яка її створила. Приватний (секретний) ключ використовується для шифрування повідомлень, підготовлених власником цього ключа. Надійність шифрування забезпечується тим, що третій особі (що намагаєтеся зламати шифр) дуже важко відрізнити закритий ключ з відкритого. Обидва ключі обчислюються з однієї пари простих чисел (p і q). Тобто ключі пов'язані між собою. Але встановити цей зв'язок дуже складно. [13]

Найбільшим недоліком симетричних алгоритмів шифрування є необхідність використання одного і того ж ключа для шифрування і дешифрування повідомлень. Тому, перш ніж зробити секретну комунікацію, відправник і одержувач повинні обміняти ключ. Ключ повинен передаватися через захищений канал. У більшості випадків люди, які спілкуються один з одним, не мають такого варіанту. Канал передачі, який вони використовують, піддається атакам, і агресор має можливість перехопити секретний ключ. Переваги алгоритмів асиметричного шифрування включають більш високий рівень безпеки і зручності використання. Ці методи не вимагають передачі секретних ключів або розкриття їх будь-кому. Додатковою перевагою асиметричних шифрів є можливість їх використання для створення цифрових підписів. Недоліком асиметричних шифрів є великий обсяг роботи, необхідний

для шифрування і дешифрування повідомлень, що призводить до низької продуктивності цих методів. Як правило, симетричні алгоритми дозволяють захищати явне і читати секретні повідомлення набагато швидше. Різниця в продуктивності може бути особливо важливо при обробці великих обсягів даних. У цьому випадку асиметричні методи шифрування не працюють [25].

У деяких ситуаціях симетричних алгоритмів цілком достатньо, і вам не потрібно використовувати складну асиметричну криптографію. Це той випадок, коли обставини дозволяють надійно встановити ключ через спілкування з особами (наприклад, на особистій зустрічі). Симетричні методи також корисні в середовищах, в яких центральний пристрій керує всіма ключами (наприклад, у закритих банківських системах). Оскільки блок управління все ще має всі ключі, немає необхідності визначати приватні та відкриті ключі. Асиметрична криптографія не потрібна, якщо потрібно захистити лише одного користувача. Наприклад, симетричний шифр достатній для шифрування приватних файлів. Асиметричні алгоритми зазвичай краще підходять для середовищ, що використовуються кількома або більше користувачами. Асиметричні види шифрування залежать від двох ключів. Спочатку користувач шифрує повідомлення за допомогою публічного ключа та отримує на виході розшифровку за допомогою приватного ключа. У кінці ми отримуємо повідомлення , яке було зашифровано та розшифровано за допогою пари ключів. (див. рис. 2.2.)

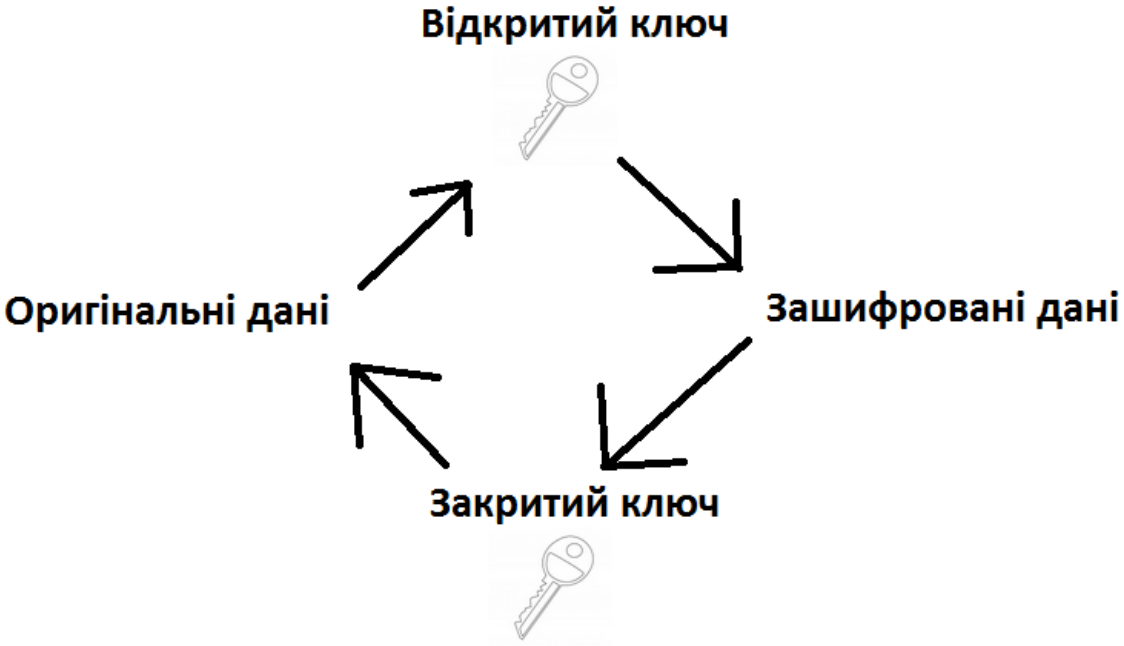

Рис. 2.2. Схема асиметричного шифрування

Дуже важливо, щоб приватний ключ був захищений від перехоплення несанкціонованими особами. Відкритий ключ, як випливає з назви, можна розділити з усіма. На відміну від симетричного шифрування, характерною особливістю цього методу є відсутність загального секрету. Недоліком такого рішення є більша обчислювальна складність процедур шифрування та дешифрування даних щодо симетричного шифрування. Тому там, де важлива швидкість роботи (зашифроване з'єднання між комп'ютерами), використовується симетричне шифрування. Закритий ключ і відкритий ключ можна поміняти місцями з ролями. Він має перевагу під час цифрового підпису. Аутентифікатор обчислює абревіатуру повідомлення, яку він підготував, використовуючи вибраний ним алгоритм (зазвичай MD-5). Потім він шифрує цей ярлик своїм закритим ключем і додає його до вихідного повідомлення як підпис. Особа, яка має відкритий ключ, може перевірити автентичність підпису шляхом розшифровки ярлика за допомогою відкритого ключа відправника. В результаті він отримує ярлик оригінального повідомлення. Порівняння скорочень вихідного повідомлення, отриманого і рішення щодо достовірності повідомлення, - ідентичність порівнюваних скорочень вказує на успішний результат верифікації. Якщо порівняні скорочення відрізняються, це означає, що

повідомлення було змінено або абревіатура була створена особою, відмінною від відправника. Протилежна ситуація означає, що повідомлення не було змінене кимось іншим, і водночас було надіслано відправником!

Симетричне та асиметричне шифрування відіграють важливу роль у забезпеченні конфіденційної інформації та комунікації в сучасному цифровому залежному світі. Хоча обидва підходи можуть бути корисними, кожен має свої переваги та недоліки. Вони використовуються в різних додатках і різними способами. Оскільки криптографія - як галузь криптології - продовжує розвиватися для захисту від нових і більш складних загроз, і симетричні, і асиметричні криптографічні алгоритми, ймовірно, продовжуватимуть відігравати важливу роль у забезпеченні безпеки комп'ютерних систем і додатків.

2.4 Методи шифрування

Шифрування може захистити інформацію про споживачів, електронні листи та інші конфіденційні дані, а також захищені мережні підключення. Сьогодні існує безліч варіантів вибору, і знайти необхідне, яке є безпечним і відповідає вашим потребам. Ось чотири методи шифрування і те, що ви повинні знати про кожну з них.

2.4.1 Метод шифрування Blowfish

Blowfish - це ще один алгоритм, призначений для заміни DES. Цей симетричний шифр, що розбиває повідомлення на блоки по 64 біта і шифрує їх окремо. Blowfish відома як своєю величезною швидкістю, так і загальною ефективністю, оскільки багато хто стверджує, що він ніколи не був переможений. Тим часом, постачальники повністю скористалися його вільною доступністю у суспільному надбанні.

Blowfish можна знайти в категоріях програмного забезпечення, від платформ електронної комерції для забезпечення платежів до інструментів керування паролями, де вони використовуються для захисту паролів. Це, безумовно, один з найбільш гнучких методів шифрування. Він являється методом , який являється дуже корисним.

2.4.2 Метод шифрування 3DES

Потрійний стандарт шифрування даних, або 3DES, є поточним стандартом, і це блоковий шифр. Він подібний до старого методу шифрування - Data Encryption Standard, який використовує 56-розрядні ключі. Однак, 3DES - це шифрування симетричного ключа, що використовує три окремих 56-розрядних ключа. Він шифрує дані три рази, тобто ваш 56-бітний ключ стає 168-бітним ключем.

На жаль, оскільки він шифрує дані три рази, цей метод набагато повільніше, ніж інші. Крім того, оскільки 3DES використовує більш короткі довжини блоків, легше розшифрувати і витоку даних. Тим не менш, багато фінансових установ і підприємств у багатьох інших галузях використовують цей метод шифрування для збереження інформації. У міру появи надійніших методів шифрування, цей процес повільно припиняється.

