НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«До захисту допущено»
В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)

“_____”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

на тему: Захист інформації в корпоративних порталах на базі Liferay_____

_____

_____

Виконав (-ла): студент (-ка) __4__ курсу, групи __ФБ-51_____
(шифр групи)

Данилюк Павло Дмитрович_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доц. к.т.н. Литвинова Тетяна Василівна_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент Баранік Д. О._____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

Данилюк Павло Дмитрович_____
(прізвище, ім'я, по батькові)

1. Тема роботи Захист інформації в корпоративних порталах на базі Liferay
_____
_____,
науковий керівник роботи Литвинова Тетяна Василівна, доц. к.т.н_____
_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____ » 2019 р. № _____

2. Термін подання студентом роботи 10 червня 2019 р.

3. Вихідні дані до роботи _____
_____
_____

4. Зміст роботи_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)_____
_____
_____
_____

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка

Студент _____ __Данилюк П. Д._____
(підпис) (ініціали, прізвище)

Науковий керівник роботи _____ ___Литвинова Т.В.___
(підпис) (ініціали, прізвище)

РЕФЕРАТ

бакалаврської дипломної роботи Данилюка Павла Дмитровича на тему “Захист інформації в корпоративних порталах на базі Liferay”

Кваліфікаційна робота містить: 62 сторінки, 6 рисунків та 9 джерел.

Метою дипломної роботи є дослідження корпоративних порталів на базі Liferay на захищеність та формування оптимальних вирішень проблем безпеки.

Завданням роботи є тестування корпоративних порталів на базі Liferay на наявність ряду вразливостей, аналіз та систематизація варіантів вирішення проблем безпеки самою платформою та опис оптимальних варіантів захисту від деяких загроз.

Об’єктом дослідження є корпоративні портали на базі Liferay. Предметом дослідження є захищеність корпоративних порталів на базі Liferay.

Наукова новизна полягає в тому, що в результаті роботи було детально протестовано корпоративні портали на базі Liferay на наявність певних вразливостей, систематизовано вирішення проблем безпеки самим Liferay та сформовано ряд рекомендацій для розробника щодо покращення захищеності порталу.

Практичне значення в тому, що результати роботи можуть бути застосовані Liferay розробником в процесі створення корпоративного порталу, з метою покращення захищеності його веб-застосунку.

КОРПОРАТИВНИЙ ПОРТАЛ, LIFERAY, ПОРТЛЕТ, SERVICE BUILDER.

ABSTRACT

to the bachelor thesis by Danyliuk Pavlo Dmytrovych on “Information security in enterprise portals based on Lifery”

The total amount of work: 62 pages, 6 illustrations, 9 sources.

The purpose of the thesis based on security researching of Lyferay’s enterprise portals and optimal security solutions.

The task of the thesis is to test Liferay's enterprise portals for vulnerabilities, to analyze and systematize solutions for security problems by the platform and to describe the best options for protection from the threats.

The subject of the thesis is the security of enterprise portals based on Liferay.

The scientific novelty is that as a result of the work, Liferay-based corporate portals have been tested for some vulnerabilities, Liferay's security solution has been systematized, and a number of recommendations have been developed for the developer to improve the security of the portal.

The practical importance of the thesis is the results of the work can be used by Liferay’s developer in the process of creating an enterprise portal, in order to improve the security of the web application.

ENTERPRISE PORTAL, LIFERAY, PORTLET, SERVICE BUILDER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Корпоративний портал - це програма, яка дозволяє компанії об’єднати внутрішню та зовнішню інформацію, а також надає внутрішнім і зовнішнім користувачам один єдиний спосіб доступу до інформації, необхідної для прийняття обгрунтованих бізнес-рішеннь. Корпоративні портали уніфікують доступ до даних, інформації та додатків, а також подають їх бізнесу користувачам в зручному форматі.

Liferay — платформа для будівництва бізнес-рішень(корпоративних порталів), яка об'єднує різні додатки в єдиний інформаційний простір. Проект Liferay, розповсюджуваний під Open Source ліцензією, цілком успішно конкурує з більшістю комерційних рішень.

Портлети Liferay – Веб-програми на Liferay Portal. Подібно до багатьох веб-програм, портлети обробляють запити та генерують відповіді. У відповіді портлет повертає вміст (наприклад, HTML, XHTML) для відображення в браузерах.

SQL ін’єкції — атака спрямована на введення на стороні клієнту SQL команд, що надсилаються на сервер для несанкціонованого виконання.

ВСТУП

У зв'язку з безперервним розвитком мережі Інтернет та інформаційних технологій в цілому веб-застосунки стали невід'ємною частиною корпоративної інформаційної системи будь-якої сучасної організації незалежно від роду її діяльності та сфери економіки. Не тільки комерційні компанії створюють і розвивають власні веб-застосунки: державний сектор активно включається в процес розвитку веб-сервісів, що забезпечують надання онлайн-послуг.

В наш час активного поширення набирають корпоративні портали. Сьогодні вони стали незамінною частиною бізнесу, забезпечуючи співробітників єдиною точкою доступу до інформації, інструментами управління бізнес-процесами та засобами для обміну даних. Open Source проект Liferay досить успішно розповсюджується та конкурує із більшістю комерційних проектів. Liferay являє собою веб-платформу для створення бізнес-рішень, які об’єднують різні сервіси та застосунки в єдиний інформаційний простір.

Liferay досить зручний спосіб для розробника швидко та ефективно реалізувати деяке бізнес-рішення. Liferay Portal надає досить широкий функціонал можливостей для полегшення роботи розробнику та його фокусуванні на бізнес-логіці. Але в той же час безпека сервісу не завжди є пріоритетною метою для розробника, що в деякій мірі переносить відповідальність на саму платформу. Очевидною є важливість використання захищеної платформи при побудові корпоративного порталу.

Для розробника корпоративних порталів не завжди очевидним є розуміння того, що платформа, в плані безпеки, вже реалізувала, а про що йому прийдеться потурбуватись самому. Відсутність чітких рекомендацій та прикладів реалізації захищеності порталу є серйозною проблемою для Liferay розробників.

Актуальність роботи зумовлюється тим, що корпоративні портали в наш час стали незамінною частиною бізнесу. Liferay Portal сьогодні успішно конкурує на ринку з більшістю комерційних проектів. Liferay розробнику часто важко

зрозуміти яку роботу із захищеності порталу потрібно виконати йому, а яку — реалізовує сама платформа. Представлена робота систематизує та пропонує оптимальні вирішення проблем безпеки для ряду вразливостей.

Метою роботи є дослідження корпоративних порталів на базі Liferay на захищеність та формування оптимальних вирішень проблем безпеки.

Завданням роботи є тестування корпоративних порталів на базі Liferay на наявність ряду вразливостей, аналіз та систематизація варіантів вирішення проблем безпеки самою платформою та опис оптимальних варіантів захисту від деяких загроз.

Об’єктом дослідження є корпоративні портали на базі Liferay.

Предметом дослідження є захищеність корпоративних порталів на базі Liferay.

Наукова новизна полягає в тому, що в результаті роботи було детально протестовано корпоративні портали на базі Liferay на наявність певних вразливостей, систематизовано вирішення проблем безпеки самим Liferay та сформовано ряд рекомендацій для розробника щодо покращення захищеності порталу.

Практичне значення в тому, що результати роботи можуть бути застосовані Liferay розробником в процесі створення корпоративного порталу, з метою покращення захищеності його веб-застосунку.

1 ТЕОРЕТИЧНИЙ ОГЛЯД

1.1 Безпека веб-застосуніків

1.1.1 Безпека в мережі Інтернет

Сучасний світ несе в собі тисячі загроз і потенційних небезпек майже на кожному кроці і в кожен момент часу. Всесвітня мережа, що стала невід'ємною частиною нашого життя, не є винятком. Кіберзлочинність зараз розвинена як ніколи, адже кількість сайтів в інтернеті перевищує сотні мільйонів, а зловмисник у мережі може легко залишатися абсолютно анонімним.

Кількість загроз зростає пропорційно зростанню кількості сайтів, проте як показала багаторічна практика, 99% атак відбуваються через десяток стандартних помилок валідації даних, або банально, через недбальство системних адміністраторів, які використовують налаштування і паролі, встановлені за замовчуванням.

Окрім наявності потенціальних загроз на захищеність неодмінно впливає як наявність чіткої політики безпеки, так і спеціалізованої документації, що допоможе розробникам при побудові рішення.

Важливо зазначити, що з кожним роком збільшується і обсяг призначених для користувача даних в мережі, адже сьогодні онлайн можна зробити практично все: від оплати комунальних послуг до покупки авіаквитків. Одночасно з цим зростає і кількість загроз. У 2014 році на весь світ прогриміли Heartbleed, Shellshock, а в минулому році відбулися зливи баз електронних пошт та даних працівників відомих компаній та багато інших подій у сфері веб-безпеки.

Та чи росте рівень знань про те, як протистояти кіберзагрозам? Особливо враховуючи, що сьогодні в результаті зламу облікового запису користувача чи при проведенні успішної атаки можна втратити набагато більше, ніж на зорі інтернету? Багато експертів вважають, що величезна кількість розробників веб-застосунків до цих пір нехтують елементарними правилами безпеки. Цю

ситуацію погіршує необережність користувачів, що фактично зводить нанівець всі зусилля, що робляться онлайн-сервісами для підвищення безпеки.

Можна констатувати, що користувачі як і раніше недостатньо уважно стежать за своєю безпекою в інтернеті. Так, майже дві третини користувачів онлайн-сервісів коли-небудь ставали жертвами шахрайства. Серед причин постраждалі найчастіше називають простий пароль, скачаний вірус або перехід на шахрайський сайт. Майже в два рази рідше користувачі відзначають, що постраждали через використання одного пароля на декількох сервісах або через те, що відповіли на шахрайське повідомлення. Захищеність самих сервісів від атак, що можуть призвести до шахрайства теж впливає на цю статистику.

При введенні особистих даних (наприклад, логіна та пароля) майже половина користувачів веб-застосунків (пошти, соціальних мереж) не перевіряють наявність безпечного з'єднання. В свою чергу в архітектуру застосунку не завжди покладені технології, що дозволяють зробити цю операції максимально захищено. Важливо давати користувачу можливість ввести тільки достатньо довгий та складний пароль.

Очевидно, що саме розробник визначає ступінь захищеності рішення, над яким він працює. Це визначає як його власні професійні якості, так і інструменти та рішення, якими він буде користуватися під час роботи.

1.1.2 Важливість безпеки в веб-застосунках

Важливо, що при розробці будь-якого веб-застосунку рано чи пізно виникають питання безпеки. Не можна виключити, що та система, яку ви розробляєте, буде якимось чином атакована. Причому, в залежності від складності системи та застосовуваних технологічних рішень, вектори атак і їх наслідки можуть мати найрізноманітніший характер. Можливо, час від часу хтось в команді проводить аналіз безпеки розроблюваного сервісу, проводить моделювання. Куди гірше якщо взяті за основу архітектури рішення з самого початку мають серйозні проблеми з точки зору захищеності. Результати можуть

бути досить лякаючими, якщо таку систему зламують в режимі комерційної експлуатації і доводиться похапцем створювати виправлення для виявленої вразливості. Очевидно, що питання, пов'язані з безпекою краще вирішувати, починаючи з найбільш ранніх етапів створення системи, таких як аналіз вимог і архітектурного моделювання.

При дизайні архітектури веб-застосунків так само вадливо застосувати ряд практик, які допоможуть підвищити їх безпеку. В першу чергу це зниження площі поверхні можливих атак і моделювання загроз. Незважаючи на близьку взаємозв'язок цих двох понять, перший механізм передбачає активне зниження можливостей зловмисника на експлуатацію невідомих вразливостей. Для зниження площі можливих атак можна застосовувати механізми пошарового захисту і принципи найменших привілеїв. Моделювання загроз в свою чергу дозволяє припустити, які компоненти системи можуть бути розглянуті в якості векторів атак.

Якщо ж розглядати саме певні технології як фундамент для захищеної системи, то можна сказати, що в кожної технології є дві сторони. Одна сторона, це властивості, корисні користувачу, які ми використовуємо у самих веб-застосунках; друга сторона — властивості, які може використовувати зловмисник, тобто слабкості технології. Іноді технологія слабка настільки, що її використання не можна виправдати ні за яких умов. Наприклад, функція eval в мові на JavaScript — практично завжди зло. Інші технології використовувати можна, вживаючи необхідні для заходи. Так використання звичних нам полів для вводу користувачем тексту потенційно може відкрити можливість для ін’єкцій будь-якого типу. Але ми знаємо, як з цим боротися і просто повинні вжити необхідних заходів захисту.

