Міністерство освіти і науки України
Сумський державний університет

Любчак В. О., Савостян В. В., Козолуп П. Д.

БЕЗПЕКА JAVA-ДОДАТКІВ

Навчальний посібник

Суми
Сумський державний університет
2023

Міністерство освіти і науки України
Сумський державний університет

Любчак В. О., Савостян В. В., Козолуп П. Д.

БЕЗПЕКА JAVA-ДОДАТКІВ

Навчальний посібник

Рекомендовано вченою радою Сумського державного університету

Суми
Сумський державний університет
2023

УДК 004.056:004.432(075.8)
Л 93

Авторський колектив:
В. О. Любчак – кандидат фізико-математичних наук, доцент, завідувач кафедри кібербезпеки;
В. В. Савостян – бакалавр спеціальності «Кібербезпека»;
П. Д. Козолуп – старший інженер програміст ТОВ «НЕТКРЕКЕР».

Рецензенти:
Н. Л. Барченко – кандидат технічних наук, доцент кафедри комп'ютерних наук, Сумський державний університет;
О. В. Чалий – начальник відділу розробки та підтримки програмних продуктів ТОВ «НЕТКРЕКЕР» (м. Суми)

Рекомендовано до видання
Вченою радою Сумського державного університету
як навчальний посібник
(протокол № 4 від 9 листопада 2023 року)

Любчак В. О.
Л 93 Безпека Java-додатків : навчальний посібник / В. О. Любчак, В. В. Савостян, П. Д. Козолуп. – Суми : Сумський державний університет, 2023. – 72 с.

Навчальний посібник спрямований надати студентам чіткого і короткого огляду основних понять про безпечне програмування та технології організації належного рівня захисту додатків мовою Java, а також на підготовку студентів до роботи з додатками на основі мови програмування Java з урахуванням аспектів безпеки і вразливостей.

Призначений для студентів освітньо-професійної програми підготовки бакалавра галузі знань 12 «Інформаційні технології» спеціальностей 121 «Інженерія програмного забезпечення», 125 «Кібербезпека та захист інформації»

УДК 004.056:004.432(075.8)

ЗМІСТ

С.

ВСТУП

У сучасному цифровому світі, де програмні додатки займають центральне місце в нашому повсякденному житті, гарантування їх безпеки стає надзвичайно важливим завданням. Зростаюча кількість загроз та атак, спрямованих на вразливості програмного забезпечення, стає загальносвітовою тенденцією та вимагає від нас постійного аналізу й удосконалення механізмів захисту.

Однією з найпоширеніших мов програмування, які використовують для розроблення додатків, є Java. Ця мова має широкий спектр застосувань, починаючи від мобільних додатків до вебсерверів та великих корпоративних систем. Зважаючи на це, виникає необхідність вивчення та аналізу практичних і теоретичних аспектів організації захисту додатків, написаних мовою Java, у контексті кібербезпеки.

Метою цього посібника є проведення аналізу і вивчення вразливостей, які можуть виникати в додатках, написаних мовою Java, та пошук ефективних стратегій і механізмів захисту.

Матеріали посібника сприятимуть розумінню проблем безпеки додатків Java, а також допоможуть розробникам та аудиторам програмного забезпечення покращити рівень безпеки своїх додатків.

Організація захисту додатків Java та кібербезпека загалом ґрунтуються на декількох основних концепціях і поняттях, які важливо розглянути, а саме:

1. Аутентифікація: перевірка ідентичності користувача або системи.

2. Авторизація: визначення прав доступу та обмежень після аутентифікації.

3. Шифрування: перетворення даних у незрозумілу форму для забезпечення конфіденційності.

4. Хешування: перетворення даних у фіксований хеш-код для перевірки цілісності.

5. Уразливості безпеки: слабкі місця або помилки, які можуть бути використані зловмисниками для атак на додатки.

Ці концепції є важливими для розроблення і застосування ефективних стратегій та механізмів захисту додатків Java з метою забезпечення їхньої безпеки та захисту від потенційних загроз.

1. ОСНОВИ БЕЗПЕЧНОГО ПРОГРАМУВАННЯ

1.1. Основні поняття та принципи безпечного програмування

Безпечне програмування – це підхід до розроблення програмного забезпечення, спрямований на гарантування безпеки та запобігання вразливостям, що можуть бути використані для атак або несанкціонованого доступу до системи. Основними цілями безпечного програмування є запобігання вразливостям, зменшення ризиків для даних і забезпечення цілісності та конфіденційності програмного забезпечення.

Основні принципи безпечного програмування передбачають таке:

1. **Валідація та перевірка даних.** Розробники повинні ретельно перевіряти вхідні дані, щоб упевнитися, що вони відповідають очікуваному формату та допустимим значенням. Це допомагає запобігати вразливостям, таким як переповнення буфера або ін'єкції коду.

2. **Керування пам'яттю.** Некоректне керування пам'яттю може призвести до вразливостей, таких як переповнення буфера або використання недійсних покажчиків. Розробники повинні бути уважними в разі виділення, використання та звільнення пам'яті, а також використовувати безпечні механізми, які дозволяють уникнути таких проблем.

3. **Безпечні стандарти кодування.** Використання безпечних стандартів кодування є важливим аспектом безпечного програмування. Це передбачає дотримання правил безпеки, таких як використання безпечних функцій для оброблення рядків, шифрування та захисту даних, а також уникнення вразливих практик, таких як використання

даних користувача без перевірки та некоректне керування даними бази даних.

4. **Автентифікація та авторизація.** Забезпечення безпеки в системі передбачає механізми автентифікації та авторизації. Дотримання правильних методів і протоколів для автентифікації та авторизації допомагає уникнути несанкціонованого доступу до системи та зберегти конфіденційність даних.

5. **Коректне оброблення помилок.** Оброблення помилок є важливим аспектом безпечного програмування. Розробники повинні використовувати безпечні механізми для оброблення помилок, щоб уникнути витоку інформації або зловмисного використання системи.

6. **Захист від уразливостей.** Розробники повинні знати про різні види вразливостей, такі як XSS (кроссайтовий скриптинг), SQL-ін'єкції, CSRF (міжсайтове подання запиту) та багато інших. Вони повинні використовувати належні захисні механізми, такі як валідація даних, параметризовані запити та захист від міжсайтових атак, щоб уникнути цих уразливостей.

Безпечне програмування є важливою складовою кібербезпеки та зосереджується на розробленні програмного забезпечення з метою запобігання вразливостям й атакам. Основи безпечного програмування охоплюють мобільні пристрої, сервери та вбудовані програми.

Основна мета безпечного програмування полягає в усуненні потенційних уразливостей у програмному коді, зменшенні ризиків для даних та забезпеченні цілісності та конфіденційності програмного забезпечення. Існує багато технік і методик, які можуть бути використані для захисту коду та бізнес-даних.