2.4.3 Метод шифрування Twofish

Twofish - симетричний блоковий шифр, заснований на більш ранньому блоковому шифрі - Blowfish. Twofish має розмір блоку 128 біт до 256 біт, і він добре працює на менших процесорах і апаратних засобах. Подібно до AES, він реалізує раунди шифрування, щоб перетворити відкритий текст на шифр. Однак кількість раундів не змінюється, як у випадку з AES; незалежно від розміру ключа, завжди є 16 раундів.

Крім того, цей спосіб забезпечує велику гнучкість. Ви можете вибрати, щоб налаштування ключа було повільним, але процес шифрування буде швидким або навпаки. Крім того, ця форма шифрування є незапатентованою та ліцензійною, тому ви можете використовувати її без обмежень.

2.4.4 Метод шифрування AES

AES - розширений стандарт шифрування AES - це симетричний алгоритм шифрування і один з найбільш безпечних. Уряд Сполучених Штатів використовує його для захисту конфіденційної інформації, і багато програмних і апаратних продуктів також використовують її. Цей метод використовує блоковий шифр, який за один раз шифрує дані одного блоку фіксованого

розміру, на відміну від інших типів шифрування, таких як шифри потоку, які шифрують дані по біту.

AES складається з AES-128, AES-192 і AES-256. Вибраний ключовий біт шифрує і розшифровує блоки в 128 бітах, 192 біти і так далі. Існують різні раунди для кожного біта ключа. Круглий процес - це перетворення відкритого тексту на шифр. Для 128-бітових є 10 раундів; 192-бітний має 12 раундів; і 256-бітний має 14 раундів.

Оскільки AES є симетричним ключем шифрування, ви повинні поділитися ключем з іншими особами для них, щоб отримати доступ до зашифрованих даних. Більше того, якщо ви не маєте безпечного способу спільного використання цього ключа та несанкціонованого доступу до нього, вони можуть розшифрувати всі зашифровані за допомогою спеціального цього ключа.

2.5 Способи шифрування веб-додатків

Криптографічна зона існує між двома точками, де симетричний ключ або асиметричні відкриті ключі спільно використовуються для шифрування конфіденційної інформації. Як тільки ключ або ключі були обмінені, дані, а в деяких випадках і інші ключі, шифруються в цій зоні. Більшість контролів безпеки, що використовуються веб-додатками, залежать від криптографії, а тому також залежать від секретних ключів. Система керування ключами (KMS) повинна бути розроблена для ефективної обробки секретних ключів під час передачі конфіденційних даних. Криптографічні зони можна розділити на 3 частини:

Зона 1: зовнішній користувач і веб-додаток

Коли користувач підключається до захищених (HTTPS) веб-сайтів, таких як програма "Інтернет-банкінг", браузер повинен встановити безпечний сеанс

TLS. [15]

Веб-сторінку слід переглядати в браузері в зашифрованому вигляді. Запит і відповідь між браузером і сервером шифруються за допомогою шифрування відкритим ключем.

Веб-програми використовують криптографію відкритого ключа для створення спільного ключа сеансу. Потім він здійснює зв'язок через криптографію симетричного ключа за допомогою цього спільного ключа сеансу. Криптографія з відкритим ключем залишається найпопулярнішим онлайновим протоколом (через криптографію приватного ключа), оскільки користувачам ніколи не потрібно передавати чи розкривати свої приватні ключі будь-кому, що зменшує шанси кіберзлочинців виявити секретний ключ
людини під час передачі.

Зона 2: веб-сервер і сервер додатків

Багато веб-додатків надсилають ці дані в прозорий текст, але якщо хакер може скомпрометувати веб-сервер, він матиме подібні права для перегляду облікових даних.

Кілька веб-додатків не використовують шифрування під час зв'язку з сервером веб-сервера та сервером додатків. Але завжди рекомендується гарантувати, що дані шифруються наскрізь, від веб-браузера до сервера додатків або сервера баз даних. Тому настійно рекомендується надсилати дані через новий зашифрований сеанс шляхом встановлення нового сеансу SSL або передачі даних через тунель IPSec.

Зона 3: сервер додатків і сервер бази даних

Після отримання конфіденційних даних сервер веб-додатків повинен надіслати його на сервер бази даних для перевірки. Сервер веб-додатків і сервер баз даних зазвичай знаходяться в одній надійній мережі, але рекомендується використовувати SSL для шифрування зв'язку.

В ідеалі, ці елементи даних шифруються симетричним ключем, попередньо узгодженим з системою мейнфреймів. Шифрування - це забезпечення конфіденційності. Дані може читати лише уповноважений одержувач. Методи шифрування також покладаються на силу і тип використовуваних алгоритмів.

Шифрування даних у спокої повинно включати сильні методи шифрування, такі як AES і RSA. Зі збільшенням уразливостей безпеки SSL, таких

як POODLE, Heartbleed або FREAK, рекомендується використовувати TLS 1.2 або вище [28].

Симетричне шифрування є достатньо безпечним для захисту комунікацій, але воно страждає від фундаментального недоліку: ключ шифрування має бути розділений обома сторонами. Таким чином, нам потрібен спосіб обміну ключами без його перехоплення, перевіряючи особистість особи, з якою ми спілкуємося.

Асиметричне шифрування забезпечує вирішення цієї проблеми. Асиметричні протоколи генерують два ключі замість одного, на основі двох випадково обраних простих чисел. Чудова властивість цієї процедури полягає в тому, що повідомлення, зашифроване одним ключем, може бути розшифровано лише за допомогою іншого ключа. Якщо необхідно відправити засіб відправлення повідомлення власником ключів, то відправитель повинен отримати відкритий ключ. Отправитель розширює своє повідомлення відкритим ключем і передає його отримує (власнику ключів) по відкритим каналам. При цьому расшифровать сообщение не може никто, кроме власниця закритого ключа. (див. рис. 2.3.)

Рис 2.3. Схема асиметричного шифрування

Повідомлення, зашифроване за допомогою ключа 1, може бути розшифровано тільки за допомогою ключа 2. Зворотне також вірно: повідомлення, зашифроване за допомогою ключа 2, може бути розшифровано тільки за допомогою ключа 1.

На практиці, перший ключ зберігається в секреті своїм власником: його називають приватним ключем. Другий ключ, відкритий ключ, передається всім одержувачам, які його запитують.

Цей механізм забезпечує простий спосіб виконання двох різних завдань: шифрування повідомлень і перевірка ідентичності комунікаційного партнера.

Якщо перший користувач хоче надіслати другому користувачу зашифроване повідомлення, він отримує свій відкритий ключ і використовує його для шифрування повідомлення. Потім перший користувач може розшифрувати повідомлення, використовуючи свій приватний ключ: вона є єдиною, яка може це зробити, оскільки вона є єдиною, яка знає приватний ключ [29].

Тепер, якщо перший користувач посилає повідомлення другому, і він хоче бути впевненим, що це, безумовно, перший, яка відіслала його, процедура трохи складніше.

Виконується наступна послідовність операцій:

- Перший користувач обчислює хеш її повідомлення за допомогою хеш-функції, як описано вище;
- Перший користувач шифрує хеш, використовуючи свій закритий ключ;
- Перший користувач посилає другого повідомлення з зашифрованим хешем;
- Другий користувач отримує це повідомлення і обчислює його хеш;
- Другий користувач розшифровує зашифрований хеш, надісланий першим, використовуючи свій відкритий ключ;
- Нарешті, він порівнює обидві хеши: якщо вони однакові, то, мабуть, перший користувач посилає повідомлення.

Звичайно, цей протокол здійснюється автоматично, і ці розрахунки виконуються програмами, такими як поштові клієнти, такі як Thunderbird.

Асиметричне шифрування є відносно надійним, оскільки в даний час неможливо швидко факторизувати добуток двох простих чисел, якщо вони обрані достатньо великими (є й інші алгоритми керування асиметричними ключами на основі еліптичних кривих, а не простих чисел; ці алгоритми не вимагають, щоб ключі були настільки великими). Для шифрування даних

використовується хеш. (див. рис. 2.4.)