Ми повинні знати обидві сторони наших технологій. Як зробити корисний застосунок — це головне для будь-якого програміста. Тому цілком зрозуміло, що основний наш час ми присвячуємо вивченню цієї сторони. Але гарний фахівець розуміє і другу сторону. Ми не зобов'язані знати, як зловмисник буде атакувати,

але ми повинні розуміти, які наші дії можуть привести до появи вразливостей, і як ми можемо цього уникнути.

Вже існує надзвичайна кількість технологій, що можуть полегшити життя розробника і йому не знадобиться створювати велосипед. Звичайно, дуже цікаво зробити щось своє, але винахідництво, що стосується питань безпеки призводить до дуже неприємних наслідків, тому очевидно, що слід використовувати перевірені засоби.

Зараз існує велика кількість різноманітних фреймворків та платформ для розробки. Можна знайти готові рішення майже для будь-якого завдання, причому знайти можна і платні бібліотеки, і бібліотеки з відкритим кодом. Наприклад, існує безліч інструментів, в яких присутні функції, які здійснюють фільтрацію вхідних даних, автентифікацію користувача, перевірку прав доступу, і багато іншого. Це тільки деякі можливості. Більш того, багато сучасних фреймворків вже містять збірки необхідних інструментів.

Іноді при розробці неможливе використання чужих бібліотек. В крайньому випадку можна використати своє власне рішення, ретельно його перевіривши і протестувавши. Та все одно це рішення в більшості випадків буде гіршим за те, що перевірене роками.

В той же час людині властиво робити помилки. При цьому відомо, що саме найбільш очевидні помилки найважче виявити: невдало поставлена крапка з комою може призвести до втрати цінних даних. Для виникнення вразливостей існує величезна кількість причин; щоб уникнути проблем, треба пам'ятати їх всі, треба перевіряти застосунок на їх наявність. Більш того, зловмисники завжди шукають нові шляхи: те, що вважалося безпечним вчора, сьогодні вже може бути вразливе. Частково зменшити проблеми з не обережними помилками і швидко мінливим світом можуть утиліти автоматизованого аналізу. Найчастіше, ці утиліти можуть проводити досить складний аналіз, який у людини зайняв би дуже багато часу. Рішення, подібні до таких утиліт вже вбудовані в сучасні фреймворки, що робить їх використання ще необхіднішим.

Зрозуміло, що використання готових архітектурних рішень, а саме фреймворків — кращий варіант з точки безпеки. Над цими рішеннями зазвичай працює велика кількість людей, що сприяє тому, що в них будуть відсутні помилки, допущені при створенні свого “велосипеду”. Та якщо цих рішень декілька? Очевидно, що кожне архітектурне рішення має свої особливості та в чомусь різні погляди на рішення одних і той самих проблем.

1.2 Веб-платформа Liferay

1.2.1 Корпоративні портали

Одним із основних способів збільшення ефективності бізнесу є організація централізованої комунікації та доступу до інформації. Велику роль в цьому відіграють корпоративні портали

Корпоративний портал - це програма, яка дозволяє компанії об’єднати зовнішню та внутрішню інформацію, а також надає своїм користувачам один єдиний спосіб доступу до інформації, необхідної для прийняття правильнх рішеннь. Корпоративні портали уніфікують доступ до даних, інформації та додатків, а також подають їх користувачам в зручному форматі. Портали використовуються бізнес-користувачами, але включають інструменти IT-адміністрування та мають певний базовий рівень наступних функціональних можливостей:

• Рольове або правове адміністування;

• Управління контентом, керування документами та пошуком тощо;

• Доступ до структурованих даних, таких як звітність чи запити користувачів

Знову популярними стають корпоративні портали (Enterprise Information Portal - EIP), популярність яких, спочатку дещо зменшувалась, але тепер починає набирати нових обертів. Цілком ймовірно, причиною такої популярності порталів

стала поява великої кількості інтернет-сервісів, які дозволили з іншої сторони обійти проблему об'єднання різних груп людей в одному просторі. І на сьогоднішній день портали вже розглядаються не тільки як якась окрема та централізована точка доступу до інформації та сервісів, але і як засіб обміну даними та платформа для комунікації співробітників і клієнтів, а також керування та менеджмент власних бізнес-процесів. Якщо об’єднати все вище сказане, тоді ця суміш призначена покращити та полегшити роботу працівників, оптимізувати керування документами та даним та зменшити витрати для компанії. В свою чергу розробникам портал дає змогу відійти від великої кількості неупорядкованих додатків, зосередившись тільки на розробці та проектуванні якогось одного сервісу.

Корпоративні портали поступово перетворилися на незамінний інструмент бізнесу, забезпечуючи співробітників єдиною точкою доступу до даних, інструментами управління бізнес-процесами та засобами обміну інформацією. Очевидною є важливість використання в їх архітектурі найактуальніших методів захисту від вразливостей.

1.2.2 Liferay

Платформа Liferay Portal компанії Liferay розповсюджується під Open Source ліцензією. Сьогодні вона дуже успішно конкурує з більшістю комерційних рішень на даному сегменті ринку. Liferay являє собою веб-платформу для будівництва бізнес-рішень, яка об'єднує різні сервіси в єдиний інформаційний простір. З його допомогою можна побудувати портали з інтеграцією корпоративних додатків, динамічні веб-сайти, базу знань і соціальні мережі. Співробітникам для доступу до інформації та обміну даними прийдеться використовувати тільки один сервіс. Платформа доступна із вихідним кодом під подвійною ліцензією: GNU GPL і комерційною. Liferay знаходиться на ринку вже тривалий час та користується популярністю по всьому світу. Починаючи з 2011

року аналітичне агентство Gartner в звіті Magic Quadrant for Horizontal Portals відносить Liferay до лідерів, де він знаходиться поруч з програмними рішеннями від Microsoft, IBM, SAP і Oracle. З офіційного сайту портал був скачаний понад чотири мільйони разів, розробники говорять про приблизно 350-500 тисяч установок в організаціях дуже різного призначення. Серед них і компанії зі світовим ім'ям: міністерство оборони Франції, Cisco, Andorra Telecom і багато інших.

Liferay, як і більшість інших порталів, дуже гнучкий і здатний адаптуватись під різні вимоги користувачів. Зразу після установки базова система містить тільки певний оптимальний набір. Решту додаткових функцій можна релізувати за допомогою добавляння модулів, які називються портлетами. У Liferay Marketplace є велике число готових до використання компонентів, частина з яких поширюється безкоштовно.

Liferay написаний на Java і тому працює на будь-якій платформі, для якої доступна Java Runtime Environment і сервер додатків. Підтримується робота з ОС Windows, Unix / Linux, Mac OS X, СУБД (MySQL, PostgreSQL, MSSQL, Oracle, DB2, Sybase, Ingres), службами каталогів LDAP і Active Directory, системами SSO - CAS, PingFederate, OpenSSO, NTML, SiteMinder . Для відправки або отримання пошти можна застосовувати будь-який зовнішній SMTP / S, IMAP / S або POP3 / S сервер, в тому числі можна використовувати і такі як GMail. Плюс десятки інших продуктів для генерації звітів (JasperServer Aperte Reports), системи електронного документообігу (Alfresco ECM), CMS, ERP / CRM (Adempiere), система бізнес-аналітики Pentaho BI, Libre / OpenOffice і багато іншого. Все це дозволяє швидко і з мінімальними витратами впровадити корпоративний портал з потрібними функціями. Також Liferay використовує кілька бібліотек сторонніх розробників - BeanShell, Geo Tools, jCIFS, Unix Crypt і інші (повний список доступний на сторінці Third Party Software).

1.2.3 Можливості Liferay

Портал, створений із застосуванням Liferay, може включати в себе систему управління контентом / веб-сайтом, документообіг (з підтримкою Microsoft Office), засоби комунікації та організації спільної роботи (календар, завдання, оповіщення, wiki, форум, база знань, дошка повідомлень, блоги, соцмережа, опитування, система миттєвого обміну повідомленнями на базі XMPP), управління бізнес-процесами і взаємодією з клієнтами, планування ресурсів, складання звітів і багато іншого. Портал надає повноцінний поштовий веб-клієнт, тому для надсилання та отримання листів стороння програма непотрібна. Система сповіщення і розсилок дозволяє повідомляти користувачів про події за допомогою email, RSS, SMS або будь-якого іншого механізму, що настроюється адміністратором. Бібліотека документів Liferay підтримує протокол Microsoft SharePoint, тобто користувачі можуть працювати з файлами Microsoft Office, що знаходяться на їх локальних дисках або автоматично завантажувати їх в сховище Liferay. Підтримуються версії документа, блок для редагування, автоматична конвертація формату, галерея зображень, пошук, доступ по WebDAV, публікування і багато іншого. Пропонується і свій власний клієнт синхронізації документів з порталом Liferay Sync, що підтримує Drag'n'Drop, журнал роботи змін і відкат до попередніх версій, можливість редагування файлу відразу декількома користувачами. Доступні версії для Windows і Mac OS X.

Передбачені інструменти пошукової оптимізації. Будь-яка інформація в Liferay структурується за допомогою тегів і категорій. Крім цього, при публікації статті можна вказати уривок, який буде доступний для попереднього перегляду, пов'язані ресурси (запис в щоденнику, повідомлення форуму, календар, закладка і т.п.) і права доступу. Для створення контенту пропонується вбудований редактор з функцією перевірки правопису (його можна легко замінити з Marketplace). Сайт може бути багатомовним, при цьому користувач при підключенні отримає сторінку на своїй мові. Зовнішній вигляд порталу, веб або окремої сторінки змінюється за допомогою тем і макетів. Для статей передбачені шаблони, які

створює сам користувач. Каталог програмного забезпечення дозволяє визначити набір ПО, доступного користувачам для завантаження.

Однин сервер Liferay може обслуговувати кілька організацій і сайтів. Рольова система доступу дає користувачам тільки необхідні права. Наприклад, контент-редактор може запропонувати матеріал на веб-сторінку, але вона буде опублікована тільки після підтвердження уповноваженою особою. Аналогічно і користувач може прочитати тільки покладений йому документ. Адміністратор вказує наявні програми в залежності від облікового запису, ролі, групи, організації. Анонімний користувач матиме доступ до якихось базових сторінок, співробітники або бізнес-клієнти - до додаткових. Користувачі можуть самостійно створювати бізнес-процеси і визначати рівні твердження, грунтуючись на вимогах бізнес-логіки і конкретних завдань.

Всі функції Liferay реалізовані за допомогою підключення модулів-портлетів, тому результуюча система містить тільки те, що дійсно необхідно. Після установки з Liferay Marketplace буде доступно більше 60 готових до використання компонентів, частина з яких поширюється безкоштовно. Також портлет і теми можна знайти на інших ресурсах Інтернет і пошуком в каталогах на кшталт sourceforge.net і Google Code.

Портлет можна створювати самостійно на різних мовах програмування Java, PHP, Ruby, Python, фреймворк Grails і інші. Документація і відкритість проекту дозволяє при необхідності легко підключити будь-який додаток.

Всі додатки та інформація за допомогою різних методів (SOAP, REST, RSS, внутрішні API) інтегруються в єдиний інтерфейс, що спрощує роботу з Liferay. Особливої підготовки від користувача і адміністратора не потрібно. Всі дії виконуються за допомогою UI, підтримується Drag'n'Drop, технологія Ajax робить середовище інтерактивним. Зовнішній вигляд і функціональність кабінету, повністю налаштовується як адміністратором, так і самим користувачем за допомогою тем, портлетів і віджетів [2].

Портал може одночасно підтримувати декілька мов інтерфейсу. В останніх версіях підтримуються різні мобільні пристрої, зовнішній вигляд порталу автоматично підлаштовується під розмір екрану.

Сервер Liferay легко масштабується, може працювати в кластері і готовий до застосування в організаціях будь-якого розміру і розгортання в хмарі, наприклад для надання послуг у вигляді SaaS.

Також слід відзначити наявність ще одного продукту - Liferay Social Office. Він має деякі можливості порталу, але порталу, орієнтованого на організацію спільної роботи, з усіма наслідками, що випливають звідси та нюансами в першу чергу з інтерфейсом.