Розглянемо декілька з таких технік:

1. **Застосування обфускації коду.** Де це можливо, захищайте свій код за допомогою технік, таких як зведення коду до мінімуму та обфускація коду.

2. **Уникнення «Cutting Corners».** Не спокушайтеся вибирати скорочені шляхи. У розробників тісні строки, але важливо поставляти код, повністю перевірений та готовий до виробництва, навіть якщо це затримує процес.

3. **Проведення огляду коду.** Огляд коду співробітниками є важливим на великих проєктах, дозволяючи розробникам обмінюватися ідеями. Крім того, це дає можливість іншим експертам критикувати код.

4. **Створення культури безпеки.** Зміна культури – це дуже складний процес, який потребує часу для впровадження в компанії. Перші кроки в напрямку формування команди, яка підтримує пріоритет безпеки, є критичними для успіху.

5. **Документування стандартів.** Стандарти безпечного програмування повинні бути документовані та поширені в приватному репозиторії. Запис правил дає розробнику можливість їх переглянути і сприяє зміні культури.

6. **Перевірка зовнішніх джерел даних.** Іноді має сенс використовувати модулі та код, які вже написані. Перевірте легітимність джерел, перевіряйте завантаження з аутентифікацією SHA та переконайтеся, що будь-які одержані дані зашифровані та дійсні.

7. **Використання моделі загроз.** Модель загроз вводить багатоетапний процес, який аналізує код на вразливості впродовж процесу розроблення програмного забезпечення.

8. **Використання автоматизованих інструментів у складі CI/CD.** Ефективне забезпечення виконання стандартів безпеки є дуже складним завданням, розгляньте

можливість інвестиції в автоматизовані інструменти, такі як Check Point CloudGuard Spectral, які виконають усю важку роботу за вас [1].

Різноманітні аспекти безпечного програмування описані та висвітлені в багаточисельних навчальних посібниках, наукових публікаціях, визнаних методиках та стандартах, інтернет-ресурсах.

Надаємо перелік деяких із цих ресурсів, які будуть корисними для вивчення безпечного програмування та актуальних практик у сфері кібербезпеки.

1. Книги та публікації

- Посібник «Захист програмного забезпечення» [2] містить теоретичні відомості щодо методів захисту програмного забезпечення від несанкціонованого дослідження – як від статичного, так і від динамічного. Детально розглянуто такі методи захисту, як обфускація програмного коду, ускладнення логіки, емуляція, антидампінгові прийоми, маніпулювання заголовками виконуваних файлів, а також методи захисту від інтерактивних та автоматичних дизасемблерів, від налагоджувачів реального та захищеного режимів.
- У праці «Безпека програмного забезпечення та безпечне програмування» [3] розглянуто сучасний стан галузі безпеки програмного забезпечення та безпечного програмування. Наведено огляд стандартів безпечного програмування та засобів автоматизованого контролю безпеки коду.
- У нагоді буде конспект лекцій «Безпека програм та даних» [4], де наведено теоретичні відомості щодо методів захисту програмного забезпечення. Наведено характеристику сучасних систем захисту програмних продуктів та різних засобів, що застосовуються для зламу існуючих систем захисту програмного забезпечення та

автоматизованих систем. Розглянуто моделі поширення програмного забезпечення.

- Підручник «Secure Coding in C and C++» від Robert C. Seacord [5] надає вичерпну інформацію про безпеку програмування мовами C і C++. Охоплює широкий спектр тем, ураховуючи керування пам'яттю, виконання коду та захист від уразливостей.
- Підручник «The Web Application Hacker's Handbook» від Dafydd Stuttard та Marcus Pinto [6] фокусується на вразливостях вебдодатків та методах їх використання зловмисниками. Також дає детальний огляд різних видів атак та практичні рекомендації щодо їх запобігання.
- У посібнику «The Art of Software Security Assessment» від Mark Dowd, John McDonald та Justin Schuh [7] розглядаються методи оцінювання безпеки програмного забезпечення та надає практичні поради щодо виявлення та усунення вразливостей.

2. Організації із стандартизації:

- Організація Open Web Application Security Project (OWASP) [8] є незалежною організацією, яка зосереджується на безпеці програмного забезпечення. Вони надають велику кількість матеріалів, зокрема перелік найбільш поширених вразливостей та рекомендації щодо їх запобігання.
- Організація National Institute of Standards and Technology (NIST) [9] публікує рекомендації та стандарти щодо безпеки програмного забезпечення, такі як NIST SP 800-64 і NIST SP 800-53.

3. Вебсайти та блоги:

- Вебсайт SecurityWeek [10] надає останні новини, статті та аналітичні матеріали про кібербезпеку й безпеку програмного забезпечення.

• Вебсайт Security Boulevard [11] публікує регулярні статті та блоги на тему безпеки програмного забезпечення, включаючи безпечне програмування та найновіші тенденції у цій галузі.

Варто зазначити, що безпека програмного забезпечення є постійно змінюваною галуззю, тому важливо бути в курсі останніх тенденцій і нововведень у цій галузі.

1.2. Загальні вразливості та принципи їх нейтралізації в процесі написання коду

У контексті безпеки програмування розглядаються загальні вразливості, які можуть бути наявними в програмному коді, а також принципи та методи, які допомагають нейтралізувати ці вразливості та запобігти можливим атакам. Безпечне програмування передбачає використання кращих практик, стандартів та методологій, спрямованих на забезпечення надійності, конфіденційності та доступності програмного забезпечення.

Ось декілька загальних уразливостей та принципів їх нейтралізації:

1. Уразливість переповнення буфера:

• Використовуйте безпечні функції для копіювання та конкатенації рядків (наприклад, strncpy, strlcpy, strncat) з обмеженням розміру буфера.

• Перевіряйте розмір вхідних даних перед копіюванням до буфера, щоб уникнути переповнення.

2. SQL-ін'єкції:

• Використовуйте параметризовані запити або підготовлені оператори замість вставлення вхідних даних безпосередньо в SQL-запит.

• Здійснюйте належну валідацію та очищення вхідних даних перед використанням їх у SQL-запитах.

3. XSS (кроссайтовий скриптинг):

- Екрануйте спеціальні символи HTML перед виведенням користувацьких даних на вебсторінці.
- Використовуйте безпечні методи для вставлення даних у DOM (наприклад, textContent замість innerHTML).

4. CSRF (міжсайтове подання запиту):

- Використовуйте токени захисту від CSRF і вимагайте їх перевірки під час оброблення критичних запитів.
- Установлюйте правильні заголовки SameSite та Csrf-Token для запобігання міжсайтовому поданню запитів.

5. Недостатня автентифікація та авторизація:

- Вимагайте сильних паролів та використовуйте безпечні методи хешування для зберігання паролів.
- Перевіряйте права доступу користувачів перед виконанням критичних операцій.
- Використовуйте принцип найменшого спеціального доступу (principle of least privilege) для обмеження привілеїв користувачів.