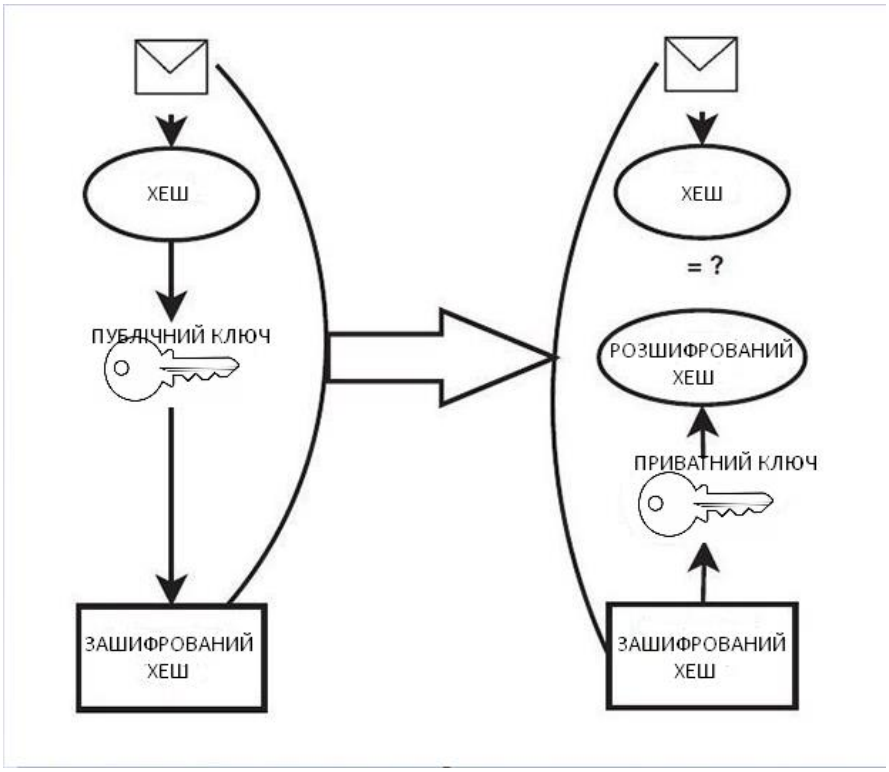

Рис. 2.4. Шифрування з використанням хеш

В даний час, прості числа, що використовуються для створення ключів, повинні мати розміри не менше 2048 біт, щоб гарантувати, що вони є надійними. Навіть сьогодні ANSSI рекомендує використовувати ключі з 3096 бітами, особливо якщо вони призначені для використання до 2030 року.

2.6 Вибір технологій розробки

Вибір відповідного технологічного стеку є особливо складним завданням для малих підприємств та стартапів, оскільки вони, як правило, мають обмежені бюджети і, таким чином, потребують технологічного стеку, який забезпечує найбільший удар для долара, щоб отримати свої проекти на місцях.

Правильний технологічний стек значною мірою є ключем до успіху вашого проекту, в той час як неправильний вибір технологій розробки веб-додатків може бути причиною невдачі [16].

Перш ніж перейти до критеріїв вибору сучасних стекових веб-технологій, слід чітко зрозуміти, що включає процес розробки веб-додатків.

Не заглиблюючись в деталі, існують дві сторони веб-розробки: клієнтська і серверна. Клієнтська сторона також називається передньою частиною.

Серверне програмування включає в себе додаток (і мова програмування, що керує нею), базу даних і сам сервер [17].

Веб-розробка на стороні клієнта (тобто інтерфейс) включає в себе все, що користувачі бачать на своїх екранах. Нижче наведено основні компоненти технології стеку:

Мова гіпертекстової розмітки (HTML) і каскадні таблиці стилів (CSS). HTML повідомляє веб-переглядачу, як відображати вміст веб-сторінок, а CSS - вміст цього вмісту. Bootstrap є корисною основою для керування HTML і CSS.

JavaScript (JS). JS робить веб-сторінки інтерактивними. Існує багато бібліотек JavaScript (наприклад, jQuery, React.js і Zepto.js) і фреймворки (такі як Angular, Vue, Backbone і Ember) для швидшої та легшої веб-розробки.

Серверна сторона не є видимою для користувачів, але вона запускає клієнтську сторону, так само, як електростанція генерує електрику для вашого будинку [18].

Завдання полягає в основному у виборі серверних технологій для розробки веб-додатків.

Що стосується серверних мов програмування, вони використовуються для створення логіки веб-сайтів і додатків. Рамки для мов програмування пропонують безліч інструментів для більш простого і швидкого кодування. Давайте згадаємо деякі популярні мови програмування та їх основні фреймворки (у дужках):

- Ruby
- Python
- PHP
- Java

Node.js, час виконання JavaScript, також використовується для програмування на сервері. Веб-додаток потребує сервера для обробки запитів від комп'ютерів клієнтів.

Щоб розробити веб-додаток, потрібно вибрати сервер, мову програмування, фреймворк і інструменти інтерфейсу, які ви збираєтеся використовувати.

Клієнтська і серверна - це терміни веб-розробки, які описують, де виконується код програми. Веб-розробники також будуть посилатися на цю відмінність як інтерфейс у порівнянні з сервером, хоча клієнтська / серверна і frontend / backend не зовсім однакові. У безсерверній архітектурі постачальник без серверного сервера розміщує і призначає ресурси для всіх процесів на стороні сервера, а процеси розширюються по мірі збільшення використання програми [19].

Значна частина Інтернету базується на моделі клієнт-сервер. У цій моделі користувальницькі пристрої спілкуються через мережу з централізовано розташованими серверами, щоб отримати необхідні дані, а не спілкуватися один з одним. Пристрої кінцевого користувача, такі як ноутбуки, смартфони та настільні комп'ютери, вважаються "клієнтами" серверів, як якщо б вони були клієнтами, які отримували послуги від компанії. Клієнтські пристрої надсилають запити до серверів для веб-сторінок або програм, а сервери надають відповіді.

Модель клієнт-сервер використовується тому, що сервери зазвичай є більш потужними і надійнішими, ніж користувацькі пристрої. Вони також постійно підтримуються і зберігаються в контрольованих умовах, щоб переконатися, що вони завжди доступні і доступні; Незважаючи на те, що окремі сервери можуть знизитися, зазвичай їх підтримують інші сервери. Тим часом користувачі можуть вмикати та вимикати свої пристрої, втрачати або ламати свої пристрої, і це не повинно впливати на інтернет-сервіс для інших користувачів [20].

Сервери можуть обслуговувати кілька клієнтських пристроїв одночасно, і кожен клієнтський пристрій надсилає запити до декількох серверів під час доступу та перегляду Інтернету.

Кілька клієнтів і серверів взаємодіють:

Кожен клієнт буде спілкуватися з декількома серверами, і навпаки. (див. рис. 2.5.)

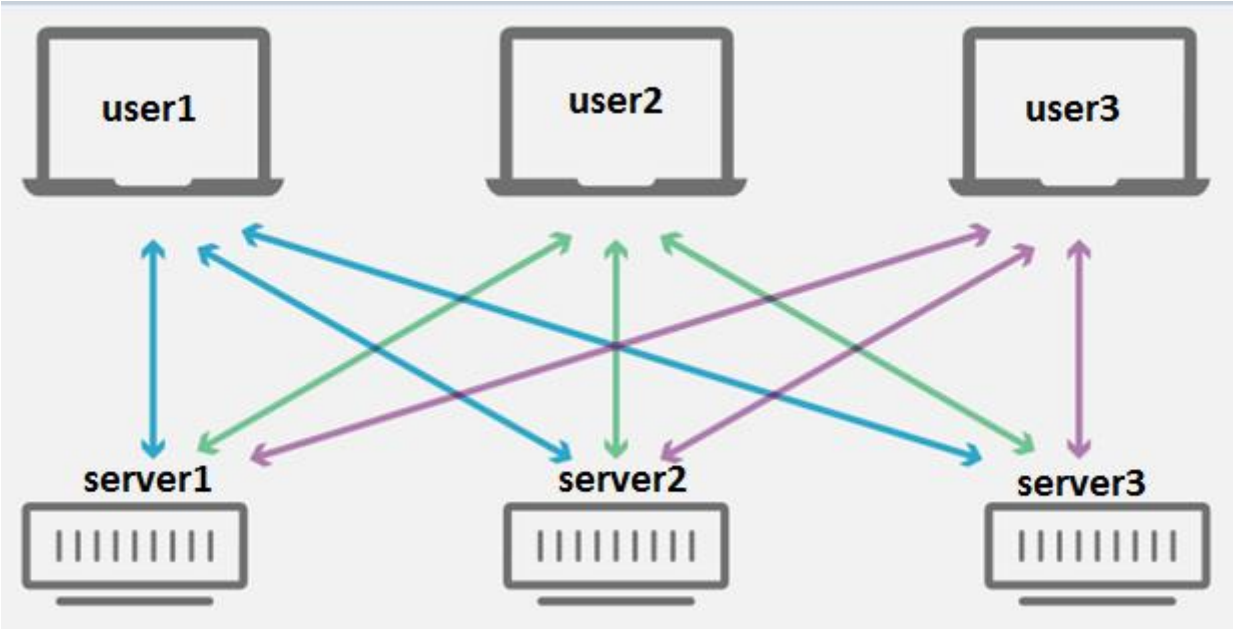

Рис. 2.5. Взаємодія клієнтів з серверами

Припустимо, що користувач переглядає Інтернет і вводить 'google.com' у панель перегляду. Це призводить до запиту на DNS-сервери для IP-адреси google.com, і DNS-сервери реагують на цей запит, обслуговуючи IP-адресу браузеру. Далі браузер користувача робить запит до серверів Google (використовуючи IP-адресу) для вмісту, який відображається на сторінці, наприклад, мініатюри фільмів, логотипу Google і панелі пошуку. Сервери Google доставляють це в браузер, і браузер завантажує сторінку на клієнтському пристрої [21].

У веб-розробці "клієнтська сторона" відноситься до всього, що відображається або відбувається на клієнтському пристрої (пристрої кінцевого користувача). Це включає в себе те, що бачить користувач, наприклад, текст, зображення та іншу частину інтерфейсу, а також будь-які дії, які виконує програма в браузері користувача [22].