Пропонується дві версії порталу Community і Enterprise Edition. Перша доступна під ліцензією GNU GPL, її можливості дещо кілька урізані в порівнянні з комерційною EE. Але у випадку з Liferay всі функції пов'язані з побудовою порталу залишилися практично недоторканими, недоступні лише деякі додаткові: комерційні плагіни, функції контролю продуктивності, аналітики, аудиту, вбудований генератор звітів. Також відсутня підтримка 24x7. В принципі, самі розробники стверджують, що при певній підготовці на перших порах особливої потреби в покупці EE немає, досить використовувати CE, щоб як мінімум придивитися до Liferay.

1.2.4 Liferay портлети

Веб-програми на Liferay Portal називаються портлетами. Подібно до багатьох веб-програм, портлети обробляють запити та генерують відповіді. У відповіді портлет повертає вміст (наприклад, HTML, XHTML) для відображення в браузерах. В чому ж відмінність портлетів від інших веб-додатків? Одна з ключових відмінностей полягає в тому, що портлети виконуються у частині веб-сторінки. Коли ви пишете портлет, вам потрібно лише потурбуватися про цю програму: решта сторінки - навігація, верхній банер і будь-який інший

глобальний компонент інтерфейсу - обробляються іншими компонентами. Ще одна відмінність полягає в тому, що портлети виконуються тільки на сервері порталу, як і в Liferay. Тому портлети можуть використовувати існуючу реалізацію для керування користувачами, аутентифікації, правами доступу, керування сторінками тощо. Це концентрує вас на розробці основних функцій портлету. Багато в чому написання програми як портлету простіше, ніж написання окремої програми.

Портлети можуть розміщуватися на сторінках користувачами або адміністраторами порталів, які можуть розміщувати декілька різних портлетів на одній сторінці. Наприклад, сторінка сайту компанії може мати портлет календаря для подій самої компанії, портлет оголошень для важливих повідомлень і портлет закладок для посилань, які є цікавими для компанії. Оскільки портал керує макетом сторінки, можна змінити розміри одного або декількох портлетів на сторінці, не змінюючи самого коду цих портлетів. Виконання цього в інших типах веб-додатків потребуватиме перепрограмування самого компоненту. Крім того, один портлет може зайняти всю сторінку, якщо він є єдиним веб-додатком, який вам потрібний на цій сторінці. Тобто можна сказати, що портлети спрощують реалізацію багатьох традиційно болючих проблем, пов'язаних з розробкою веб-додатків [7].

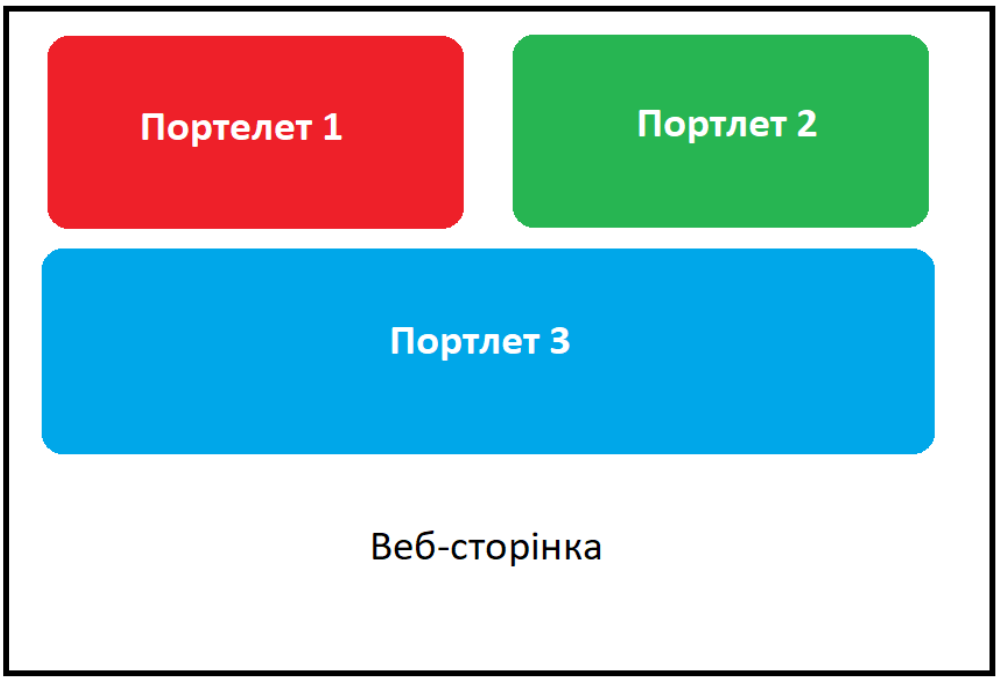

Рисунок 1.1 – Розміщення портлетів на сторінці

1.2.5 Портлети Liferay в точки зору розробки

Портлети обробляють запити в декількох фазах. Це робить портлети набагато гнучкішими, ніж сервлети. Кожна фаза портлету виконує різні операції :

Фаза відображення (Render phase) — генерує весь контент портлету на основі його поточного стану. Коли цей етап виконується в одному портлеті, він також спрацьовує на всіх інших портлетах на тій же сторінці. Фаза відображення виконується тільки тоді коли всі портлети на сторінці завершують фази дії (Action) та події (Event).

Фаза дії (Action phase) – у відповідь на дію користувача виконується деяка операція, яка змінює стан портлету. Фаза дії також може ініціювати події, які обробляються фазою подій. Після фази дії і фази події фаза відображення регенерує вміст портлету.

Фаза події (Event phase) — обробляє події, що запускаються у фазі дії. Після того, як портлет обробить всі події, портал викликає фазу Render на всіх портлетах на сторінці.

Фаза обслуговування ресурсів (Resource-serving) – служить ресурсом, який є незалежним від решти життєвого циклу. Це дозволяє портлету подавати динамічний вміст без запуску фази Render на всіх портлетах на сторінці. Фаза, що обслуговує ресурси, обробляється AJAX-запитами [1].

Також, варто зазначити, що Liferay Portal дозволяє використовувати різні технології для розробки портлетів. Офіційна документації Liferay містить опис, як можливо розробляти портлети за допомогою таких фреймворків та методів як:

- Liferay's MVCPortlet
- Soy Portlet
- Spring MVC
- JavaServer Faces (JSF) Portlets with Liferay Faces
- Automatic Single Page Applications
- Creating Layouts Inside Custom Portlets

1.2.5.1 Liferay MVCPortlet

В нашій дипломній роботі ми будемо розробляти портлети, використовуючи Liferay MVCPortlet [5]. Для повного розуміння дипломної роботи, пропонуємо ознайомитись із деякими нюансами розробки MVCPortlet.

Фреймворк Liferay MVCPorltlet інкапсулює частину складності портлетів, і це робить широко поширені операції простішими. Стандартний MVCPortlet за замовчуванням використовує окремі JSP для кожного режиму портлету: наприклад, edit.jsp призначений для режиму редагування, а help.jsp - для режиму допомоги.

У MVC є три рівня:

• Model: рівень моделей, містить дані програми і логіку для різного роду маніпулювання з ними.

• View: рівень відображення — містить логіку для відображення даних.

• Controller – посередник у шаблоні MVC, контролер містить логіку для передачі і прийому даних між шарми відображення і моделей.

Liferay застосунки поділеної на декілька дискретних модулів. За допомогою Service Builder генеруються моделі, які описуються і задаються в модулі сервісів, а після генерації вони з’являються в модулі api. Це відповідає Model в моделі MVC. View і Controller реалізуються в спільному модулі — веб-модулі.

Якщо реалізовувати всю логіку контроллера в одному -Portlet класі, код може стати громіздким і нечитабельним. Liferay надає клас команд MVC для розбиття функцій контролера:

• MVCActionCommand — використовується для виконання дій над портлетами, викликаються за допомогою URL цих дій.

• MVCRenderCommand — використовується для відображення портлету по певному URL-адресу. Команда використовується для зв’язування і надсилання потрібних даних на JSP.

• MVCResourceCommand — використовується для виконання отримання якихось ресурсів асинхронно, шляхом відповіді на URL-адреси ресурсів.

Варто зауважити, що всі конфігурації керуються в компоненті OSGi класу -Portlet. Незалежно від того, чи потрібно розділити контролер на класи команд MVC, використовується клас компонентів портлету з певним набором властивостей. Приклад простого компонету MVCPortlet:

```
@Component(
    immediate = true,
    property = {
        "com.liferay.portlet.css-class-wrapper=portlet-hello-world",
        "com.liferay.portlet.display-category=category.sample",
        "com.liferay.portlet.icon=/icons/hello_world.png",
        "com.liferay.portlet.preferences-owned-by-group=true",
        "com.liferay.portlet.private-request-attributes=false",
        "com.liferay.portlet.private-session-attributes=false",
        "com.liferay.portlet.remoteable=true",
        "com.liferay.portlet.render-weight=50",
        "com.liferay.portlet.use-default-template=true",
        "javax.portlet.display-name=Hello World",
        "javax.portlet.expiration-cache=0",
        "javax.portlet.init-param.always-display-default-configuration-icons=true",
        "javax.portlet.name=" + HelloWorldPortletKeys.HELLO_WORLD,
        "javax.portlet.resource-bundle=content.Language",
        "javax.portlet.security-role-ref=guest,power-user,user",
        "javax.portlet.supports.mime-type=text/html"
    },
    service = Portlet.class
)
public class HelloWorldPortlet extends MVCPortlet {
}
```

При використанні команд MVC властивість javax.portlet.name є обов’язковою. Ця властивість є однією з двох, які повинні бути включені в кожен компонент команди MVC; вона пов'язує певну комбінацію URL / команду портлету з правильним портлетом.

Також важливою частиною написання власних портлетів є сама структура web-модудя. Можна сказати що ця частина є критично важливою, оскільки OSGI модуль повинене відповідати певній структурі. Приклад наведемо нижче:

```
docs.liferaymvc.web/
    src/main/java/
        com/liferay/docs/liferaymvc/web/portlet/LiferayMVCPortlet.java
    src/main/resources/
        content/
            Language.properties
        META/-INF/resources/
            init.jsp
            view.jsp
    build.gradle
    bnd.bnd
```

Щонайменше, для коректної роботи OSGi модуля, потрібно вказати вказати bundle symbolic name та bundle version:

```
Bundle-Name: Example Liferay MVC Web
Bundle-SymbolicName: com.liferay.docs.liferaymvc.web
Bundle-Version: 1.0.0
```

1.2.5.2 Liferay Service Builder

Веб-додатки без надійної бізнес-логіки або шару збережння даних (persistence) важко назвати додатками. На жаль, написання власного коду на рівні persistence часто займає багато часу. На щастя, Liferay надає нам чудовий інструмент — Liferay Service Builder, можливість створити persistence шар замість розробника. Основним плюсом даного інструменту є гнучкість. При бажанні розробник може доповнити, змінити, переписати шар роботи з базою даних, згенерований Service Builder'ом. Незалежно від того, як розробник створюєте свій код збереження, він може використовувати його для реалізації бізнес-логіки програми.

Service Builder [4] є інструментом генерування коду, створеним Liferay, який дозволяє розробникам визначати власні об'єктні моделі, які називаються сутностями (entities). Service Builder генерує рівень обслуговування через технологію об'єктно-реляційного відображення (ORM), яка забезпечує чітке розділення між моделлю об'єкта та кодом для бази даних. Це звільняє вас від додавання необхідної бізнес-логіки для вашої програми. Service Builder приймає XML-файл у якості вхідних даних і генерує необхідні шари model, persistence та service для програми. Ці шари забезпечують розділення логіки. Service Builder генерує більшу частину загального коду, необхідного для реалізації операцій створення, читання, оновлення, видалення та пошуку у базі даних, що дозволяє зосередитися на більш високих аспектах бізнес логіки. Ось деякі переваги використання Service Builder:

• Інтеграція з Liferay

• Автоматична генерація шарів model, persistence, service

• Автоматично створюються конфігурації Hibernate і Spring

• Вбудована підтримка кешування об'єктів

• Підтримка користувацьких запитів SQL і динамічних запитів

Щоб визначити модель в програмі, та згенерувати для неї базові класи для роботи з базою даних, потрібно налаштувати всього тільки один файл: service.xml.

Щоб визначити моделі, потрібно:

• Створити файл service.xml у модулі *-service проекту. Він знаходиться в кореневій папці цього модуля, якщо він ще не існує.