6. Небезпечне керування пам'яттю:

- Використання автоматичного керування пам'яттю, як у мовах програмування Java або C#, для уникнення проблем, таких як витоки пам'яті або небезпечний доступ до пам'яті.
- Правильне виділення та звільнення пам'яті, щоб уникнути переповнення або доступу до недійсних об'єктів.

7. Небезпека виконання коду з віддалених джерел:

- Використання механізмів відокремленого виконання коду (sandboxing) для ізоляції небезпечних джерел або використання машин віртуалізації.
- Перевірка джерела та достовірності виконуваного коду перед його виконанням.

- Використання механізмів білого списку (whitelisting), які дозволяють виконувати лише певний, надійний код.

8. Уразливості вебдодатків:

- Валідація та екранування вхідних даних, що надходять від користувачів, перед їх обробленням або зберіганням.
- Використання безпечних механізмів автентифікації та авторизації, таких як використання хешів паролів та ролей користувачів.
- Захист від атак, таких як міжсайтовий скриптинг (XSS) і міжсайтова підробка запитів (CSRF), за допомогою валідації даних та встановленням безпечних політик безпеки.

9. Уразливості мережевого зв'язку:

- Використання шифрування та безпечних протоколів зв'язку, таких як HTTPS, для захисту передавання даних.
- Використання механізмів аутентифікації та авторизації в разі з'єднання з мережевими ресурсами.
- Фільтрація та валідація даних, що надходять із мережі, для уникнення атак на мережевий рівень.

10. Недостатнє керування сесіями:

- Використання безпечних механізмів керування сесіями, таких як генерація сильних ідентифікаторів сесій, обмеження терміну дії сесій та видалення старих сесій.
- Застосування механізмів автентифікації та авторизації для кожної сесії, щоб запобігти несанкціонованому доступу до ресурсів.

11. Використання старих або вразливих компонентів:

- Регулярне оновлення і патчення всіх використовуваних компонентів, таких як бібліотеки, фреймворки та залежності.

- Моніторинг оголошень про вразливості та швидке реагування на них за допомогою встановлення оновлень або заміни компонентів.

12. Відмова в обслуговуванні (DoS – Denial of Service):

- Використання механізмів обмеження ресурсів, таких як обмеження швидкості запитів або обмеження розміру даних, що надсилаються.
- Розроблення та реалізація механізмів виявлення та запобігання DoS-атак, таких як фільтрація небезпечних запитів або резервне копіювання ресурсів.

Це лише деякі приклади загальних уразливостей та принципів їх нейтралізації. Важливо також використовувати найновіші версії бібліотек та фреймворків, регулярно оновлювати програмне забезпечення та проводити безпекові аудити для виявлення потенційних проблем у коді.

2. УРАЗЛИВОСТІ JAVA-ДОДАТКІВ ТА ЗАСОБИ ЗАХИСТУ

2.1. Коротко про мову програмування Java

Java є мовою програмування загального призначення, яка була створена в 1995 році, залишається популярною, незважаючи на значний «вік» і великий список нових розробок.

Java – універсальна та входить до списку затребуваних мов. Різноманітність програм, які можна створити з її допомогою, порадує будь-якого веброзробника. Java використовується на серверах, смартпристроях, комп'ютерах, смартфонах. Можна виділити основні сфери використання мови Java, а саме:

1. **Розроблення вебдодатків.** Java використовують для створення серверних компонентів вебдодатків, таких як Java Servlets і JavaServer Pages (JSP). Фреймворки, такі як Spring і JavaServer Faces (JSF), також популярні для веброзроблення на Java.

2. **Розроблення мобільних додатків.** Java використовують для розроблення мобільних додатків для платформи Android. Розробники можуть використовувати Java або мову Kotlin, яка компілюється в байт-код Java.

3. **Розроблення десктопних додатків.** Java також використовується для розроблення десктопних додатків за допомогою бібліотек, таких як JavaFX та Swing. Це дозволяє створювати переносні додатки, які можуть працювати на різних операційних системах.

4. **Вбудовані системи.** Java застосовують у вбудованих системах, таких як системи керування трафіком, системи керування промисловістю та інші. Вона надає зручний і безпечний спосіб програмування для таких систем.

5. **Фінансова сфера.** Багато фінансових установ використовують Java для розроблення програмного забезпечення, такого як торгові системи, оброблення транзакцій та аналіз даних.

Відповідно до направленості існує багато відомих компаній, які працюють із Java: Amazon, eBay, Yahoo!, OpenOffice та ін. Недаремно Java називають мовою Ентерпрайза [12].

Мова програмування Java підтримує безліч принципів програмування, що дозволяє ефективно вирішувати різні завдання.

Розглянемо основні з таких особливостей:

1. **Об'єктно-орієнтоване програмування.** Java підтримує концепцію об'єктно-орієнтованого програмування, що дозволяє створювати модульний код, забезпечуючи поліморфізм, спадкування та інкапсуляцію.

2. **Кросплатформеність.** Код, написаний на Java, може запускатися на будь-якій платформі, що підтримує віртуальну машину Java (JVM). Це дозволяє розробникам писати програми один раз і виконувати їх на різних операційних системах.

3. **Автоматичне керування пам'яттю.** Java використовує механізми збирання сміття для автоматичного вивільнення пам'яті. Розробнику не потрібно безпосередньо керувати виділенням та звільненням пам'яті.

4. **Багатопотоковість.** Java підтримує багатопотоковість, що дозволяє розробникам створювати програми, що виконуються паралельно. Це особливо корисно для розроблення програм, що вимагають одночасного виконання багатьох завдань.

5. **Безпека.** Java має вбудовані механізми безпеки, які дозволяють розробникам створювати безпечні програми. Наприклад, Java має обмежений доступ до

системних ресурсів та контролює виконання небезпечного коду.

Це лише кілька прикладів сфер використання та особливостей мови програмування Java [13].

2.2. Можливості мови Java для забезпечення інформаційної безпеки

Мову Java використовують для розроблення різноманітних програмних додатків: мобільних додатків, вебсерверів, корпоративних систем та багато іншого. Мова Java також має значний потенціал для забезпечення інформаційної безпеки в цифровому світі.

Розглянемо можливості мови Java для забезпечення інформаційної безпеки. Ми дослідимо різні аспекти безпеки, з якими можна стикнутися під час розроблення і програм із використанням Java, а також обговоримо інструменти, методи та практики, які можуть допомогти забезпечити високий рівень захисту в Java-додатках.

2.2.1. Технологічний аспект забезпечення безпеки

Одним із ключових аспектів безпеки є захист від уразливостей програми, які можуть бути використані зловмисниками для несанкціонованого доступу до інформації або злому системи. Мова Java надає кілька механізмів для забезпечення безпеки, таких як система керування пам’яттю, контроль доступу та перевірка типів даних. Першим аспектом, який ми розглянемо, є захист від уразливостей програми в мові Java. Вразливості програм можуть бути використані зловмисниками для несанкціонованого доступу до системи або для злому додатка. Для забезпечення інформаційної безпеки важливо розуміти ці вразливості та приймати заходи для їх запобігання.