Мови розмітки, такі як HTML і CSS, інтерпретуються браузером на стороні клієнта. Крім того, багато сучасних розробників включають клієнтські процеси у своїй архітектурі додатків і віддаляються від роботи на серверній стороні; бізнес-логіка для динамічних веб-сторінок *, наприклад, зазвичай запускає клієнтську сторону в сучасному веб-застосунку. Процеси на стороні клієнта майже завжди написані на JavaScript.

У прикладі google.com вище, HTML, CSS і JavaScript, які диктують, як головна сторінка Google з'являється користувачеві, інтерпретуються браузером на стороні клієнта. Сторінка також може відповідати на "події": наприклад, якщо миша користувача зависає над одним із мініатюр зображень, зображення розширюється, а сусідні ескізи злегка переміщуються в одну сторону, щоб створити місце для великого зображення. Це приклад процесу на стороні клієнта; код самої веб-сторінки відповідає миші користувача і ініціює цю дію без спілкування з сервером.

Клієнтська сторона також відома як інтерфейс, хоча ці два терміни не означають точно одне й те саме. Клієнтська сторона відноситься виключно до місця розташування процесів, а frontend - до типів процесів, які виконуються на стороні клієнта.

Динамічна веб-сторінка - це веб-сторінка, яка не відображає однаковий вміст для всіх користувачів і змінюється на основі вводу користувача. Як і на стороні клієнта, "серверна частина" означає все, що відбувається на сервері, а не на клієнті. У минулому майже вся бізнес-логіка працювала на стороні сервера, і це включало надання динамічних веб-сторінок, взаємодія з базами даних, аутентифікацію ідентичності та push-повідомлення.

Проблема з розміщенням всіх цих процесів на серверній стороні полягає в тому, що кожен запит, що стосується одного з них, повинен проходити весь шлях від клієнта до сервера, кожен раз. Це викликає значну затримку. З цієї причини сучасні програми запускають більше коду на стороні клієнта; один випадок використання відображає динамічні веб-сторінки в режимі реального часу, запускаючи скрипти в браузері, які вносять зміни до вмісту, який бачить користувач [23].

Як і у випадку з frontend і backend також є терміном для процесів, які відбуваються на сервері.

Сценарії на стороні клієнта - це просто запуск сценаріїв, таких як JavaScript, на клієнтському пристрої, як правило, в браузері. Всі види скриптів можуть запускатися на стороні клієнта, якщо вони написані на JavaScript, тому що

JavaScript універсально підтримується. Інші мови сценаріїв можна використовувати, лише якщо їх підтримує браузер користувача.

Серверні скрипти виконуються на сервері замість клієнта, часто з метою доставки динамічного вмісту на веб-сторінки у відповідь на дії користувача. Сценарії на сервері не потрібно писати на JavaScript, оскільки сервер може підтримувати різні мови.

У безсерверних обчисленнях всі процеси на сервері або на сервері все ще працюють на серверах, а не на клієнтських пристроях, але вони не розгортаються на якомусь конкретному сервері або наборі серверів. Процеси бекенда розбиваються на функції, які виконуються на вимогу, і розширюються автоматично. Розробники все ще можуть створювати всі функціональні можливості, які зазвичай виконуються на стороні сервера в архітектурі без сервера [24].

Одним із перших кроків є створення інтерфейсу для користувача на сервері, розглянемо це питання докладніше.

Створюється зручний вид інтерфейсу, який допомагає користувачу користуватися веб-додатком якомога зручніше.

Веб-додаток може використовувати операції CRUD (створити, прочитати, оновити, видалити) без складної логіки endpoint (кінцевих точок).

Кінцева точка - це URL, адреса місцезнаходження ресурсу в мережі Інтернет. Ця електронна адреса використовує клієнт, щоб отримати доступ до ресурсу. Відсутній поділ на ролі, відсутня система аутентифікації, не використовується бібліотеки, що розширюють мова (babel, typescript).

JavaScript спочатку замислювався як проста мова, яка могла б використовуватися не тільки програмістами. Задумка вдалася: на даний момент у нас є нескладна мова для реалізації як простих, так і складних продуктів.

JavaScript забезпечує динамічність сторінки. Анімація, інтерактивні кнопки і графічні елементи.

JavaScript - це клієнтська мова. Клієнтом в цьому випадку є браузер, який інтерпретує код програми та виконує його. Інтерпретатор коду вбудований в усі

популярні браузери, тобто в плані підтримки ніяких обмежень не існує. з іншого боку, JavaScript можна використовуватися і на стороні сервера завдяки платформі Node.js. (див. рис. 2.6.)

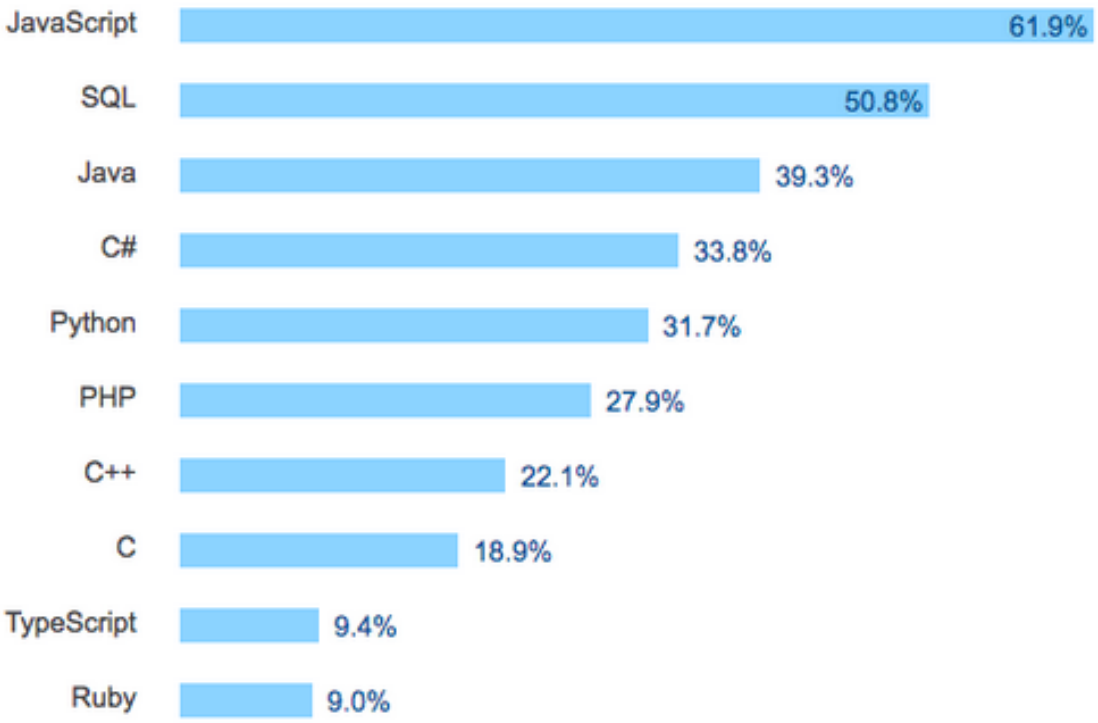

Рис. 2.6. Найбільш використовувані мови програмування

Angular, React, jQuery, Vue, Lodash - про них багато говорять і сперечаються, але, головне, їх успішно використовують, і вони виконують своє призначення.

По суті, це середовище програмування, в якій створені умови для застосування JavaScript як серверної мови. Тобто розробник може використовувати JavaScript і для сторони сервера, і для сторони клієнта, і йому не потрібно перемикатися на специфіку іншої мови програмування, і при цьому зберігається формат даних.

Кожен програміст відповість по-своєму на питання про те, яка мова краще використовувати для серверної частини. У кожному разі зіграють роль накопичений досвід роботи, звичка, особливості продукту, що розробляється. Згідно зі статистикою використання Node.js для веб-сайтів, частка застосування платформи стабільно збільшується: (див. рис. 2.7.)