• Визначити глобальну інформацію для сервісів.

• Визначити сутності сервісу

• Визначити атрибути для кожної сутності

• Визначити відношення між всіма сутностями

• Визначити порядок для сутностей, які потрібно витягнути з бази даних

• Визначити методи пошуку для витягування об’єктів з бази даних

Приклад того, як повинен виглядати файл service.xml [8]:

```
<?xml version="1.0"?>
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD Service Builder 7.0.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_7_0_0.dtd">
<service-builder package-path="fr.smile.pet.service">

	<namespace>FOO</namespace>
	<!--<entity data-source="sampleDataSource" local-service="true" name="Foo" remote-
service="false" session-factory="sampleSessionFactory" table="foo" tx-
manager="sampleTransactionManager uuid="true"">-->
	<entity local-service="true" name="Foo" remote-service="true" uuid="true">

	<!-- PK fields -->
	<column name="fooId" primary="true" type="long" />

	<!-- Group instance -->
	<column name="groupId" type="long" />

	<!-- Audit fields -->
	<column name="companyId" type="long" />
	<column name="userId" type="long" />
	<column name="userName" type="String" />
	<column name="createDate" type="Date" />
	<column name="modifiedDate" type="Date" />

	<!-- Other fields -->
	<column name="field1" type="String" />
	<column name="field2" type="boolean" />
	<column name="field3" type="int" />
	<column name="field4" type="Date" />
	<column name="field5" type="String" />

	<!-- Order -->
	<order by="asc">
		<order-column name="field1" />
	</order>
	<!-- Finder methods -->
	<finder name="Field2" return-type="Collection">
		<finder-column name="field2" />
	</finder>

	<!-- References -->
	<reference entity="AssetEntry" package-path="com.liferay.portlet.asset" />
	<reference entity="AssetTag" package-path="com.liferay.portlet.asset" />
	</entity>
</service-builder>
```

1.2.6 Захищеність Liferay

Офіційна документації Liferay Portal стведжує, що Liferay старається притримутись списку OWASP Top 10 (2013) and CWE/SANS Top 25 щоб зробити максимально безпечний портал Liferay. Дотримання цих рекомендацій захищає портал від відомих видів атак і вразливостей безпеки. Наприклад, рівень збереження даних Liferay Portal створюється і підтримується Service Builder, який запобігає SQL-ін'єкції, використовуючи запити Hibernate і параметри. Щоб запобігти перехресним сценаріям атак (XSS), подані користувачем значення **екрануються** на виводі. Для підтримки функцій інтеграції портал Liferay не кодує вхідні дані. Дані зберігаються в оригінальній формі, наданій користувачем. Портал Liferay включає в себе вбудований захист від атак CSRF, відкритих перенаправлень, завантаження та обслуговування файлів небезпечних типів, перехоплення вмісту, перехоплення Clickjacking, Path Traversal та багатьох інших поширених атак.

Також, щоб витримувати конкуренцію Liferay містить виправлення для найсучасніших атак і методики для поліпшення безпеки продукту. Наприклад, Liferay Portal використовує PBKDF2 для зберігання паролів. Портал Liferay також містить функціонал для зменшення наслідків атаки для квадратичної атаки Blowup XXE, уразливості Rosetta Flash, програми Reflected File Download і інших видів атак.

Висновок до розділу 1

Liferay досить зручний спосіб для розробника швидко та ефективно реалізувати деяке бізнес-рішення. Liferay Portal надає досить широкий функціонал можливостей для полегшення роботи розробнику та його фокусуванні на бізнес-логіці. Але в той же час безпека сервісу не завжди є пріоритетною метою для розробника, що в деякій мірі переносить

відповідальність на саму платформу. Очевидною є важливість використання захищеної платформи при побудові корпоративного порталу.

2 АНАЛІЗ LIFERAY ПОРТАЛІВ НА НАЯВНІСТЬ ВРАЗЛИВОСТЕЙ

2.1 Система проведення дослідження

Аналіз потенційних загроз, методів захисту реалізовних самою платформою та наявність документації щодо безпеки будуть основним показником щодо захищеності корпоративного порталу на базі Liferay.

В даному розділі будуть описані три основних підрозділи: тестування, аналіз документації і вихідних кодів та поради програмістам Liferay щодо покращення захисту, чи запобігання деяких видів атак.

За основу для тестування на вразливості було взято Top Ten OWASP – десять найкритичніших загроз безпеці веб-застосунків. Відкритий проект по забезпеченню безпеки веб-додатків (OWASP) - це відкрите співтовариство, яке дозволяє організації розробляти, здобувати і підтримувати безпечні програми та інтерфейси прикладного програмування (API).

Топ-10 OWASP 2017 заснований головним чином на 40+ комплектах даних, отриманих від організацій, які спеціалізуються на безпеці додатків, а також на галузевих дослідженнях, проведених понад 500 незалежними дослідниками. Дані містять інформацію про вразливості, виявлені в сотнях організацій та понад 100.000 реальних додатків і API. На основі даних про поширеність, простоті в експлуатації і складності виявлення вразливостей, а також збитки, яких вони можуть завдати, складається список Топ-10.

Оскільки аналіз всіх 10 виходить за рамки даної дипломної роботи, нами було вибрано такі вразливості із даного списку:

- Ін’єкція [9]
- Недоліки аутентифікації
- Недоліки контролю доступу

Наш вибір був зумовлений декількома факторами. Перш за все це розповсюдженість. Всі наведені вразливості (окрім XSS) входять в ТОП-5 вразливостей по версіїї 2017 року. Також нами було проаналізовано попередні списки OWASP TOP-10 за 2013 рік і були виявлені деякі вразливості, які сильно піднялись в позиціях, або ж навіть не були присутніми в попередніх списках (Недоліки контролю доступу). Ще одним важливим фактором, який зумовив вибір даних вразливостей була сама можливість та складність тестування веб-ресурсу на наявність тієї чи іншої вразливості.

Отож, визначившись з вразливостями, ми будемо проводити тестування веб-ресурсу на базі Liferay на наявність цих вразливостей. Тестування буде відбуватись на трьох порталах: “чистий лайфрей”, тестовий портал з наявністю одного портлета з операціями CRUD та примітивним UI, робочий проект наданий компанією, де проходилась практика. В залежності від самої вразливості, можливості її реалізації та необхідних компонентів, буде вибрано один з вищеперечислених порталів для тестування. Для того, щоб більш ширше і більш загальніше розкрити тему, в даній дипломній роботі буде використовуватись система пріоритизації (пріоритет зліва - направо): “чистий лайфрей” - “чистий лафрей” + CRUD портлет - робочий портал. Тобто, при можливості тестування даної вразливості на “голому Liferay”, відбудеться саме тестування на ньому, якщо ж ні — буде вибраний наступний по пріоритету портал.

В кожному розділі детально буде описуватись процес самого тестування та короткий опис засобів, які використовувались для проведення тестів.

На основі тестів буде проведено дослідження та опис методів захисту реалізованим самим Liferay, які дозволяють уникнути атак на ті, чи інші вразливості. Дослідження і опис будуть взяті із самої документації по безпеці Liferay, а також шляхом аналізу вихідного коду.

В кінці буде запропоновано як захиститись програмним методом з та без використання нативних Liferay засобів. Ця інформація може бути корисна для

розробників Liferay порталів, та для тих, кого цікавить як захистити свій портал від тієї, чи іншої атаки.

2.2 Тестування корпоративних порталів на наявність вразливостей

Як говорилось вище, тестування корпоративних порталів на базі Liferay буде проводитись на наявність таких вразливостей:

- Ін’єкція
- Недоліки аутентифікації
- Недоліки контролю доступу

Відповідно тестування на кожну із вразливостей винесено у відповідний підрозділ. Тестування буде проводитись для порталу по пріоритезації, яка також була описана вище.

2.2.1 Тестування порталу на можливість проведення атак ін’єкцій

Вразливості, пов'язані, наприклад, з ін’єкцією SQL, NoSQL, OS і LDAP, виникають, коли неперевірені дані відправляються інтерпретатора в складі команди або запиту. Шкідливі дані можуть змусити інтерпретатор виконати непередбачувані команди або звернутися до даних без проходження відповідної авторизації.

Перевіримо портал на вразливість до SQL-ін’єкцій. Для цього скористаємось сканером sqlmap.

Програма sqlmap дозволяє перевіряти сайти на наявність в них вразливості SQL-ін'єкцій, уразливості XSS, а також експлуатувати SQL-ін'єкцію. Підтримуються різноманітні типи SQL-ін'єкцій і різноманітні бази даних. За допомогою sqlmap можна перевірити, чи є в сайтах вразливість. Якщо сайт

вразливий до SQL-ін'єкції, то можливо: отримати інформацію з бази даних, в тому числі і дамп бази даних, змінювати і видаляти інформацію з бази даних, заливати шелл (бекдор) на веб-сервер.

Якщо сайт отримує дані від користувача методом GET (коли і ім'я змінної і передані дані видно в адресному рядку браузера), то потрібно вибрати адресу сторінки, в якій присутня ця змінна. Вона йде після знаку питання.

Спробуємо протестувати портал, який являє собою “чистий Лайфрей” і один портал написаний нами з реалізацією CRUD операцій. Портал являє собою список тестових сутностей у вигляді таблиці, при кліку на рядок у нас відбувається доступ до сторінки з деталями цієї сутності. На бек-енд частині у нас є реалізований портлет із Action Methods для кожної операції CRUD, також взаємодія із БД описана в сервісах (див. Додаток А). Також паралельно будемо тестувати робочий портал.

Будемо тестувати URL сторінки деталей, де через GET запит витягується сутність із БД. Отже, тестовий URL: “http://localhost:8080/web/guest/details?p_p_id=fr_pet_test_portlet_BookingPortlet&p_p_lifecycle=0&_fr_pet_test_portlet_BookingPortlet_bookingId=473108” Варто зазначити, що параметри p_p_id та p_p_lifecycle часто добавляє сам Liferay, а також “_fr_pet_test_portlet_BookingPortlet_” - простір імен портлета, який добавляється на початок будь-якого параметра URL.

Команда в sqlmap для тестування змінної GET запиту дуже проста:

```
sqlmap -u адрес_сайту
```

Для автоматизації запиту добавимо додатковий параметр для команди: --batch. В процесі виконання сканування sqlmap задає багато запитань користувачеві, з використанням параметра batch на всі питання буде вибрана відповідь по-замовчуванню.

В результаті ми отримали такі результати:

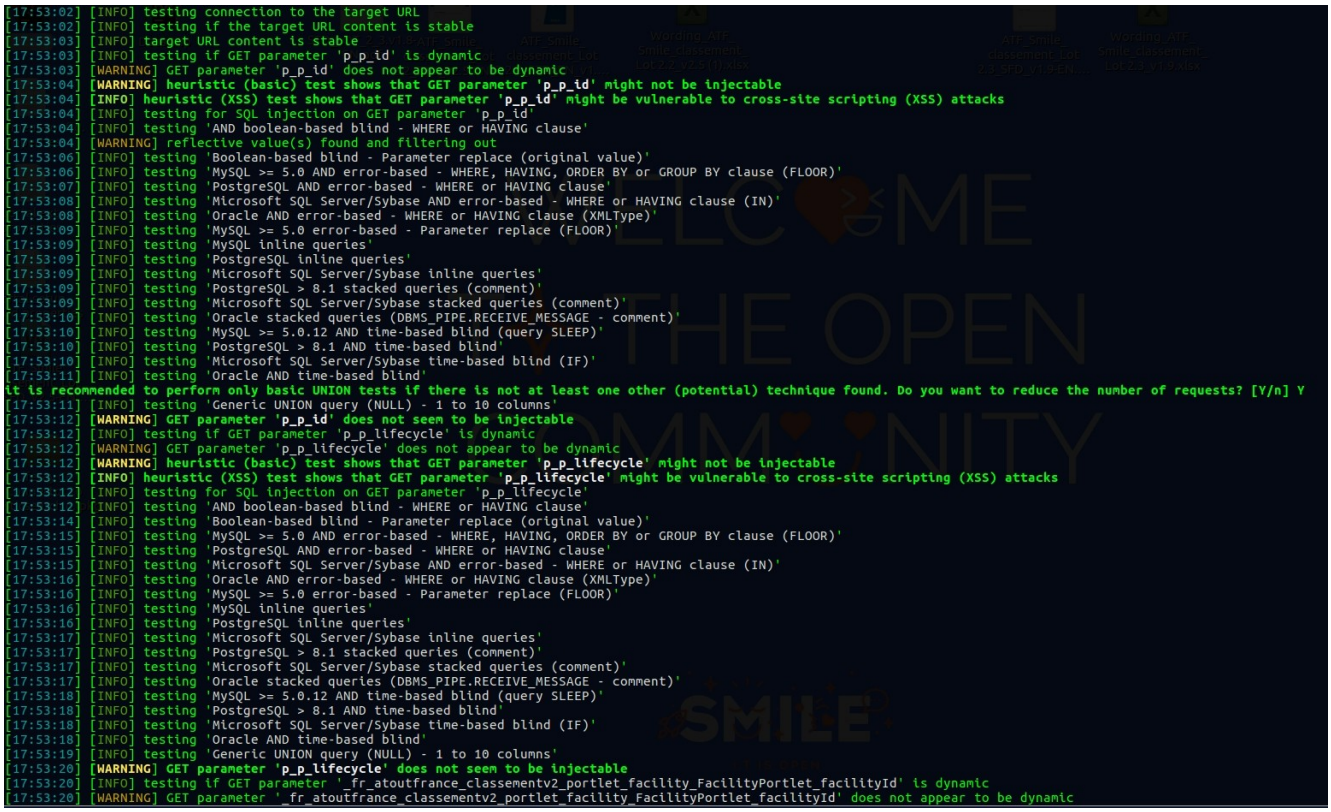

Рисунок 2.1 — Скріншот результатів роботи sqlmap

Отже, в результаті, sqlmap перевірив 3 параметра. Як бачимо, для стандартних параметрів Liferay “p_p_id” та “p_p_lifecycle” ми отрмали такі попередження:

[17:53:03] [WARNING] GET parameter 'p_p_id' does not appear to be dynamic

[17:53:04] [WARNING] heuristic (basic) test shows that GET parameter 'p_p_id' might not be injectable

[17:53:04] [WARNING] reflective value(s) found and filtering out

[17:53:12] [WARNING] GET parameter 'p_p_id' does not seem to be injectable

[17:53:12] [WARNING] GET parameter 'p_p_lifecycle' does not appear to be dynamic

[17:53:12] [WARNING] heuristic (basic) test shows that GET parameter 'p_p_lifecycle' might not be injectable

Як бачимо, heuristic(базовий) тест показав що всі 3 параметри не вразливі до ін’єкційних атак. Також ми не отримали ніякої інформації про вид атаки, чи таблиці бази даних, тільки попередження. Попробуємо зробити те ж саме

тестування з параметром --db, де добавляється аналіз назв баз даних, отримуємо наступні результати:

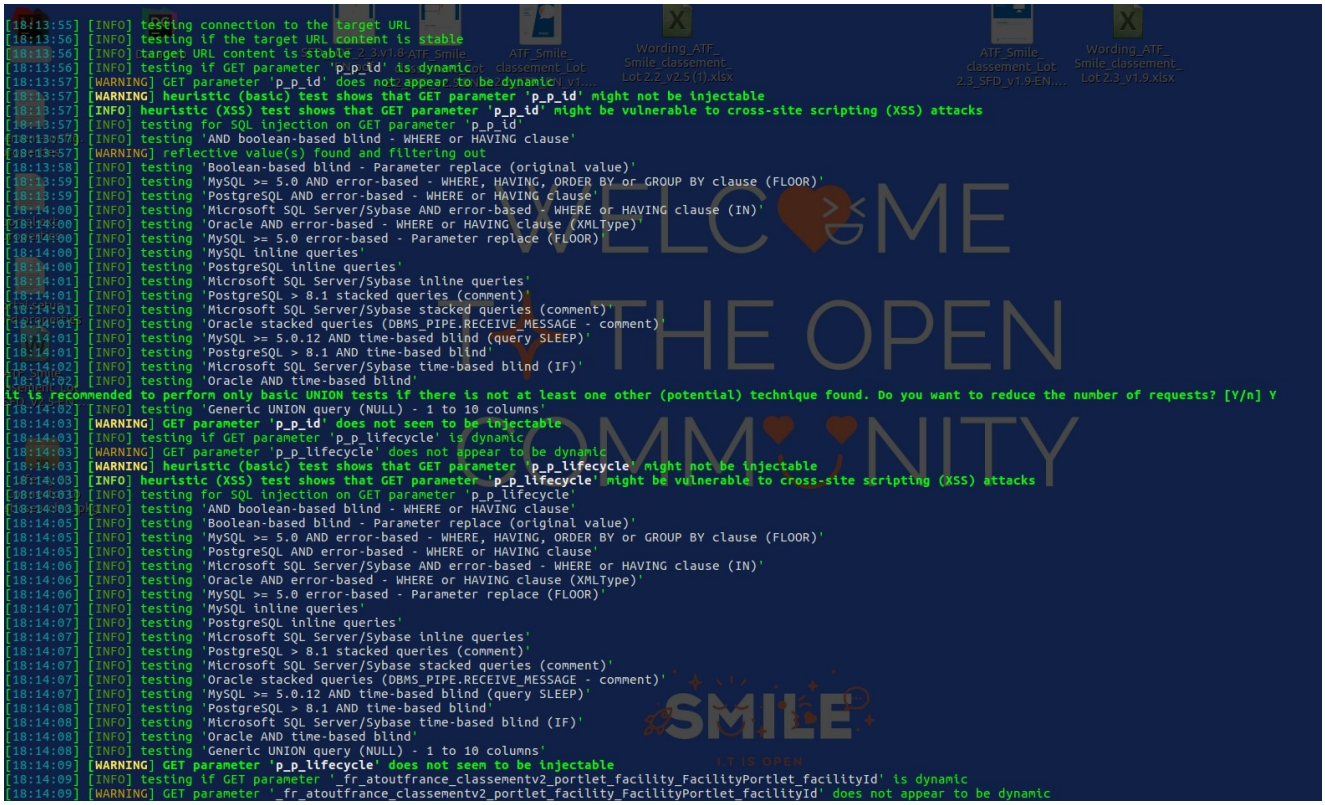

Рисунок 2.2 – Скріншот результатів роботи sqlmap

Як бачимо, нічого нового ми не отримали. Можемо зробити висновок, що даний портал не піддатливий до SQL-ін'єкцій шляхом ін'єкції зловмисного коду в параметр GET запиту. Також, дослідження реалізації, яка по-замовчуванню (без додаткових перевірок програмістом) забезпечує захист від SQL-ін'єкцій буде розглянуто в наступному розділі.

2.2.2 Тестування порталу на наявність недоліків аутентифікації

Функції додатків, пов'язані з аутентифікацією і управлінням сесіями, часто некоректно реалізуються, дозволяючи зловмисникам скомпрометувати паролі, ключі або сесійні токени, а також експлуатувати інші помилки реалізації для тимчасового або постійного перехоплення облікових записів користувачів.

Зловмисники мають доступ до сотень тисяч дійсних комбінацій імен і паролів для атак на облікові записи, списків стандартних облікових даних адміністраторів, інструментів для автоматизації атак методом підбору і атак за словниками.

Підтвердження особистості користувача, аутентифікація і управління сесіями грають важливу роль в захисті від атак, пов'язаних з аутентифікацією. Веб-додаток має недоліки в аутентифікації, якщо:

• допускається проведення автоматизованих атак, наприклад, на облікові записи, коли у атакуючого є список діючих імен і паролів користувачів ;

• допускається проведення атак методом підбору або інших автоматизованих атак;

• допускається використання стандартних, ненадійних або добре відомих паролів, наприклад, "Password1" або "admin / admin";

• використовуються ненадійні або неефективні методи відновлення облікових даних і паролів, наприклад, "відповіді на основі знань", які є небезпечними;

• використовуються незашифровані, зашифровані або ненадійно хешовані паролі;

• відсутня або є неефективною багатофакторна аутентифікація;

• відображаються ідентифікатори сесії в URL (наприклад, перезапис URL);

• не змінюються ідентифікатори сесій після успішного входу в систему;

• некоректно анулюються ідентифікатори сесій. Призначені для користувача сесії або токени аутентифікації (зокрема, токени єдиного входу (SSO)) неправильно анулюються при виході з системи або бездіяльності.

В даному розділі ми проаналізуємо та протестуємо портал на основі “чистого Liferay” без наявності жодних доробок в плані аутентифікації на наявність вищезгаданних недоліків. Оскільки платформа Liferay Portal надає нам цілком завершений та пропрацьований компонент автентифікації, пропонується в

цьому розділі розглянути його окремо, без можливих доробок програмістами у вигляді так званих “хуків” (hooks).

Для тестування Liferay Portal на стійкість щодо автоматизованих атак підбору паролю (brute force) будемо використовувати Hydra CLI. Hydra – це програма для підбирання паролів на основі комбінації файлів із частими чи типовими логінами та паролями користувачів.

Для запуску програми використаємо команду:

```
hydra <HOSTNAME> <PROTOCOL/METHOD> -m
"</PATH/TO/AUTH/FILE>:<USERNAME_FIELD>=^USER^&<PASSWORD_FIELD>=^PASS^:<ERROR_MESSAGE>" -L
listOfUsers.txt -P listOfPasswords.txt -t 10 -w 30 -o <OUTPUT_FILENAME> -s <PORT>
```

Параметри використовуємо наступні:

<HOSTNAME> - оскільки у нас локально запущений сервер із порталом, ім’я хоста буде рівне *localhost* або *127.0.0.1*

<PORT> - 8080

<PROTOCOL/METHOD> - оскільки в нас відправка форми йде через POST запит, даний параметр повинен бути *http-form-post*

<PATH/TO/AUTH/FILE> - в нашому випадку він рівний action параметру зазначеному у формі (див. Рис.)

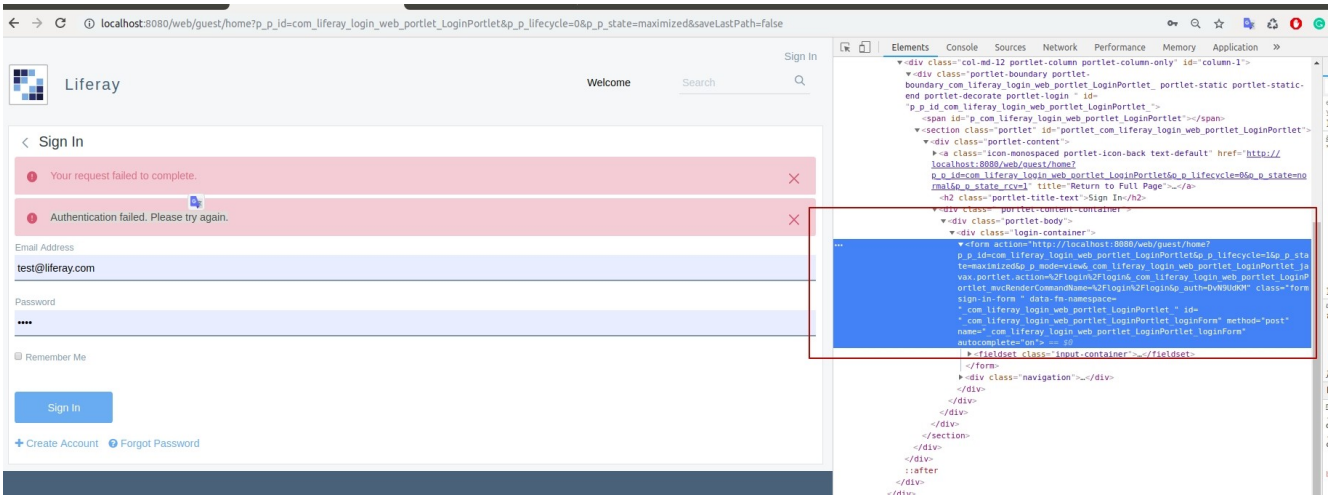

Рисунок 2.3 – Скріншот інспекції коду та витягування потрібних даних

Отож, в результаті ми маємо повністю сформовану команду:

Рисунок 2.4 — Скріншот роботи hydra

Виконаючи її, ми змогли побачити, що в Liferay Portal немає обмеження на кількість спроб аутентифікації з одного й того ж IP. Оскільки було зроблено понад 1000 спроб автентифікації підряд і ніякого обмеження з боку платформи не було. Платформа показала себе погано з точки зору захисту від автоматизованих атак на аутентифікацію. Але, варто зазначити, це стандартний компонент Liferay. Кастомізація і покращення LoginPortlet буде запропонована в наступному розділі.