Розглянемо за пунктами основні механізми, які відповідають за безпеку в цьому аспекті. Ось деякі з них:

1. **Система керування пам'ятю.** У мові Java використовується автоматичне керування пам'яттю, що дозволяє уникнути багатьох типових уразливостей, пов'язаних із керуванням пам'яттю, таких як переповнення буферу або використання недійсних покажчиків. Гарбедж-колектор Java автоматично видаляє об'єкти, які більше не використовуються, зменшуючи ризик витоку пам'яті та пам'яткових помилок.

2. **Контроль доступу.** Java надає можливість контролювати доступ до класів, методів та змінних за допомогою модифікаторів доступу, таких як public, private та protected. Це дозволяє обмежувати доступ до конкретних частин програми лише для необхідних сутностей. Наприклад, використання модифікатора private для змінних та методів забезпечує, що вони доступні лише в межах самого класу, що допомагає уникнути несанкціонованого доступу.

3. **Перевірка типів даних.** У мові Java всі об'єкти мають визначений тип, і компілятор перевіряє, чи використовуються дані об'єкти відповідно до їх типу. Це дозволяє уникнути помилок, пов'язаних із некоректним використанням даних, і забезпечити більшу надійність та безпеку програми. Наприклад, якщо змінній типу String спробують присвоїти значення типу int, компілятор видаватиме помилку.

4. **Оброблення винятків.** У мові Java використовується механізм оброблення винятків, що дозволяє контролювати та обробляти виняткові ситуації. Правильне оброблення винятків може допомогти запобігти некоректному виконанню програми та захистити її від непередбачуваних помилок або зловмисних атак. Приклад

реалізації в коді цього механізму поданий на рисунку 2.1.1.1.

5. **Валідація вхідних даних.** Перевірка та валідація вхідних даних є важливим аспектом безпеки. У Java розроблено багато бібліотек та інструментів, які допомагають валідувати вхідні дані, такі як Apache Commons Validator або Hibernate Validator. Вони дозволяють перевірити правильність формату, обмеження довжини, наявність уразливостей і т. д., що сприяє запобіганню атак, таких як SQL-ін'єкція або переповнення буфера.

```java
package org.example;

public class ExceptionHandlingExample {

    public static void main(String[] args) {
        try {
            // Код, в якому можуть виникнути виключні ситуації
            int result = divide( dividend: 10,  divisor: 0);
            System.out.println("Результат: " + result);
        } catch (ArithmeticException e) {
            // Обробка виключної ситуації, коли відбувається ділення на нуль
            System.out.println("Помилка: Ділення на нуль");
        } finally {
            // Код, який буде виконано незалежно від наявності виключної ситуації
            System.out.println("Код в блоку finally");
        }
    }

    public static int divide(int dividend, int divisor) {
        if (divisor == 0) {
            throw new ArithmeticException("Ділення на нуль недопустиме");
        }
        return dividend / divisor;
    }
}
```

Рисунок 2.1.1.1 – Приклад реалізації оброблення винятків у мові Java

6. **Безпечність мережевих з'єднань.** Мова Java має багато вбудованих класів та інструментів для роботи з мережами, які допомагають забезпечити безпеку з'єднань. Наприклад, класи javax.net.ssl.SSLSocket або

javax.net.ssl.HttpsURLConnection дозволяють створювати безпечні SSL/TLS-з’єднання для захисту від перехоплення чутливої інформації під час комунікації із серверами. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.1.2.

```java
import javax.net.ssl.HttpsURLConnection;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;

public class SecureConnectionExample {

    public static void main(String[] args) {
        String url = "https://www.example.com";

        try {
            // Створення об'єкту URL з посиланням
            URL apiUrl = new URL(url);

            // Встановлення безпечного SSL/TLS-з'єднання
            HttpsURLConnection connection = (HttpsURLConnection) apiUrl.openConnection();

            // Отримання вхідного потоку для читання відповіді
            InputStream inputStream = connection.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

            // Читання відповіді з сервера
            String line;
            StringBuilder response = new StringBuilder();
            while ((line = reader.readLine()) != null) {
                response.append(line);
            }
```

Рисунок 2.1.1.2 – Приклад реалізації безпечного SSL/TLS-з’єднання за допомогою класу javax.net.ssl.HttpsURLConnection

Рисунок 2.1.1.2, аркуш 2

7. **Безпека файлової системи.** Java надає можливості для роботи з файловою системою, зокрема для забезпечення безпеки доступу до файлів та директорій. Використання відповідних дозволів, аутентифікації та авторизації може допомогти уникнути несанкціонованого доступу до файлової системи та зберегти конфіденційні дані.

8. **Механізми аутентифікації та авторизації.** Java має підтримку різних механізмів аутентифікації та авторизації, таких як JAAS (Java Authentication and Authorization Service) або OAuth. Ці механізми дозволяють перевіряти ідентичність користувачів, надавати їм права доступу до ресурсів та контролювати їхні дії в системі.

9. **Серіалізація та десеріалізація.** Java надає можливості для збереження об’єктів у серіалізованому вигляді та їхнього відновлення через десеріалізацію. Цей процес може викликати ризик уразливостей, таких як ін’єкція коду або виконання небезпечного коду. Для запобігання цим атакам рекомендується використовувати механізми, такі як контроль версій серіалізації, або обмеження серіалізації конфіденційних даних.

10. **Безпека баз даних.** Під час роботи з базами даних важливо забезпечити безпеку доступу до них. Java має підтримку JDBC (Java Database Connectivity), який

дозволяє розробникам установлювати з’єднання з базами даних та виконувати запити. Використання параметризованих запитів та підготовлених виразів допомагає уникнути SQL-ін’єкцій та зберегти безпеку баз даних. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.1.3.