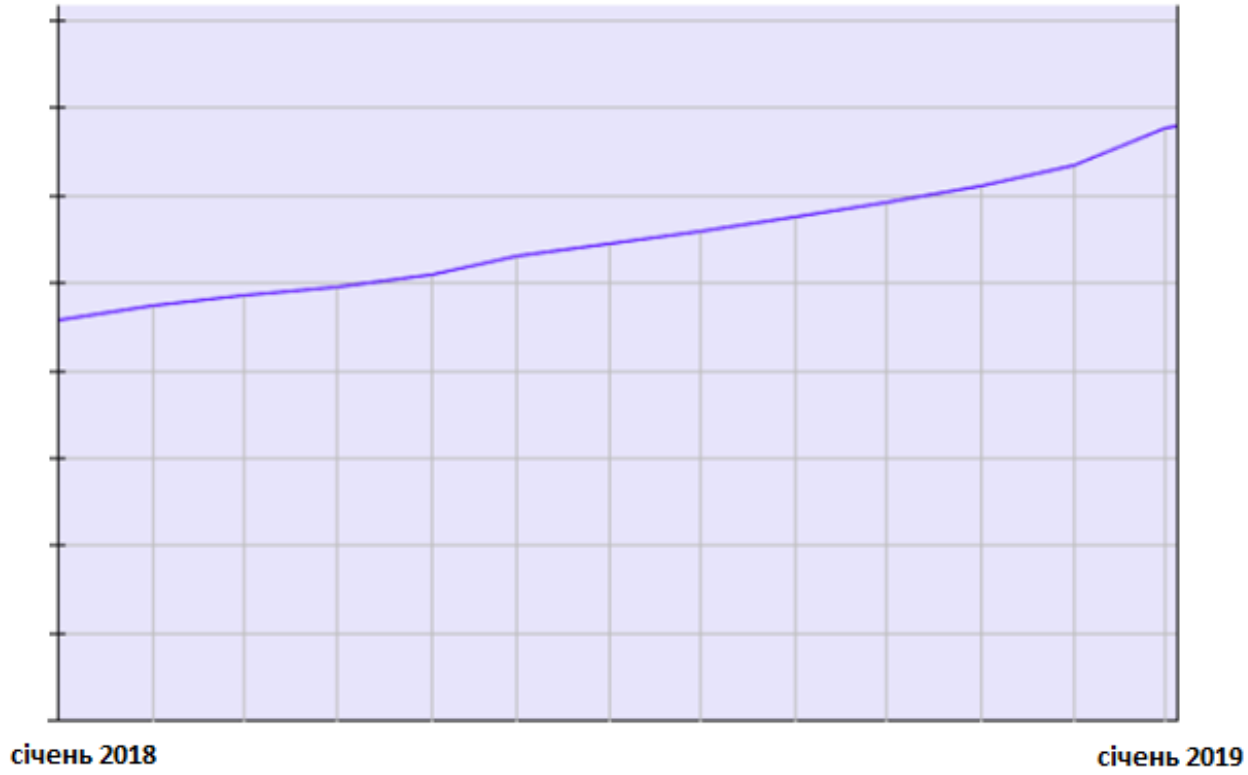

Рис 2.7. Застосування NodeJS

За допомогою Node.js, можна створювати практично будь-які веб-додатки, крім, хіба, складних обчислень. Хоча складні обчислення завжди можна зробити на іншій мові, а потім надати ці дані для використання і маніпулювання ними вже на JavaScript [30].

Специфічною областю застосування платформи Node.js є мікросервісна архітектура. Тобто Node.js оптимально використовувати для додатків, які розбиті на невеликі модулі. При цьому кожен модуль не залежить один від одного, але всі разом вони складають єдину структуру.

Додаткова перевага полягає в тому, що навколо Node.js утворилося розвинене співтовариство. Завдяки цьому доступна хороша документація, а також існує велика кількість готових прикладів для використання, фреймворків і бібліотек, які дозволяють полегшити і прискорити розробку. NodeJS використовується для підняття веб-серверу, який здатний взаємодіяти між сервером там юзером. Він має методи, які дають змогу ділити код на модулі та створювати швидкісні додатки.

2.7 Висновки до другого розділу

RSA це не тільки введення в ступінь по модулю більшої кількості. Це ще одне доповнення, яке дає змогу реалізовувати додатковий захист вашої інформації від несанкціонованого доступу.

Відкритий ключ не потрібно ховати. Неважливо кому відомий даний ключ, оскільки він призначений тільки для шифрування даних. Цей метод придатний для широкого застосування. Якщо привласнити кожному користувачеві в Інтернет свою пару ключів і опублікувати відкриті ключі як номери в телефонній книзі, то практично всі зможуть обмінюватися один з одним шифрованими повідомленнями. Це схоже на коробку з двома дверцятами з різних сторін. Кожна така дверцята має свій замок. У коробку кладуть документ, замикають, відмикають з іншого боку ключем одержувача. При цьому використовується теорія простих чисел. Такий алгоритм криптографічного захисту отримав назву RSA.

Всі асиметричні криптосистеми є об'єктом атак шляхом прямого перебору ключів, і тому в них повинні використовуватися набагато довші ключі, ніж ті, які використовуються в симетричних криптосистемах, для забезпечення еквівалентного рівня захисту. Це відразу ж позначається на обчислювальних ресурсах, необхідних для шифрування. RSA перетворився в промисловий стандарт алгоритму з асиметричними ключами, який використовується в бізнесі для цифрового підпису та шифрування.

Симетричні і асиметричні системи шифрування мають кожна свої достоїнства і недоліки. Недоліки симетричною системи шифрування полягають в складності заміни скомпрометованого ключа. Криптостійкість довжини ключа в 128 біт симетричною системи відповідає ключ у 2304 біта асиметричною.

В даний момент поширення набули системи шифрування, які використовують комбінований алгоритм, що дозволяє при високій швидкості шифрування, властивою AES використовувати відкриту пересилання ключів шифрування (як в RSA).

Для підвищення швидкості асиметричного шифрування, генерується тимчасовий симетричний ключ для кожного індивідуального повідомлення. Текст починає шифруватися з використанням цього тимчасового симетричного сеансового ключа. Потім цей сеансовий ключ шифрується за допомогою відкритого асиметричного ключа одержувача і асиметричного алгоритму шифрування. Оскільки сеансовий ключ набагато коротше самого повідомлення час його шифрування буде порівняно невеликим. Після цього цей зашифрований сеансовий ключ разом із зашифрованим повідомленням передається одержувачу.

Відкритий ключ передається по відкритому (незахищенному) каналу, і використовується для завантаження даних. Секретний же ключ зберігається тільки у власника, і використовується для розширювання будь-яких даних, зашифрованих ключем. Таким чином, ми можемо передати відкритий ключ кому завгодно, і отримаємо необхідну інформацію про них.

Даний алгоритм являється досить надійним та саме головне, що швидким. У ході роботи ми будемо використовувати саме алгорит RSA. Головна перевага алгоритму RSA - відкритий ключ і знання алгоритму шифрування неможливо повторити закодованим повідомленням, на базі алгоритму RSA працює програма шифрування PGP, реалізуються хэш-функції (элетронно-цифровая подпись).

Веб-додатки дуже вразливі на сьогоднішній день та потребують ретельнішого захисту. Для реалізації чату між двома користувачами найбільш вдалим буде шифрування повідомлень за допомогою алгоритму RSA. З асиметричним шифруванням досить легко взаємодіяти і за допомогою RSA ми створюємо захист, який нам і потрібен, щоб повідомлення між двома користувачами залишалися конфіденційними.

Було обрано найбільш зручні програмні мови та додатки, які підходять до нашої системи.

Розділ 3. ПРОГРАМНА РЕАЛІЗАЦИЯ МОДУЛЯ, ЯКИЙ РЕАЛІЗУЄ ПЕРЕВІРКУ ЗАХИЩЕНОСТІ ІНФОРМАЦІЇ ВЕБ-ДОДАТКІВ ВІД МЕРЕЖЕВИХ АТАК

3.1 Структура веб-додатку

Найчастіше веб-додатки складаються як мінімум з трьох основних компонентів.

Клієнтська частина веб додатка - це графічний інтерфейс. Це те, що ви бачите на сторінці. Графічний інтерфейс відображається в браузері. Користувач взаємодіє з веб-додатком саме через браузер, клацаючи по посиланнях і кнопках.

Серверна частина веб-додатка - це програма або скрипт на сервері, обробна запити користувача (точніше, запити браузера). Найчастіше серверна частина веб-додатки програмується на PHP. При кожному переході користувача по посиланню браузер відправляє запит до сервера. Сервер обробляє цей запит, викликаючи деякий PHP-скрипт, який формує веб-сторінку, описану мовою HTML, і відсилає клієнтові по мережі. Браузер тут же відображає отриманий результат у вигляді чергової веб-сторінки.

База даних (БД, або система управління базами даних, СУБД) - програмне забезпечення на сервері, що займається зберіганням даних і їх видачею в потрібний момент. У разі форуму або блогу, збережені в БД дані - це пости, коментарі, новини, і так далі. База даних розташовується на сервері. Серверна частина веб-додатку (тобто, PHP скрипт) звертається до бази даних, витягуючи дані, які необхідні для формування сторінки, запитаної користувачем.

Спілкування клієнта з сервером відбувається за протоколом HTTP. Основа цього протоколу - це запит від клієнта до сервера і відповідь сервера клієнту.

Для запитів зазвичай використовують методи GET, якщо ми хочемо отримати дані, і POST, якщо ми хочемо змінити дані. Ще в запиті вказується Host (домен сайту), тіло запиту (якщо це POST-запит) та багато додаткової технічної інформації.

Сучасні веб-додатки використовують протокол HTTPS, розширену версію HTTP з підтримкою шифрування SSL / TLS. Використання шифрованого каналу

передачі даних, незалежно від важливості цих даних, стало хорошим тоном в інтернеті.

Є ще один запит, який робиться перед HTTP. Це DNS (domain name system) запрос. Він потрібен для отримання ip-адреси, до якого прив'язаний запитуваний домен. Ця інформація зберігається в браузері і ми більше не витрачаємо на це час.