В будь-якому випадку аутентифікація Liferay Portal являє собою тільки нативний компонет (out of the box), в той же час він надає розробникам дуже широкий та гнучкий функціонал для кастомізації аутентифікації. Основні з них це:

- Auto Login — надає можливість аутентифікації, використовуючи облікові дані в HTTP заголовку з іншої системи
- Authentication Pipelines — використовується у випадку, якщо потрібно перевірити облікові дані в інших системах, а не прямо в базі даних Liferay Portal.
- Custom Login Portlet - якщо потрібно змінити повністю логіку аутентифікації, розробникам пропонується реалізувати свій власний портлет аутентифікації.

2.3 Аналіз методів захисту Liferay

В цьому розділі на основі результатів тестів буде проведено дослідження та опис методів захисту реалізованих самим Liferay, які дозволяють уникнути атак на ті, чи інші вразливості. Дослідження і опис будуть взяті із самої документації по безпеці Liferay, а також шляхом аналізу вихідного коду [6].

2.3.1 Аналіз захисту Liferay від ін’єкцій

В попередньому розділі ми протестували на можливість проведення SQL-ін’єкцій два портали. Одним з них був тестовий портал, де був реалізований один портлет з операціями CRUD для однієї сутності БД. Іншим був робочий Liferay проект наданий компанією, де проходилась практика. Варто зазначити, що основним компонентом, який впливає на захист від SQL-ін’єкцій є правильна обробка даних на моменті запису в БД. Під правильною обробкою даних мається на увазі екранування спеціальних символів, а також використання параметризованих SQL-запитів. Тобто логіка, яка обробляє дані перед записуванням в БД, або її ще називають persistence рівень, повинна передбачати і реалізовувати захист від SQL-ін’єкцій. Оскільки на етапі тестування не було виявлено вразливостей щодо можливості проведення SQL-атак ні на жодному із порталів, можна стверджувати, що persistence шар правильно обробляє параметри, які до нього надходять.

Також варто зауважити, що в persistence шарі в нашому порталі ми не використовували жодного самостійно написаних обробок щодо SQL-ін’єкцій. Був використаний тільки Service Builder як стандартний компонент та API Лайфрея. За допомогою Service Builder було згенеровано всі persistence, service та api модулі. Також в ServiceImpl було добавлено ряд своїх сервіс методів для зміни логіки

роботи з БД (див. Додаток А). Одночасно, інший тестовий портал також використовував в якості persistence шару згенеровані Service Builder’ом класи.

Отже, можна зробити висновок, що весь захист він можливості здійснення SQL-ін’єкцій успішно реалізований Service Builder’ом. Спробуємо дослідити, що ж дає можливість Service Builder’у ефективно справлятись з SQL-ін’єкціями. Оскільки зі сторони програміста потрібно тільки задати конфігурацію service.xml, а решту роботи API виконає за нього. Якщо детально проаналізувати роботу Service Builder, можна зробити наступні висновки.

Service Builder створює поле бази даних для кожного стовпця, який ви додаєте до файлу service.xml. Він встановлює тип поля бази даних, відповідний типу Java, визначеному для кожного стовпця, і може робити це у всіх базах даних, які підтримує Liferay. Після запуску Service Builder він генерує конфігурацію Hibernate, яка забезпечує нам якраз об'єктно-реляційне відображення. Для цих атрибутів службу Builder автоматично генерує методи getter / setter у класі моделі. Ім'я стовпця вказує ім'я, яке використовується в геттерах і сеттерах, створених для поля Java об'єкта. Тип стовпця для сутності вказує тип Java цього поля. Якщо значенням атрибута Primary (тобто первинний ключ) стовпця встановлено як true, то стовпець стає частиною первинного ключа для сутності. Первинний ключ сутності - це унікальний ідентифікатор для сутності. Якщо лише один стовпець має первинне значення, встановлене як істина, то цей стовпець представляє весь первинний ключ. Проте можна використовувати кілька стовпців як первинний ключ для сутності. У цьому випадку комбінація стовпців становить складений первинний ключ.

Тобто, в середині роботи з даними та БД лежить ORM Hibernate. Hibernate полегшує зберігання та витягування об'єктів Java через об'єкт / реляційне відображення (ORM). Hibernate використовує HQL мову запитів для доступу до БД. Приклад використання HQL:

```
Query hqlQuery = session.createQuery("from Orders as orders where orders.id = ?");
List results = hqlQuery.setString(0, "123-ADB-567-QTWYTFDL").list();
```

Наведені вище фрагмент коду використовують параметри для побудови sql запиту. Драйвер JDBC зможе уникнути шкідливих даних перед виконанням запиту, переконавшись, що дані використовуються належним чином.

Припускаючи, що дані були введеними користувачем і які не були перевірені або видалені та містять шкідливий код, шкідливий код буде належним чином вилучено драйвером JDBC (оскільки використовуються параметризовані запити). І введені користувачем дані будуть використовуватися як дані, а не як код.

2.3.2 Аналіз захисту аутентифікації в Liferay

Процес аутентифікації в Liferay Portal - це послідовність дій (pipeline), проходячи які, користувачі можуть бути перевірені однією або кількома системами. Гнучкість і розширюваність Liferay Portal дають можливість зробити аутентифікацією користувачів в будь-якому вигляді, який програміст може втілити, а не обмежуватися тим, що реалізовано самим Liferay.

В цьому розділі ми проаналізуємо аутентифікацію реалізовану самим Liferay, а в наступному — буде запропоновані методи для покращення аутентифікації.

Вся логіка щодо авторизації в Liferay 7.0 у вихідному коді знаходиться в модулі login-web/com/login/web/intrnal . А структура цього модуля виглядає так:

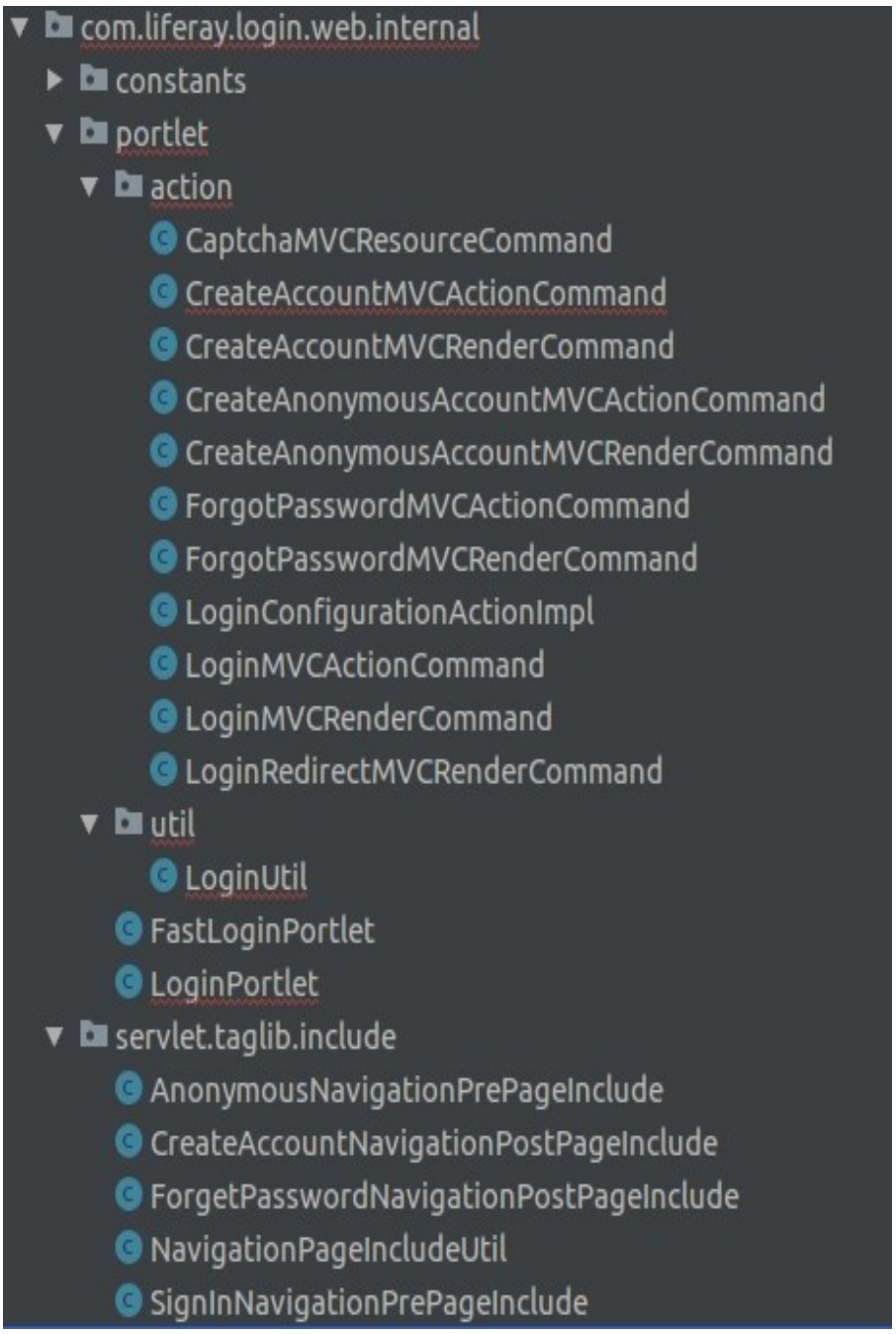

Рисунок 2.5 – Скіншот структури модулю автентифікації

В пакеті constants знаходяться набір класів констант. Пакет portlet містить в собі самий LoginPortlet, а також підпакети action та util, які відповідно описують всі дії (Action, Render та Resource команди) та деякі винесені методи.

В основному портлеті даного модуля LoginPortlet міститься набір наступних налаштувань:

```
@Component(
	immediate = true,
	property = {
	"com.liferay.portlet.add-default-resource=true",
	"com.liferay.portlet.css-class-wrapper=portlet-login",
	"com.liferay.portlet.display-category=category.tools",
	"com.liferay.portlet.icon=/icons/login.png",
	"com.liferay.portlet.preferences-owned-by-group=true",
	"com.liferay.portlet.private-request-attributes=false",
	"com.liferay.portlet.private-session-attributes=false",
	"com.liferay.portlet.render-weight=50",
	"com.liferay.portlet.single-page-application=false",
	"com.liferay.portlet.use-default-template=true",
	"javax.portlet.display-name=Sign In",
	"javax.portlet.expiration-cache=0",
	"javax.portlet.init-param.add-process-action-success-action=false",
	"javax.portlet.init-param.config-template=/configuration.jsp",
	"javax.portlet.init-param.template-path=/",
	"javax.portlet.init-param.view-template=/login.jsp",
	"javax.portlet.name=" + LoginPortletKeys.LOGIN,
	"javax.portlet.resource-bundle=content.Language",
	"javax.portlet.security-role-ref=guest,power-user,user",
	"javax.portlet.supports.mime-type=text/html"
	},
	service = Portlet.class
)
public class LoginPortlet extends MVCPortlet {
}
```

Також в стандартному модулі Liferay 7.0 містяться 5 рендер команд для відображення тієї чи іншої сторінок. LoginMVCRenderCommand – являє собою контроллер для відображення основної сторінки входу. ForgotPasswordMVCRenderCommand – контроллер для відображення сторінки відновлення паролю. CreateAccountMVCRenderCommand – контроллер для відображення сторінки реєстрації.

Кожна з рендер команд являє собою java клас, який реалізує інтерфейс MVCRenderCommand, а також шлях до відповідної JSP, яка відображає даний портлет на стороні користувача.

```
@Component(
        property = {
        "javax.portlet.name=" + LoginPortletKeys.LOGIN,
        "mvc.command.name=/login/login"
        },
        service = MVCRenderCommand.class
)
public class LoginMVCRenderCommand implements MVCRenderCommand {

        @Override
        public String render(
        RenderRequest renderRequest, RenderResponse renderResponse) {

        return "/login.jsp";
        }

}
```

В кожній Action команді першим що робиться, при обробленні запиту, це детальна перевірка на коректність параметрів, які надходять в запиті, наприклад для ForgotPasswordMVCActionCommand ми перевіряємо чи наявні параметри, які надходять з JSP, для коректної роботи обов’язково потрібно наявність одного з двох параметрів, в протилежному випадку викидається помилка, яка обробляється на JSP за допомогою скріплетів:

```
if (!company.isSendPassword() && !company.isSendPasswordResetLink()) {
                throw new PrincipalException.MustBeEnabled(
                        company.getCompanyId(),
                        PropsKeys.COMPANY_SECURITY_SEND_PASSWORD,
                        PropsKeys.COMPANY_SECURITY_SEND_PASSWORD_RESET_LINK);
}
```

Після детальної перевірка в Action командах викликається проміжний шар — шар сервісів, де знову ж таки, він звертається до persistence шару, який в свою чергу працює з Hibernate. Основний сервіс з яким взаємодіють Action команди в модулі авторизації це UserLocalService.