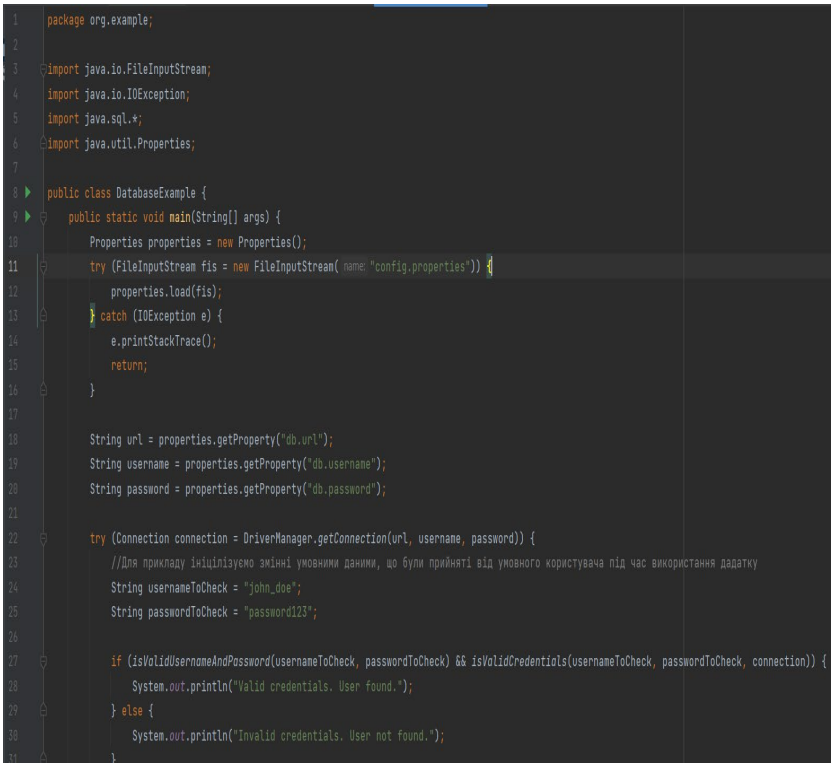


Рисунок 2.1.1.3 – Приклад реалізації безпеки баз даних у Java за допомогою JDBC

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static boolean isValidCredentials(String usernameToCheck, String passwordToCheck, Connection connection) throws SQLException {
        String sql = "SELECT * FROM users WHERE username = ? AND password = ?";

        try (PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setString( parameterIndex: 1, usernameToCheck);
            statement.setString( parameterIndex: 2, passwordToCheck);

            try (ResultSet resultSet = statement.executeQuery()) {
                return resultSet.next();
            }
        }
    }

    private static boolean isValidUsernameAndPassword(String username, String password) {
        // Перевірка на порожні значення
        if (username.isEmpty() || password.isEmpty()) {
            return false;
        }

        // Перевірка довжини
        if (username.length() < 5 || username.length() > 15 || password.length() < 5 || password.length() > 15) {
            return false;
        }
```



```
        // Перевірка наявності букв та чисел у паролі
        boolean hasLetter = false;
        boolean hasDigit = false;
        for (char ch : password.toCharArray()) {
            if (Character.isLetter(ch)) {
                hasLetter = true;
            } else if (Character.isDigit(ch)) {
                hasDigit = true;
            }
            // Якщо знайшли і букву і цифру, далі перевіряти не потрібно
            if (hasLetter && hasDigit) {
                break;
            }
        }

        return hasLetter && hasDigit;
    }
}
```

Рисунок 2.1.1.3, аркуш 2

11. **Аудит безпеки.** Для забезпечення безпеки важливо вести аудит дій, що відбуваються в системі. Java має підтримку різних інструментів для аудиту, таких як логування подій через Log4j або SLF4J. Ці інструменти дозволяють реєструвати дії користувачів, помилки, спроби несанкціонованого доступу та інші події, що допомагає виявити потенційні загрози та реагувати на них.

12. **Механізми захисту від переповнення буфера.** Переповнення буфера є однією з найпоширеніших уразливостей програмного забезпечення. У мові Java існують механізми, такі як SafeVarargs або описувачі масивів (array descriptors), які допомагають уникнути переповнення буфера та зменшити ризик уразливостей. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.1.4.

13. **Захист від перехоплення сеансу.** Для забезпечення безпеки комунікації між клієнтом і сервером Java надає підтримку протоколів, таких як SSL (Secure Sockets Layer) і TLS (Transport Layer Security). Використання цих протоколів дозволяє шифрувати дані під час передавання та захищати їх від перехоплення.

```java
public class BufferOverflowExample {
    @SafeVarargs
    public static <T> void processElements(T... elements) {
        // Обробка елементів без ризику переповнення буфера
        for (T element : elements) {
            System.out.println("Елемент: " + element.toString());
        }
    }

    public static void main(String[] args) {
        String element1 = "Елемент 1";
        String element2 = "Елемент 2";
        String element3 = "Елемент 3";

        // Виклик методу з різними кількостями аргументів
        processElements(element1);
        processElements(element1, element2);
        processElements(element1, element2, element3);
    }
}
```

Рисунок 2.1.1.4 – Приклад використання SafeVarargs для запобігання переповненню буфера під час оброблення аргументів методу

14. **Безпека вебдодатків.** Під час розроблення вебдодатків важливо забезпечити їхню безпеку. Java має підтримку специфікації Java EE (Enterprise Edition), яка включає механізми безпеки, такі як контроль доступу до ресурсів, фільтри безпеки, обробники аутентифікації та авторизації. Використання цих механізмів допомагає захистити вебдодатки від різних атак, таких як перехоплення інформації, міжсайтовий скриптинг (XSS) або міжсайтова підробка запитів (CSRF).

15. **Моніторинг безпеки.** Для ефективного керування безпекою важливо мати механізми моніторингу. У мові Java можна використовувати інструменти, такі як Java Management Extensions (JMX), для збирання та аналізу різних метрик безпеки, таких як активність користувачів, спроби несанкціонованого доступу або виявлення аномальної поведінки. Це допомагає виявити потенційні загрози та реагувати на них швидко. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.1.5.

```
public class SecurityMonitor {
    private static final String MBEAN_NAME = "security:type=SecurityMetrics";
    private int userActivityCount;
    private int unauthorizedAccessAttempts;

    public void incrementUserActivityCount() {
        userActivityCount++;
    }

    public void incrementUnauthorizedAccessAttempts() {
        unauthorizedAccessAttempts++;
    }

    public int getUserActivityCount() {
        return userActivityCount;
    }

    public int getUnauthorizedAccessAttempts() {
        return unauthorizedAccessAttempts;
    }