Це, мабуть, основні компоненти більшості веб-додатків. Графічно схему їх взаємодії можна представити так: (див. рис.3.1.)

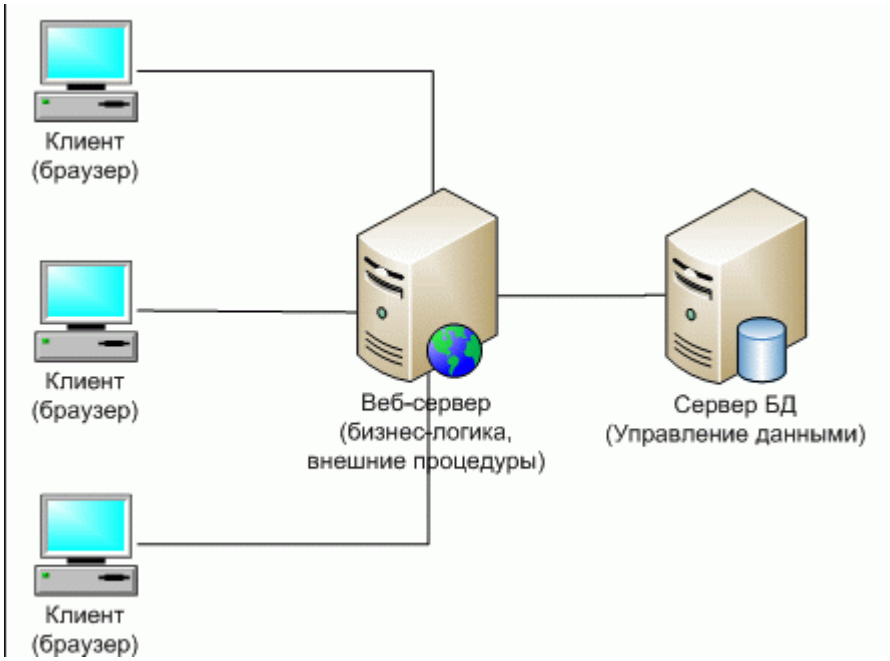

Рис.3.1. Структура веб-додатків

- Браузер через Інтернет відсилає HTTP-запити веб-сервера
- Веб-сервер викликає PHP-скрипт, написаний розробником веб-додатки
- PHP-скрипт звертається до бази даних, якщо це потрібно

В результаті PHP-скриптов повертає клієнту веб-сторінку, яку і відображає браузер.

3.2 Розробка структури та інтерфейсу модуля

У ході роботи для демонстрування модуля, який забезпечує захист нашого продукту був розроблений веб-додаток, який створювався для тестування коду програмного модуля з підвищення від мережевих атак.

Модуль має можливість сканувати продукт та попередити користувача про загрозу, яка може бути небезпечною. Структуру небезпеки можна розглянути на рис. 3.1.

Рис.3.1. Сценарій взлому інформації користувача

У ході роботи спочатку був розроблений інтерфейс. Була розроблена форма для клієнтів, які будуть користуватися даним модулем. В результаті обрали оптимальні технології, які використовувалися при створенні веб-додатка , інтерфейс якого зображено на рис. 3.2.

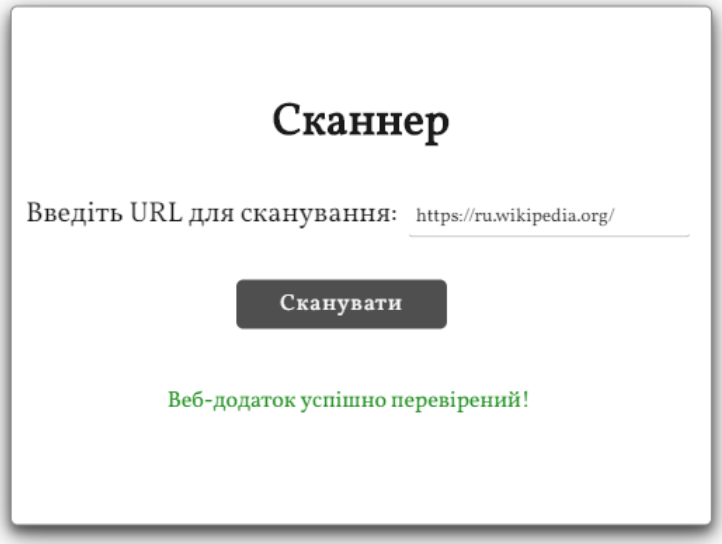

Рис. 3.2. Інтерфейс модуля

Для створення інтерфейсу використовувались такі технології :

- Html
- Css
- VueJS
- Node.js
- Express

Відкривши додаток, користувач повинен знати, що робити без використання інструкцій і довідок. Навігацію, розмір і стиль іконок потрібно проектувати так, щоб користувач не відчував незручність. Задоволення, яке отримує користувач при використанні програми - це важливий фактор. Якщо додаток інтерактивний і корисний, то до нього будуть повертатися знову.

Оптимальність інтерфейсу і зручність навігації являється дуже важливою темою. Модуль має бути зручний для розуміння користувачам там має містити якомога зручніший інтерфейс. У веб-додатку використовувалась блокова верстка. Цей спосіб є найбільш вдалим для створення структури веб-додатка, яка здійснюється за допомогою контейнера (див. рис. 3.2).

```
<div class="container">
    <div class="scanner">
        <h1>Сканнер</h1>
        <label>Введіть URL для сканування:</label>
        <input type="text"  id="scanner-input" placeholder="URL..." required>
        <br>
        <button id="btn">Сканувати</button>

        <div id="result"></div>
    </div>
</div>
<script src="index.js"></script>
```

Рис 3.3. Використання блокової верстки

3.3 Розробка програмного продукту

У Ході створення модулю використовувався метод шифрування RSA, який дає змогу шифрувати нашу інформацію для того, щоб уникнути крадіжку.

Міжсайтові сценарії (XSS) - одна з найвідоміших вразливостей веб-додатків. Він навіть має спеціальну главу в проекті OWASP Top 10, і це дуже вразлива програма в програмах з помилками. Сканер отримує посилання від користувача та сканує веб-сайт на наявність уразливості XSS, вводячи шкідливі

сценарії у місце введення. Вприскування відбувається в безголовому браузері під назвою Chromium і контролюється автоматикою Puppeteer. Це працює у два етапи. Знайти ціль: на цьому першому кроці інструмент намагається визначити всі місця на сторінці, включаючи ін'єкційні параметри у формах, URL-адресах, заголовках тощо. Тест на XSS: для кожного місця, виявленого на попередньому кроці, сканер намагатиметься виявити, чи параметри вразливі для міжсайтового сценарію. Інструмент вводить шматок коду JavaScript, включаючи деякі спеціальні символи HTML (>, <','), і він спробує перевірити, чи повертаються вони на сторінці відповідей без санітарної обробки. Якщо інструмент виявить хоча б одну вразливість, він поверне, що веб-сайт має вразливість XSS. Даний код можна використовувати також для свого веб-додатку, який буде постійно робити перевірку вразливостей у вашій системі. Після розробки інтерфейсу користувача, перейдемо до програмної частини веб-додатка, частини якого можна виділити:

- Використання папки node_modules, яка використовується для встановлення залежностей веб-додатка перед його запуском;
- Файл scanner.css, який містить в собі опис зовнішнього вигляду документа, написаного з використанням мови розмітки;
- Файл scanner.html, в якому знаходиться контейнери веб-розмітка нашого додатка, яка дозволяє нам побудувати сторінку для користувача;
- Папка js, які містить у собі файли з розширенням .js і за допомогою коду, який знаходиться в них ми створюємо функціонал нашого веб-додатка. Наприклад, генерація публічних ключів. Після цього можна розглянути детальніше функціонал наших .js розширень.
- Проаналізуємо файл page.js. У цьому файлі ми починаємо з взаємодії з фреймворком vuejs, який дозволяє нам надати додаткового функціоналу на стороні клієнта. Після цього користувач має змогу самостійно перевірити свій веб-додаток просто ввівши url. Увівши свій url ми маємо змогу натиснути клавішу "Сканувати". Давши змогу просканувати веб-додаток, який ввели, ми маємо змогу отримати результат з позитивним або

негативним результатом;

- Файл server.js, у якому ми створюємо функціонал для перевірки вразливостей веб-додатка, якого ми хочемо перевірити.

Використання коду та удосконалення програмного продукту використовуються технології, які допомагають зробити веб-додаток швидким та надійним. (див. рис. 3.4.)

```
const http = require('http');
const express = require('express');
const bodyParser = require('body-parser');
const queryString = require('query-string');
const puppeteer = require('puppeteer');

const app = express();
const server = http.createServer(app);
app.use(bodyParser.urlencoded({ extended: true }));



app.get('/scanner', async(req,res)=>{
  let globalURL = req.query.url
  console.log('get request')
  let hasXSS = await check_xss(globalURL)
  res.send(hasXSS)
})
```

Рис. 3.4. Частина коду , який використовується для удосконалення програмного продукту за допомогою стеку технологій

У даній програмі також будемо використовувати дві технології npm та git.

npm - найбільший у світі реєстр програмного забезпечення. Реєстр містить більше 800 000 пакетів кодів. Nmp використовується для складання та запуску пакетів у веб-додатку. Багато організацій також використовують npm для управління приватним розвитком. Git - розподілена система управлінням версіями. Після того, як веб-додаток був розроблений, потрібно скласти усі його модулі, для цього використаємо систему.