2.3.3 Аналіз контролю доступу в Liferay

Основним способом контролю доступу в Liferay є Liferay Permission. Система Liferay Permission являє собою гнучкий механізм, який визначає дії, які даний користувач може виконати в контексті Liferay або конкретної програми. Розробники Liferay розмежовують операції, які можуть бути виконані в Liferay на різні дії. Процес надання можливості виконувати дії певній ролі користувача називається розподіл прав. У Liferay права (permissions) не присвоюються безпосередньо користувачам. Натомість вони присвоюються надаються ролі. Ролі, у свою чергу, можуть бути призначені для певних користувачів, сайтів, організацій або груп користувачів.

Розробникам в свою чергу необхідно тільки визначити команди, які потрібні, щоб реалізувати бізнес-логіку їх додатків. Вони не повинні турбуватися про те, які користувачі і яким чином отримають права. Після того, як дії були визначені та коректно налаштовані, адміністратори порталу можуть надавати дозволи для виконання цих дій користувачам, сайтам, організаціям або групам користувачів шляхом призначення ролей. Адміністратори можуть використовувати інструменти адміністрування порталу для присвоєння прав первній ролі.

2.4 Поради розробникам щодо захисту інформації в корпоративних порталах на базі Liferay

В даному підрозділі будуть надані практичні поради Liferay розробникам щодо організації захищеного порталу.

2.4.1 Поради Liferay розробникам щодо захисту від ін’єкцій

На рахунок захисту порталів реалізованих з використанням платформи Liferay Portal від SQL-ін’єкцій, можна сказати, що готова Liferay реалізація Service Builder буде найкращим варіантом захисту. І порадою, в даному випадку буде

застосування Service Builder як генератора persistence, service та model шарів порталу. Оскільки згенерований persistence шар взаємодіє напряму із конфігурованою ORM Hibernate. І як пояснювалось в попередніх розділах, це являється надійним і перевіреним варіантом захисту від SQL-ін'єкцій.

Також, варто окремо згадати один варіант, в якому розробник реалізує свої так звані custom SQL. Liferay Custom SQL є можливістю, яку підтримує Service Builder для виконання користувацьких складних запитів до бази даних, викликаючи користувацький SQL запит з методу finder у persistence шарі.

Як бачимо, використання Custom SQL не передбачає собою взаємодії з Hibernate, що може стати причиною виникнення SQL-ін'єкцій. Отже, важливою частиною створення своїх SQL запитів є грамотне їх проектування та взаємодія із іншими частинами persistence шару. Спробуємо надати поради, щодо створення своїх Liferay Custom SQL [3].

Service Builder допомагає створювати інтерфейси для користувацького методу пошуку. Це можна зробити, виконавши такі дії:

1. Визначити свою Custom SQL
2. Реалізувати метод пошуку
3. Отримати правильний доступ із шару сервісів

Після створення SQL запиту та його тестування, потрібно його зберегти у відповідному файлі в проекті, щоб пізніше отримати до нього доступ. Клас CustomSQLUtil (в модулі com.liferay.portal.dao.orm.custom.sql) витягує SQL з файлу default.xml у папці src/main/resources/META-INF/custom-sql/custom-sql. Необхідно створити папку custom-sql і створити файл default.xml. Файл default.xml повинен відповідати наступному формату:

```
<custom-sql>
  <sql id="[повне ім'я назви класу + метод]">
  SQL запит поміщається всередину <![CDATA[...]]>
  </sql>
</custom-sql>
```

Наступним кроком буде реалізування методу пошуку (finder method). Спочатку потрібно створити клас *FinderImpl в пакеті persistence.impl. Клас

повинен реалізовувати інтерфейс BasePersistenceImpl <Entry>. Далі потрібно запустити Service Builder для створення інтерфейсу *Finder на основі класу *FinderImpl.

Тепер потрібно добавити метод пошуку в новому класі. Наприклад, це все може виглядати так:

```
public List<Entry> findByEntry(String entryName, int begin, int end) {
    Session session = null;
    try {
        session = openSession();

        String sql = CustomSQLUtil.get( getClass(), FIND_BY_ENTRYNAME);

        SQLQuery q = session.createSQLQuery(sql);
        q.setCacheable(false);
        q.addEntity("ENTRY", EntryImpl.class);

        QueryPos qPos = QueryPos.getInstance(q);

        qPos.add(entryName);

        return (List<Entry>) QueryUtil.list(q, getDialect(), begin, end);
    }
    catch (Exception e) {
        try {
            throw new SystemException(e);
        }
        catch (SystemException se) {
            se.printStackTrace();
        }
    }
    finally {
        closeSession(session);
    }

    return null;
}

public static final String FIND_BY_ENTRYNAME = EntryFinder.class.getName() +
".nameOfTheMethod";
```

На цьому етапі появляється важливий момент безпеки, який варто не пропустити — це використання параметризованих SQL-запитів. Як можна

побачити, така структура дозволяє нам витягнути самий запит із файла, де оголошується набір файлів за допомогою:

```
CustomSQLUtil.get( getClass(), FIND_BY_ENTRYNAME);
```

І вже маючи його із допомогою QueryPos вставити на потрібні місця відповідні параметри. Не використовуючи параметеризованих запитів, є великий шанс отримати вразливість в порталі щодо SQL-ін’єкцій.

2.4.2 Поради розробникам в реалізації аутентифікації в Liferay

Як було визначено на етапі тестування, портлет аутентифікації не є досконалим. Його можна використовувати в маленьких, без складної бізнес-логіки порталах. Для створення ж маштабного проекту, або для забезпечення суттєвого захисту вашого порталу, пропонується покращення стандартного портлету аутентифікації наступними методами:

- Auto Login — надає можливість аутентифікації, використовуючи облікові дані в HTTP заголовку з іншої системи
- Authentication Pipelines — використовується у випадку, якщо потрібно перевірити облікові дані в інших системах, а не прямо в базі даних Liferay Portal.
- Custom Login Portlet — якщо потрібно змінити повністю або частково логіку аутентифікації, розробникам пропонується реалізувати свій власний портлет аутентифікації.

2.4.3 Поради Liferay розробникам щодо організації контролю доступу

Перше, що потрібно розробнику для коректної роботи із системою Liferay Permissions – це правильне визначення ресурсів і прав. Вони визначаються дуже просто в модулі в модулі resource створюється XML файл default.xml

```
<?xml version="1.0"?>
<!DOCTYPE resource-action-mapping PUBLIC "-//Liferay//DTD Resource Action Mapping
7.0.0//EN" "http://www.liferay.com/dtd/liferay-resource-action-mapping_7_0_0.dtd">

<resource-action-mapping>
   <portlet-resource>
      <portlet-name>com_liferay_booking_web_portlet_BookingPortlet</portlet-name>
      <permissions>
         <supports>
            <action-key>DETAILS_PAGE_ACCESS</action-key>
            <action-key>CONFIGURATION</action-key>
            <action-key>VIEW</action-key>
         </supports>
         <site-member-defaults>
            <action-key>VIEW</action-key>
         </site-member-defaults>
         <guest-defaults>
            <action-key>VIEW</action-key>
         </guest-defaults>
         <guest-unsupported>
            <action-key>DETAILS_PAGE_ACCESS</action-key>
            <action-key>CONFIGURATION</action-key>
         </guest-unsupported>
      </permissions>
   </portlet-resource>
</resource-action-mapping>
```

Всі дії, які підтримуються програмою, визначаються в тезі <supports>, що є підтегом тегу <permissions>. Права за замовчуванням для користувачів порталу визначені в тезі <site-member-defaults>. Те ж саме для гостів даного порталу, всі права задаються в тезі <guest-defaults>. В тезі <guest-unsupported> навпаки задаються права, які заборонені гостям.

Після визначення прав для портлету, потрібно вказати Liferay на файл default.xml, який містить опис прав. У ядрі Liferay є декілька файлів XML для опису різних портлетів Liferay у каталозі portal/portal-impl/src/resource-actions. Файл default.xml у цій папці містить розташування для кожного файлу визначення прав для різних портлетів. В ньому файли потрібно описувати в такій структурі:

```
<resource-action-mapping>
   <resource file="resource-actions/portal.xml" />
   <resource file="resource-actions/announcements.xml" />
   <resource file="resource-actions/asset.xml" />
   <resource file="resource-actions/blogs.xml" />
   ...
</resource-action-mapping>
```

Також, файл з налаштуваннями default.xml потрібно прив’язати до портлету через portlet.properties. В ним ми повинні добавити файл таким чином:

```
resource.actions.configs=resource-actions/default.xml
```

Використання прав зручно в адміністування Liferay, але варто зауважити для чіткого контролю кожної Action, Resource та Render команди ми хотіли б порекомендувати розробникам в своїх портлетах використовувати додаткову логіку для перевірки наявності певних прав. Оскільки, по замовчуванню даної перевірки не відбувається, часткового ефекту можна добитись, використовуючи xml конфігурацію, але для більш складнішої логіки такий варіант не підійде.

Отож, ми пропонуємо розробнику для доступу до будь-якої Action, Resource, Render команди добавляти додаткову перевірку на наявність певних прав. Найлегший та найоптимальніший спосіб реалізації даної перевірки виглядає так:

```
ThemeDisplay themeDisplay =
        (ThemeDisplay) request.getAttribute(WebKeys.THEME_DISPLAY);

String primKey = themeDisplay.getLayout().getPlid() +
LiferayPortletSession.LAYOUT_SEPARATOR +    portletId;

if ( themeDisplay.getPermissionChecker().hasPermission(themeDisplay.getScopeGroupId(),
portletId, primKey, actionId) ) {

        // Дії, які потрібно виконати

}
```

В першому рядку для доступу до додаткових даних, нам потрібно із запиту витягнути об’єкт ThemeDisplay. Він надає нам загальні методи конфігурації для порталу, які надають сам доступ до сторінок порталу, сайтів, тем, локалей, URL-адрес тощо. Цей клас є інформаційним контекстним об'єктом, який містить дані, які зазвичай відносяться до різних front-end видів інформації. Він дозволить сформувати унікальний ключ портлету для коректної перевірки на наявність прав:

```
String primKey = themeDisplay.getLayout().getPlid() +
        LiferayPortletSession.LAYOUT_SEPARATOR +    portletId;
```

Наступним рядком, із об’єкта ThemeDisplay ми витягуємо PermissionChecker для перевірки наявності даного права для даного портлету в БД. Метод hasPermission вимагає декілька параметрів: унікальний ідентифікатор групи, ідентифікатор портлету, ключ для пошуку портлету в БД та власне самий пермішин actionId, який задається в тезі <action-id> в файлі конфігурації default.xml.

Відповідно, результатом роботи методу hasPermission() буде булеве значення true чи false відповідно знайдений даний пермішн для конкретного користувача та конкретного портлету.

Висновок до розділу 2

В даному розділі було розглянуто три основних теми: тестування, аналіз документації і вихідних кодів та поради програмістам Liferay щодо покращення захисту, чи запобігання деяких видів атак.

Було проведено тестування веб-ресурсів на базі Liferay на наявність вразливостей sql ін’єкцій та недоліків автентифікаціє. Тестування відбувалось на трьох порталах: “чистий лайфрей”, тестовий портал з наявністю одного портлета з операціями CRUD та примітивним UI, робочий проект наданий компанією, де проходилась практика. В результаті тестування не було виявлено вразливостей порталів до sql ін’єкцій. Було знайдено недоліки базової версії автентифікації (набір можливостей, які надає сам Liferay), але це зумовлено саме базовим набором, проблема вирішується шляхом гнучких налаштувань порталу.

Також було проведено аналіз методів захисту порталу реалізованих самим Liferay. Було досліджено модуль автентифікації, а саме його структуру та набір базових портлетів і команд. Було досліджено механізми, які дозволяють захистити портал від sql ін’єкцій та механізми забезпечення керування доступом.