    public static void main(String[] args) throws Exception {
        // Створення MBean сервера
        MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
```

Рисунок 2.1.1.5 – Приклад реалізації використання Java Management Extensions (JMX) для збирання та аналізу метрик безпеки

Рисунок 2.1.1.5, аркуш 2

16. **Безпека мобільних додатків.** Java також використовується для розроблення мобільних додатків на платформі Android. Під час розроблення мобільних додатків важливо дотримуватися безпекових практик, таких як обмеження дозволів, шифрування чутливих даних, перевірка цілісності підпису додатка тощо. Java забезпечує механізми для реалізації цих практик та забезпечення безпеки мобільних додатків.

17. **Безпека коду.** Важливим аспектом безпеки в мові Java є запобігання вразливостям у самому коді. Розробники можуть використовувати інструменти, такі як статичний аналізатор коду, для виявлення потенційних проблем безпеки, таких як уразливості переповнення буфера, виконання небезпечного коду або вразливості, пов’язані з багатопотоковістю. Такі інструменти допомагають виявити потенційні проблеми безпеки та виправити їх на ранніх етапах розроблення [14].

Ці механізми спільно допомагають створювати більш безпечні програми в мові Java, зменшуючи ризик уразливостей та зловживань. Урахування їх під час проєктування та розроблення додатків може підвищити рівень безпеки та надійності програмного забезпечення.

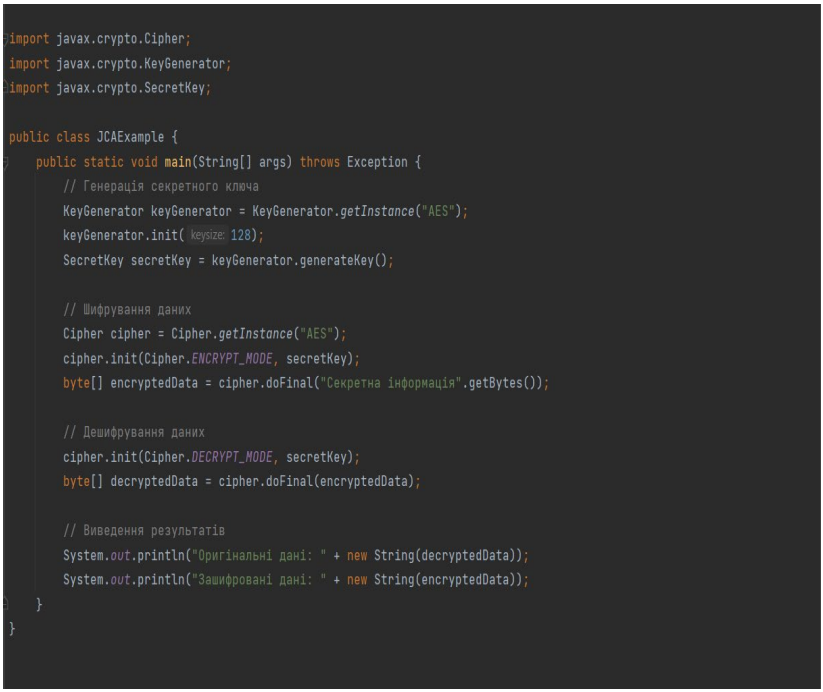
2.2.2. Криптографічний аспект забезпечення безпеки

Другим важливим аспектом є криптографічний захист інформації, що передається та зберігається у Java-додатках. Java має вбудовану підтримку криптографічних алгоритмів та протоколів, що дозволяє надійно захистити дані від несанкціонованого доступу та перехоплення.

Тож спочатку в Java є різноманітні механізми криптографічного захисту інформації, які можна використовувати для зашифрування та підпису даних. Розглянемо декілька таких механізмів:

1. **Java Cryptography Architecture (JCA)**. JCA є стандартною бібліотекою Java, яка надає інтерфейси та реалізації для різноманітних криптографічних алгоритмів. Вона включає шифрування, геш-функції, цифрові підписи та інші криптографічні операції. Ви можете використовувати JCA для захисту інформації в Java-додатках. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.2.1.

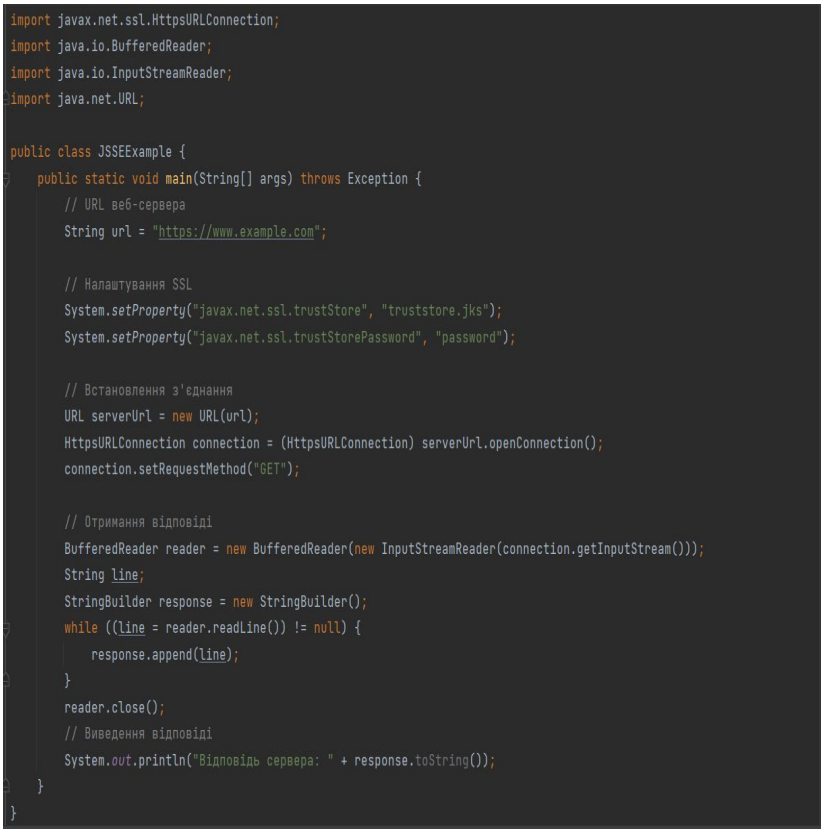
2. **Java Cryptography Extension (JCE)**. JCE є доповненням до JCA і надає більше криптографічних алгоритмів та функцій. Вона дозволяє використовувати сильні алгоритми шифрування, такі як AES, Triple DES та ін. JCE дозволяє також створювати та керувати ключами шифрування, генерувати випадкові числа та виконувати інші криптографічні операції.



```java
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class JCAExample {
    public static void main(String[] args) throws Exception {
        // Генерація секретного ключа
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init( keysize: 128);
        SecretKey secretKey = keyGenerator.generateKey();

        // Шифрування даних
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedData = cipher.doFinal("Секретна інформація".getBytes());

        // Дешифрування даних
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedData = cipher.doFinal(encryptedData);