За допомогою командного рядка можна запустити нашу програму, потрібно використати команду npm start і запустить наш веб додаток для тестування на локальному сервері.

У модулі також використовується Express. Express – це мінімалістичний та гнучкий веб-фреймворк для додатків NodeJS, що надає великий вибір функцій

для веб-додатків. Додає можливість значно прискорити ваш веб-додаток та розробити гарний код.

Програмний продукт розділений на 2 частини. Перша частина використовується для клієнта, який має змогу перевірити свій веб-додаток через URL. Після натискання кнопки “Сканувати” йде запит до веб-додатка, отримуємо дані з сервіса та робимо перевірку на вразливість. (див. рис.3.5)

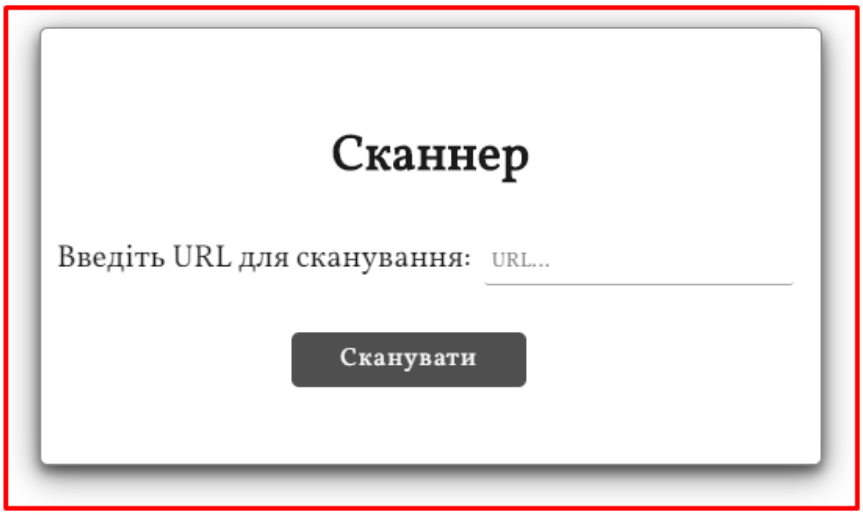

Рис. 3.5. Зображення першої частини програмного продукту

Спочатку ми отримуємо після натискання клавіши малюнок, який рухається по колу та дає змогу нам зрозуміти, що наш веб-додаток сканує інший веб-додаток та шукає вразливості. (див.рис.3.6)

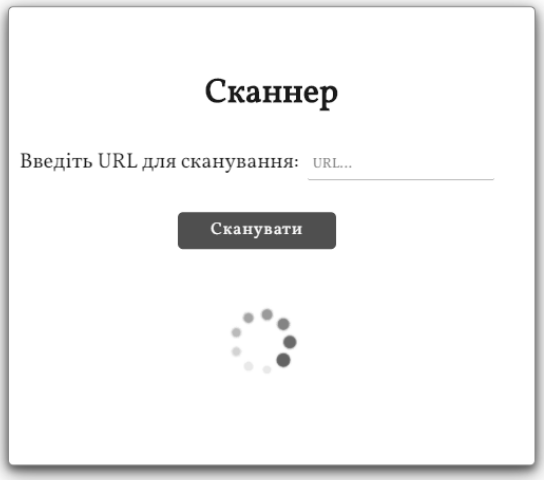

Рис.3.6. Зображення пошуку вразливостей

Після того, як ми отримуємо результат, ми можемо виводити інформацію про те, що не так у модулі, який ми сканували, для користувача результат ми записуємо в окремий блок. (див. рис. 3.7)

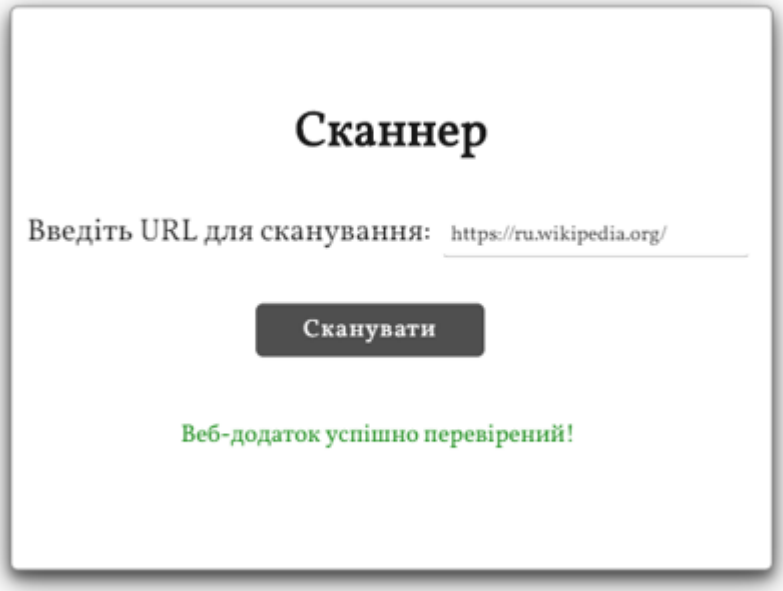

Рис. 3.7. Зображення пошуку вразливостей

Даний програмний продукт також являється адаптивним та кроссбраузерним. Це ми можемо зрозуміти з рис.3.8.

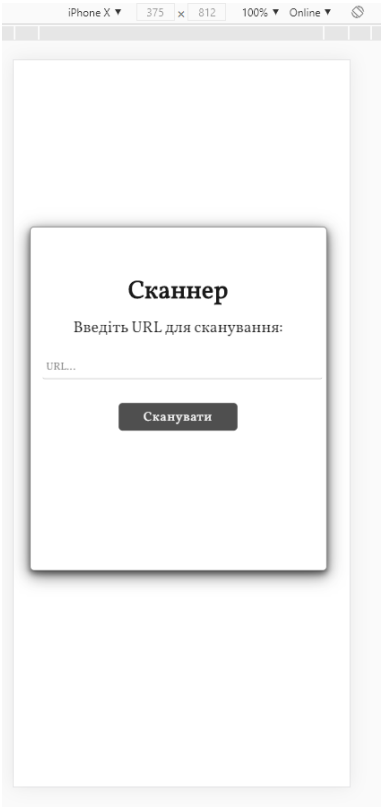

Рис.2.8. Зображення програми на мобільному пристрої iphone X

Друга частина складається з того, що ми можемо підключити наш модуль до будь-якого проекту та перевірити наявність помилок. Так як використовуються сучасні технології, то у проектах маємо також їх використовувати та ініціалізувати для того, щоб перевірка була точною та швидкою. Даний модуль також може пришвидшити весь проект, у якому ви його використовуєте.

З самого початку передаємо технології у змінні, які будемо використовувати , вони повинні бути налаштовані. (див.рис.3.9)

```
const http = require('http');
const express = require('express');
const bodyParser = require('body-parser');
const queryString = require('query-string');
const puppeteer = require('puppeteer');

const app = express();
const server = http.createServer(app);
app.use(bodyParser.urlencoded({ extended: true }));
```

Рис. 3.9. Підключення змінних для використання технологій

Після цього потрібно перенести функцію та налаштувати потрібні параметри для модуля. (див. рис. 3.10)

```
async function check_xss(url) {
    const browser = await puppeteer.launch({ headless: false,defaultViewport: null, args: ['--start-maximized'] })
    const page = await browser.newPage();
    await page.goto(url,{waitUntil: "networkidle2"});
    const formsArray = await page.$$('form');
    const page2 = await browser.newPage();
    let isVulnerable = false;
    page2.on('dialog', async dialog => {
      isVulnerable=true;
      dialog.accept();
    });
    for (script in MALICIOUS_SCRIPT){
      let newUrl= check_url(url,MALICIOUS_SCRIPT[script]);
      if (newUrl != ''){await page2.goto(newUrl);}
      if (isVulnerable) {
        browser.close();
        return true;
      }
      for (i in formsArray){
        try {
          await page2.goto(url, {waitUntil: "networkidle2",timeout: 60000})
          let inputsArray = await formsArray[i].$$eval('input[type="text"],input[type="search"],input:not([type]),textarea', inputs => inputs.map(input => input.id ? '#'+input.id : '.' + input.className));
          for(input in inputsArray){
            let selector = inputsArray[input];
            if(selector.charAt(0)=='.'){selector = selector.split(" ")};
            await page2.type(selector,MALICIOUS_SCRIPT[script],{delay: 20});
          }
          let btnsArray=await formsArray[i].$$eval('input[type="submit"],button[type="submit"]', subs => subs.map(sub => sub.id ? '#'+sub.id : '.'+sub.className));
          let btn=btnsArray[0].split(" ");
          await page2.click(btn);
          newUrl= check_url(page2.url(),MALICIOUS_SCRIPT[script]);
          if (newUrl != ''){await page2.goto(newUrl);}
        } catch (error) {
          browser.close();
          return error.message;
          }
      }
      await new Promise(resolve => setTimeout(resolve, 2000));
      if (isVulnerable) {
        browser.close();
        return true;
      }
    }
    browser.close();
    await new Promise(resolve => setTimeout(resolve, 2000));
    if (!isVulnerable){return false;}
    else {return true;}
}
```

Рис. 3.10. Зображення функції для перевірки вразливостей

Після манупуляцій можна перейти до перевірки загроз, які можуть бути використані для присоєння конфіденційної інформації.