На основі результатів досліджень та знань було запропоновані методи покращення безпеки для розробників порталів на Liferay.

ВИСНОВКИ

Результатом даної роботи є систематизовані дані щодо безпки Liferay та зформовані поради для розробки безпечних корпоративних порталів на базі Liferay. Також було протестовано корпоративні портали на наявність вразливостей для можливості проведення SQL-ін’єкцій та проаналізована реалізація аутентифікації Liferay.

Дана дипломна робота розглянула три основних підрозділи: тестування, аналіз документації та вихідного коду та поради програмістам щодо написання захищених порталів. За основу для дослідження було взяті деякі вразливості із списку Top Ten OWASP 2017 : ін’єкція, недоліки аутентифікації, недоліки контролю доступу.

На основі цих вразливостей проводилось тестування різних корпоративних порталів на базі Liferay. В результаті тестування було виявлено хорошу захищеність порталу від SQL-ін’єкцій, а також недосконаліть аутентифікації Liferay.

В наступному розділі було проаналізовані технічні рішення, які дали змогу захисту базового Liferay порталу від SQL-ін’єкцій. В цьому ж таки розділі розглянулась реалізація аутентифікації а також наявність можливості та технологій в Liferay для забезпечення контролю доступу.

В останньому розділі були сформовані практичні поради для Liferay розробників щодо написання захищених корпоративних порталів.

Дані тестування та аналізів свідчать про те, що Liferay є добре захищеною платформою для розробки своїх корпоративних рішень. В процесі були знайдені та виявлені деякі недоліки в плані захисту інформації в базовій реалізації Liferay і надані практичні поради розробникам для боротьби з ними

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Richard Sezov, Jr. . Liferay in action. [Текст] — Manning Publications Co. — 2012. — с.30.
2. Ashish Sarin. Portlets in Action. [Текст] — Manning Publications Co. — 2011. — с.38.
3. Jonas X. Yuan. Liferay Portal 6 Enterprise Intranets. [Текст] – Packt Publishing Ltd – 2010. – с. 74.
4. Samir Bhatt. Liferay Portal Performance Best Practices. [Текст] – Packt Publishing Ltd – 2013. – с. 122.
5. Robert Chen. Liferay Beginner's Guide. [Текст] – Packt Publishing Ltd – 2011. – с. 325.
6. Liferay Portal on GitHub [Електронний ресурс] — Режим доступу до ресурсу: https://github.com/liferay/liferay-portal
7. Liferay Portal Development information [Електронний ресурс] – Режим доступу до ресурсу: https://portal.liferay.dev/docs/7-0/
8. OSGi Declarative Services (DS) Annotations [Електронний ресурс] – Режим доступу до ресурсу: http://www.liferaysavvy.com/2017/10/osgi-declarative-services-ds-annotations.html
9. SQL Injection and how to prevent it? Hibernet/JPA/SQL [Електронний ресурс] – Режим доступу до ресурсу: http://zakirrizvi.blogspot.com/2016/02/sql-injection-and-how-to-prevent-it.html

ДОДАТКИ

Додаток А

```
@Component(
	immediate = true,
	property = {
		"com.liferay.portlet.display-category=category.social",
		"com.liferay.portlet.instanceable=false",
		"com.liferay.portlet.scopeable=true",
		"javax.portlet.display-name=Booking",
		"javax.portlet.expiration-cache=0",
		"javax.portlet.init-param.template-path=/",
		"javax.portlet.init-param.view-template=/view.jsp",
		"javax.portlet.name=" + BookingPortletKeys.Booking,
		"javax.portlet.resource-bundle=content.Language",
		"javax.portlet.security-role-ref=power-user,user",
		"javax.portlet.supports.mime-type=text/html"
	},
	service = Portlet.class
)
public class BookingPortlet extends MVCPortlet {

	public void addBooking(ActionRequest request, ActionResponse response) throws
PortalException {
		ServiceContext serviceContext = ServiceContextFactory.getInstance(
				Booking.class.getName(), request);

		DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");

		String description = ParamUtil.getString(request, "description");
		String place = ParamUtil.getString(request, "place");
		Date bookingDate = ParamUtil.getDate(request, "bookingDate", dateFormat);

		System.out.println(bookingDate.toString());
		long bookingId = ParamUtil.getLong(request, "bookingId");

		if (bookingId > 0) {

			try {

				_bookingLocalService.updateBooking(
						serviceContext.getUserId(), bookingId, bookingDate,
description,
						place, serviceContext);

				SessionMessages.add(request, "bookingAdded");

				response.setRenderParameter(
						"bookingId", Long.toString(bookingId));
```

```
                }
                catch (Exception e) {
                        System.out.println(e);

                        SessionErrors.add(request, e.getClass().getName());

                        PortalUtil.copyRequestParameters(request, response);

                        response.setRenderParameter(
                                        "mvcPath", "/edit_entry.jsp");
                }

        }
        else {

                try {
                        _bookingLocalService.addBooking(
                                        serviceContext.getUserId(), description, bookingDate,
place,
                                        serviceContext);

                        SessionMessages.add(request, "bookingAdded");

                        response.setRenderParameter(
                                        "bookingId", Long.toString(bookingId));

                }
                catch (Exception e) {
                        SessionErrors.add(request, e.getClass().getName());

                        PortalUtil.copyRequestParameters(request, response);

                        response.setRenderParameter(
                                        "mvcPath", "/edit_entry.jsp");
                }
        }

    }

    public void addNote(ActionRequest request, ActionResponse response) throws PortalException
{
        ServiceContext serviceContext = ServiceContextFactory.getInstance(
                        Note.class.getName(), request);

        String text = ParamUtil.getString(request, "text");
        long bookingId = ParamUtil.getLong(request, "bookingId");
        long noteId = ParamUtil.getLong(request, "noteId");

        if (noteId > 0) {
```

```
                try {

                        _noteLocalService.updateNote(noteId, text, serviceContext);

                        SessionMessages.add(request, "noteAdded");
                        Map<String, String[]> params = new HashMap();
                        params.put("noteId", new String[]{Long.toString(noteId)});
                        params.put("bookingId", new String[]{Long.toString(bookingId)});

                        response.setRenderParameters(params);

                }
                catch (Exception e) {
                        e.printStackTrace();

                        SessionErrors.add(request, e.getClass().getName());

                        PortalUtil.copyRequestParameters(request, response);

                        response.setRenderParameter(
                                        "mvcPath", "/edit_notes.jsp");
                }

        }
        else {

                try {

                        _noteLocalService.addNote(
                                        serviceContext.getUserId(), bookingId, text,
                                        serviceContext);
                        SessionMessages.add(request, "noteAdded");

                        response.setRenderParameter(
                                        "bookingId", Long.toString(bookingId));

                }
                catch (Exception e) {
                        SessionErrors.add(request, e.getClass().getName());

                        PortalUtil.copyRequestParameters(request, response);

                        response.setRenderParameter(
                                        "mvcPath", "/edit_notes.jsp");
                }
        }

}

        public void deleteBooking(ActionRequest request, ActionResponse response) throws
PortalException {
```

```
		long bookingId = ParamUtil.getLong(request, "bookingId");

		ServiceContext serviceContext = ServiceContextFactory.getInstance(
				Booking.class.getName(), request);

		try {

			response.setRenderParameter(
					"bookingId", Long.toString(bookingId));

			_bookingLocalService.deleteBooking(bookingId, serviceContext);
		}

		catch (Exception e) {
			Logger.getLogger(BookingPortlet.class.getName()).log(
					Level.SEVERE, null, e);
		}
	}

	public void deleteNote(ActionRequest request, ActionResponse response) throws
PortalException {
		long noteId = ParamUtil.getLong(request, "noteId");
		long bookingId = ParamUtil.getLong(request, "bookingId");

		ServiceContext serviceContext = ServiceContextFactory.getInstance(
				Note.class.getName(), request);

		try {

			response.setRenderParameter(
					"noteId", Long.toString(noteId));

			Map<String, String[]> params = new HashMap<>();
			params.put("mvcPath", new String[] {"/details.jsp"});
			params.put("bookingId", new String[] {Long.toString(bookingId)});

			response.setRenderParameters(params);

			_noteLocalService.deleteNote(noteId);

		}

		catch (Exception e) {
			Logger.getLogger(BookingPortlet.class.getName()).log(
					Level.SEVERE, null, e);
		}
	}


	public void subscribe(ActionRequest request, ActionResponse response) throws
PortalException {
```

```
		ServiceContext serviceContext =
ServiceContextFactory.getInstance(User.class.getName(), request);

		ThemeDisplay themeDisplay = (ThemeDisplay)
request.getAttribute(WebKeys.THEME_DISPLAY);

		User user = themeDisplay.getRealUser();

		ExpandoBridge bridge = user.getExpandoBridge();

		if(bridge.getAttribute("Subscribed") == null) {
			bridge.addAttribute("Subscribed");
			bridge.setAttribute("Subscribed", false);
		}
		bridge.setAttribute("Subscribed", !(boolean)bridge.getAttribute("Subscribed"));
		user.setExpandoBridgeAttributes(bridge);

		UserLocalServiceUtil.updateUser(user);

	}


	@Override
	public void render(RenderRequest renderRequest, RenderResponse renderResponse)
			throws IOException, PortletException {

		super.render(renderRequest, renderResponse);
	}


	@Reference(unbind = "-")
	protected void setBookingService(BookingLocalService bookingLocalService) {
		_bookingLocalService = bookingLocalService;
	}

	@Reference(unbind = "-")
	protected void setNoteService(NoteLocalService noteLocalService) {
		_noteLocalService = noteLocalService;
	}

	private BookingLocalService _bookingLocalService;
	private NoteLocalService _noteLocalService;

}
10.
11.	SERVICE:
12.
public class BookingLocalServiceImpl extends BookingLocalServiceBaseImpl {


	@Indexable(type = IndexableType.REINDEX)
```

```
	public Booking addBooking(
			long userId, String description, Date bookingDate, String place, ServiceContext
serviceContext)
			throws PortalException {

		long groupId = serviceContext.getScopeGroupId();

		User user = userLocalService.getUserById(userId);

		Date now = new Date();

		long bookingId = counterLocalService.increment();

		Booking booking = bookingPersistence.create(bookingId);

		booking.setUuid(serviceContext.getUuid());
		booking.setUserId(userId);
		booking.setGroupId(groupId);
		booking.setCompanyId(user.getCompanyId());
		booking.setUserName(user.getFullName());
		booking.setCreateDate(serviceContext.getCreateDate(now));
		booking.setModifiedDate(serviceContext.getModifiedDate(now));
		booking.setBookingDate(bookingDate);
		booking.setDescription(description);
		booking.setPlace(place);
		booking.setExpandoBridgeAttributes(serviceContext);

		bookingPersistence.update(booking);

		return booking;

	}

	@Indexable(type = IndexableType.REINDEX)
```

```
public Booking updateBooking (
        long userId, long bookingId, Date bookingDate, String description,
        String place, ServiceContext serviceContext)
        throws PortalException, SystemException {

    Booking booking = getBooking(bookingId);

    Date now = new Date();

    User user = userLocalService.getUserById(userId);

    booking.setUserId(userId);
    booking.setUserName(user.getFullName());
    booking.setModifiedDate(serviceContext.getModifiedDate(now));
    booking.setPlace(place);
    booking.setDescription(description);
    booking.setBookingDate(bookingDate);
    booking.setExpandoBridgeAttributes(serviceContext);

    bookingPersistence.update(booking);

    return booking;
}

@Indexable(type = IndexableType.DELETE)
public Booking deleteBooking (long bookingId, ServiceContext serviceContext)
        throws PortalException {

    Booking booking;
    noteLocalService.deleteBookingNotes(bookingId, serviceContext);

    booking = deleteBooking(bookingId);
```

```
            return booking;
        }

        public List<Booking> getBookings(long groupId) {

            return bookingPersistence.findByGroupId(groupId);
        }

        public List<Booking> getBookings (long groupId, int start, int end,

OrderByComparator<Booking> obc) {

            return bookingPersistence.findByGroupId(groupId, start, end, obc);
        }

        public List<Booking> getBookings (long groupId, int start, int end) {

            return bookingPersistence.findByGroupId(groupId, start, end);
        }

        public int getBookingsCount (long groupId) {

            return bookingPersistence.countByGroupId(groupId);
        }

}
```