        // Виведення результатів
        System.out.println("Оригінальні дані: " + new String(decryptedData));
        System.out.println("Зашифровані дані: " + new String(encryptedData));
    }
}
```

Рисунок 2.1.2.1 – Приклад використання Java Cryptography Architecture (JCA) для шифрування та дешифрування даних за допомогою алгоритму AES

3. **Java Secure Socket Extension (JSSE).** JSSE надає безпечний шар транспортного рівня для захисту мережевого зв’язку. Вона підтримує шифрування та аутентифікацію за допомогою протоколів, таких як SSL (Secure Sockets Layer) і TLS (Transport Layer Security). JSSE можна використовувати для захисту комунікації між Java-додатками та віддаленими серверами. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.2.2.



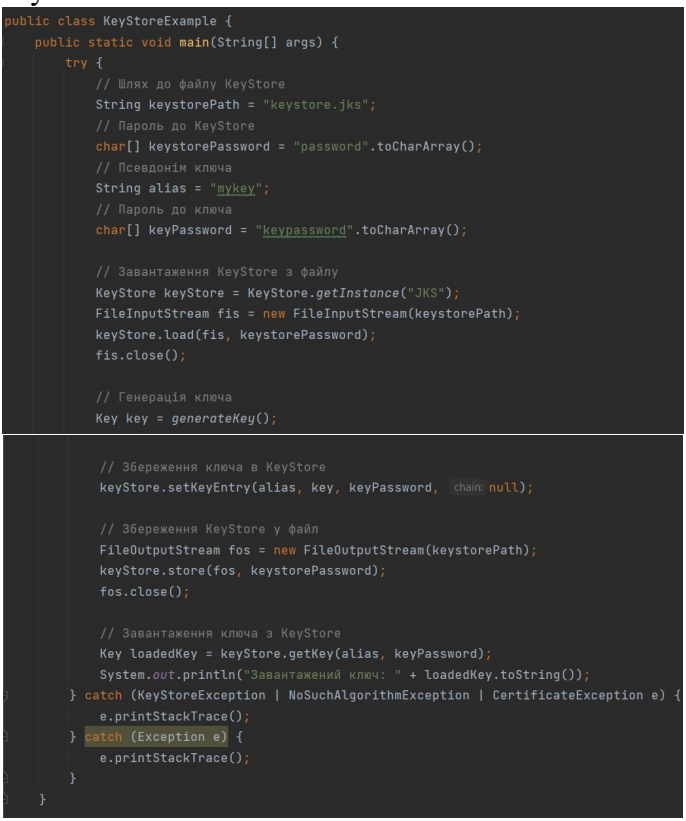
```java
import javax.net.ssl.HttpsURLConnection;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;

public class JSSEExample {
    public static void main(String[] args) throws Exception {
        // URL веб-сервера
        String url = "https://www.example.com";

        // Налаштування SSL
        System.setProperty("javax.net.ssl.trustStore", "truststore.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "password");

        // Встановлення з'єднання
        URL serverUrl = new URL(url);
        HttpsURLConnection connection = (HttpsURLConnection) serverUrl.openConnection();
        connection.setRequestMethod("GET");

        // Отримання відповіді
        BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        String line;
        StringBuilder response = new StringBuilder();
        while ((line = reader.readLine()) != null) {
            response.append(line);
        }
        reader.close();
        // Виведення відповіді
        System.out.println("Відповідь сервера: " + response.toString());
    }
}
```

Рисунок 2.1.2.2 – Приклад використання Java Secure Socket Extension (JSSE) для встановлення безпечного SSL-з’єднання з вебсервером

4. **Bouncy Castle**. Bouncy Castle є незалежною бібліотекою криптографії для Java. Вона надає реалізації різноманітних криптографічних алгоритмів, включаючи шифрування, геш-функції, цифрові підписи та інші. Bouncy Castle також підтримує додаткові алгоритми та формати, які можуть бути корисними для певних сценаріїв застосування.

5. **KeyStore API.** KeyStore API дозволяє зберігати і керувати ключами та сертифікатами в Java. Ви можете

використовувати KeyStore API для зберігання приватних ключів, сертифікатів та інших матеріалів криптографічного ключа. KeyStore може бути використаний для захисту важливих конфіденційних даних у Java-додатках. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.2.3.



```
public class KeyStoreExample {
    public static void main(String[] args) {
        try {
            // Шлях до файлу KeyStore
            String keystorePath = "keystore.jks";
            // Пароль до KeyStore
            char[] keystorePassword = "password".toCharArray();
            // Псевдонім ключа
            String alias = "mykey";
            // Пароль до ключа
            char[] keyPassword = "keypassword".toCharArray();

            // Завантаження KeyStore з файлу
            KeyStore keyStore = KeyStore.getInstance("JKS");
            FileInputStream fis = new FileInputStream(keystorePath);
            keyStore.load(fis, keystorePassword);
            fis.close();

            // Генерація ключа
            Key key = generateKey();

            // Збереження ключа в KeyStore
            keyStore.setKeyEntry(alias, key, keyPassword, chain: null);

            // Збереження KeyStore у файл
            FileOutputStream fos = new FileOutputStream(keystorePath);
            keyStore.store(fos, keystorePassword);
            fos.close();

            // Завантаження ключа з KeyStore
            Key loadedKey = keyStore.getKey(alias, keyPassword);
            System.out.println("Завантажений ключ: " + loadedKey.toString());
        } catch (KeyStoreException | NoSuchAlgorithmException | CertificateException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Рисунок 2.1.2.3 – Приклад використання KeyStore API для зберігання та завантаження ключа з KeyStore

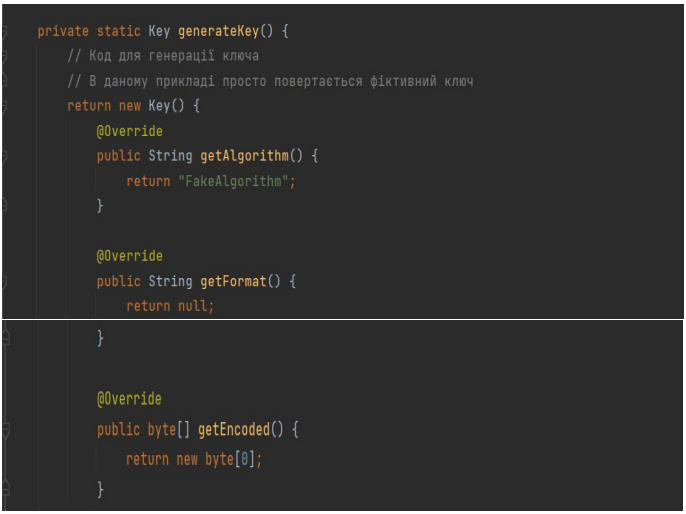


Рисунок 2.1.2.3, аркуш 2

6. **Java KeyStore (JKS).** Java KeyStore є форматом файлу, який використовується для зберігання приватних ключів, сертифікатів та цілого ряду інших криптографічних матеріалів. Ви можете створити, завантажити та керувати JKS-файлами з використанням Java KeyStore API.

7. **SecureRandom**. SecureRandom – це клас у Java, який надає криптографічно безпечні випадкові числа. Він може бути використаний для генерації випадкових значень, таких як ключі шифрування, ініціалізаційні вектори. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.2.4.

```java
import java.security.SecureRandom;

public class SecureRandomExample {
    public static void main(String[] args) {
        try {
            // Створення об'єкту SecureRandom
            SecureRandom secureRandom = new SecureRandom();

            // Генерація випадкового числа
            byte[] randomBytes = new byte[16];
            secureRandom.nextBytes(randomBytes);

            // Виведення результату
            System.out.println("Випадкове число: " + byteArrayToHex(randomBytes));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static String byteArrayToHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (byte b : bytes) {
            sb.append(String.format("%02x", b));
        }
        return sb.toString();
    }
}
```

Рисунок 2.1.2.4 – Приклад використання класу SecureRandom для генерації випадкового числа

8. **Cipher**. Cipher – це клас в Java, який використовується для шифрування та розшифрування даних. Він дає можливість використовувати різні алгоритми шифрування, такі як AES, DES або RSA. Ви можете використовувати Cipher для захисту конфіденційних даних в Java-додатках.