У ході тестування були зроблені тести, які можуть показати нам швидкість сканування веб-додатків.

Таблиця 3.1

Швидкість сканування веб-додатків за домогою модуля

Назва:	Час перевірки:
https://ru.wikipedia.org/	44 секунди
http://sopilochka-studios.com/	11 секунд
http://fcavangard.com/	14 секунд

Після повторного тестування з використанням інших модулів, які дають змогу захистити продукт звертаємо увагу на швидкість роботи модулю, який дає нам більше можливостей та може використовуватися у більшій кількості веб-додатків, що дає змогу бути йому більш гнучким.

Таблиця 3.2

Швидкість сканування розробленого модулю та інших

Назва:	Час сканування:
Розроблений модуль	34 секунди
One button scan	52 секунди
ASafaWeb	59 секунд
Snyk	1 хвилина 10 секунд
CSP	1 хвилина 4 секунди

3.4 Результати реалізації модуля

У ході розробки модуля було створено можливість захисту інформації від мережевих атак веб-додатків. Було створено дві частини, які дають змогу перевірити власний веб-додаток за допомогою використання потрібних технологій та коду, який буде надано для користувачів або веб-додаток, який можемо знайти через інтернет та зробити перевірку через URL.

Розроблений веб-додаток являє собою досить зручний спосіб користування та може використовуватись багатьма користувачами та компаніями для збереження конфіденційності інформації.

У модулі також використовувалось шифрування методом RSA для того, щоб забезпечити захист інформації та коду, який використовується у модулі, щоб уникнути крадіжку даних.

Веб-додаток, який був створений у ході дипломної роботи дає змогу перевірити наявність загроз у с власному веб-додатку або у іншому. Структура інтерфейсу використовувалась блокова верстка. Для реалізації блокової верстки використовувались HTML,CSS та були використані такі технології, як :

- JavaScript
- NodeJS
- VueJS
- Express
- HTML5
- CSS3

Створення та розробка модулю включає:

- твердження початкового технічного завдання на розробку модулю;
- визначення структурної схеми модулю;
- розробка програмного коду, модулів, бази даних і інших елементів веб-додатка необхідних в проекті;
- тестування і розміщення веб-додатка в мережі інтернет.

Перевагою використання створеної програми є те, що вона дає змогу залишити інформацію користувачів конфіденційною та цілісною. Програма є доступною для усіх користувачів та легкою для розуміння інтерфейсу. Модуль придатний для використання людиною, яка не має глибоких знань в області спілкування в чаті та може бути використано на практиці. Було використано багато технологій, які були розбиті на модулі та які дають змогу зробити швидшими веб-додатки,

які потребують оброки інформації та можуть залишити її конфіденційною, доступною та цілісною.

ВИСНОВКИ

У ході розробки веб додатка було створено можливість захисту інформації в веб-додатках від мережевих атак. Був реалізован веб-додаток, який дає змогу тестувати модуль для захисту інформації. Були розглянуті нормативно-правові документи із захисту інформації, було проведено аналіз вразливостей веб-додатків. Проаналізувавши існуючі методи шифрування , можна зробити висновок , що для реалізації шифрування модулю підходить метод RSA. Для розробки програмного продукту був проведений аналіз технологій , які найбільше підійдуть для розробки та захисту повідомлень в веб-додатку. Розробка відбувалась на основі технологій JavaScript, NodeJS та на платформі VueJS , HTML, CSS, Express.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sankar R. Burpsuite – A Beginner’s Guide For Web Application Security or Penetration Testing [Електронний ресурс] / Ravi Sankar. – 2018. – Режим доступа: World Wide Web. — URL: https://kalilinuxtutorials.com/burpsuite/

2. Ricca F. https://dl.acm.org/citation.cfm?id=381476 [Електронний ресурс] / F. Ricca, P. Tonella. – 2001. – Режим доступа: World Wide Web. — URL: https://dl.acm.org/citation.cfm?id=3814763.

3. K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and Don’ts of Client Authentication on the Web. – 2011. - Режим доступа: World Wide Web. — URL: https://pdos.csail.mit.edu/papers/webauth:sec10.pdf.

4. J. Bercegay. Double Choco Latte Vulnerabilities. http://www.gulftech.org/ ?node=research&article id=00066-04082005, April 2005.

5. N. Jovanovic. txtForum: Script Injection Vulnerability. http://www.seclab. tuwien.ac.at/advisories/TUVSA-0603-004.txt, March 2006.

6. . A. Klein. Cross Site Scripting Explained. Technical report, Sanctum Inc., 2002

7. Про захист персональних даних [Електронний ресурс] / Закон України – Режим доступа: World Wide Web. — URL: https://zakon.rada.gov.ua/laws/show/2297-17 – Загл. з екрану (дата звернення: 9 травня 2015).

8. Стакснет [Електронний ресурс]. – Режим доступа: World Wide Web. — URL: https://uk.wikipedia. org/wiki/Стакснет – Загл. з екрану (дата звернення: 8 травня 2019).

9. Нормативні документи [Електронний ресурс]. – Режим доступа: World Wide Web. — URL: https://testportal.gov.ua/normdokzno/

10. Шифрування методом RSA , його властивості та інші дані [Електронний ресурс]. – Режим доступа: World Wide Web. — URL: https://works.doklad.ru/view/n5VwDdCV88Y.html

11. Скляров, Д.В. Искусство защиты и взлома информации / Д.В. Скляров. — СПб. : БХВ-Петербург, 2004. — 36 с.

12. Іванов М. А. ,Зенін О. С., Стандарт криптографічного захисту – AES. Кінцеві поля. М.: КУДИЦ – ОБРАЗ, 2003. 171 с.

13. New types of cryptanalytic attacks using related keys./ Biham E. // Advances in Criptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer – Verlag, 1993 – P. 398 – 409.

14. CSS стили наше все [Електронный ресурс]. — Режим доступа: World Wide Web. — URL:https://www.w3.org/Style/Examples/007/figures.pl.html/– Загл. з екрану (дата звернення: 9 травня 2019).

15. Шнаєр В. Прикладна криптографія. Протоколи, алгоритми, вихідні тексти на мові С. М.: Видавництво «ТРИУМФ», 2002.703 с.

16. Столінгс В. криптографія і захист мереж: принципи і практика. М.: видавничий дім «Вильямс», 2001. 677 с.

17. Инструменты Kali Linux [Електронний ресурс] – Режим доступу до ресурсу: https://kali.tools.

18. Using Burp Proxy [Електронний ресурс] // 2018 – Режим доступу до ресурсу: https://support.portswigger.net/customer/portal/articles/1783119-usingburp-proxy.

19. Web Application Risk – the Threat of and Solution to Sensitive Data Exposure [Електронний ресурс] – Режим доступу до ресурсу: https://www.immuniweb.com/blog/OWASP-sensitive-data-exposure.html.

20. Top 10-2017 A6-Security Misconfiguration [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Top_10- 2017_A6-Security_Misconfiguration.

21. Authentication Hacking: What are Authentication Hacking Attacks? [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://www.acunetix.com/websitesecurity/authentication/.

22. Charan H. Broken Authentication and Session Management—part I [Електронний ресурс] / Hari Charan. – 2017. – Режим доступу до ресурсу:

https://medium.com/@grep_security/broken-authentication-and-sessionmanagement-part-i-50e760c9f599.

23. Testing for CSRF (OTG-SESS-005) [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005).

24. Cross Site Scripting (XSS) Attack Tutorial With Examples, Types & Prevention [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/cross-site-scripting-xss-attack-test/.

25. Методы защиты от CSRF-атаки [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://habr.com/ru/post/318748/.

26. Cross-Site Request Forgery (CSRF) [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.owasp.org/index.php/CrossSite_Request_Forgery_(CSRF).

27. SQL_Injection_Prevention_Cheat_Sheet [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md.

28. How to Prevent SQL Injection Attacks [Електронний ресурс] // eSecurityPlanet. – 2018. – Режим доступу до ресурсу: https://www.esecurityplanet.com/threats/how-to-prevent-sql-injection-attacks.html.

29. Уязвимости веб-приложений [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.ptsecurity.com/upload/corporate/ruru/analytics/Web-Vulnerabilities-2019-rus.pdf.

30. Web Server Vulnerabilities Attacks: How to Protect Your Organization [Електронний ресурс] // Tech Funnel. – 2018. – Режим доступу до ресурсу: https://www.techfunnel.com/information-technology/web-server-vulnerabilitiesattacks-how-to-protect-your-organization/