9. **MessageDigest**. MessageDigest – це клас у Java, який використовується для обчислення геш-функцій. Він дає можливість використовувати різні алгоритми гешування, такі як MD5, SHA-1 або SHA-256. MessageDigest можна використовувати для перетворення даних в унікальний геш-код, що може бути використаний для перевірки цілісності даних.

10. **Signature**. Signature – це клас в Java, який використовується для генерації та перевірки цифрових підписів. Він дає можливість використовувати різні алгоритми підпису, такі як RSA або DSA. Signature може бути використаний для забезпечення автентичності та цілісності даних. Приклад реалізації в коді цього механізму поданий на рисунку 2.1.2.5.

```java
import java.security.*;

public class SignatureExample {
    public static void main(String[] args) {
        try {
            // Створення об'єкту KeyPairGenerator для генерації ключів
            KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
            keyGen.initialize( keysize: 2048);
            KeyPair keyPair = keyGen.generateKeyPair();

            // Отримання приватного та публічного ключів
            PrivateKey privateKey = keyPair.getPrivate();
            PublicKey publicKey = keyPair.getPublic();

            // Створення об'єкту Signature
            Signature signature = Signature.getInstance("SHA256withRSA");

            // Ініціалізація об'єкту Signature з приватним ключем для підпису
            signature.initSign(privateKey);

            // Встановлення даних, які будуть підписані
            String data = "Hello, world!";
            signature.update(data.getBytes());

            // Генерація цифрового підпису
            byte[] digitalSignature = signature.sign();

            // Виведення результату
            System.out.println("Цифровий підпис: " + byteArrayToHex(digitalSignature));
```

Рисунок 2.1.2.5 – Приклад використання класу Signature для генерації та перевірки цифрового підпису

Рисунок 2.1.2.5, аркуш 2

Ці механізми надають потужні інструменти для криптографічного захисту інформації в Java-додатках. Так вони надають багато функцій і можливостей для забезпечення безпеки даних у ваших додатках. Ви можете використовувати їх для різних сценаріїв застосування, включаючи шифрування, цифрові підписи, хешування та багато іншого.

2.3. Приклади вразливостей мови Java, її бібліотек та фреймворків

Розглянемо та проаналізуємо декілька відомих прикладів уразливостей, виявлених у мові програмування Java, а також у її бібліотеках та фреймворках.

Мова Java використовується в різних сферах програмування, проте вона також має потенційні вразливості, які можуть бути використані зловмисниками для зламу програмного коду або зловживання.

У цьому підпункті будуть розглянуті різні приклади вразливостей, що мали реалізацію в минулому та чинили вагомий вплив на індустрію, в якій виникали.

Аналізуючи ці приклади вразливостей, ми отримаємо більш глибоке розуміння потенційних ризиків, що стосуються розроблення на Java, та зможемо вжити відповідних заходів для запобігання та виправлення цих проблем. Цей розділ містить корисний огляд інформації, яка допоможе програмістам та розробникам забезпечити високий рівень безпеки своїх проєктів, що базуються на мові Java та використовують її бібліотеки та фреймворки.

1. Однією з найбільш відомих та критичних вразливостей, що стосуються мови Java та її бібліотек, є вразливість Log4Shell у Apache Log4j 2. Ця вразливість стала відомою глобальною проблемою безпеки в грудні 2021 року і швидко набула значної популярності через свою серйозність та потенційну загрозу для багатьох систем, які використовують Log4j.

Log4j є потужним і популярним фреймворком для оброблення журнальної інформації в програмах на Java.

Вразливість Log4Shell, також відома як CVE-2021-44228, стосується Apache Log4j 2 – популярного фреймворку для оброблення журнальних повідомлень у Java-додатках.

Основна проблема виникає через недостатню перевірку даних, переданих у рядку журнального повідомлення. Зловмисник може сконструювати спеціально сформований рядок, який містить виконуваний код, і впливати на поведінку інфраструктури, що використовує Log4j. Під час оброблення такого рядка журнального повідомлення Log4j виконує код, який у ньому міститься, що може призвести до виконання довільних команд на сервері або зламу системи.

Ця вразливість особливо небезпечна через свої можливості віддаленого виконання коду (RCE), що дозволяє зловмиснику виконувати будь-який код на системі без необхідності автентифікації. Зловмисники можуть скористатися цим, щоб отримати несанкціонований доступ до серверів, використати їх для поширення шкідливих програм або зламувати конфіденційну інформацію.

Вразливість Log4Shell стала особливо підозрілою та відомою після того, як були опубліковані її деталі. Це призвело до появи активних атак, а також до виходу швидких виправлень та патчів для Apache Log4j 2 з боку розробників.

Захист від цієї вразливості включає оновлення Apache Log4j 2 до виправленої версії та вживання додаткових заходів безпеки, таких як обмеження вхідних даних, налаштування фільтрів журналу та встановлення вогнезахисних правил на рівні мережі. Крім того, розробники й адміністратори систем повинні регулярно відстежувати оновлення та вразливості у своєму програмному забезпеченні, а також стежити за новими випусками патчів та виправлень для Apache Log4j 2 [15].

2. Вразливість unserialize в бібліотеці commons-collections для мови Java, зокрема для Java Application Server, є серйозною проблемою безпеки, яку необхідно вирішити.

Ця вразливість виникає через проблему в процесі десеріалізації, що дозволяє зловмиснику виконати довільний код під час десеріалізації об’єкта. Маніпулюванням серіалізованими даними зловмисник може створити шкідливий код, який під час десеріалізації виконується з несанкціонованими діями.

Бібліотека commons-collections широко використовується в Java-додатках і фреймворках, тому ця вразливість є серйозною проблемою. Використання цієї

вразливості може мати серйозні наслідки, такі як віддалене виконання коду, несанкціонований доступ до конфіденційної інформації або навіть повне компрометування системи.

Для зменшення ризику, пов'язаного з вразливістю unserialize в бібліотеці commons-collections, важливо дотримуватися таких рекомендацій щодо безпеки:

1. **Оновлення до останньої версії.** Будьте в курсі оновлень та виправлень безпеки, які випускаються проєктом commons-collections. Регулярно оновлюйте ваші залежності, щоб включити нові виправлення та покращання безпеки.

2. **Забезпечення безпеки десеріалізації**. Реалізуйте відповідну перевірку та фільтрацію вхідних даних під час процесу десеріалізації. Перевіряйте та очищуйте серіалізовані дані, щоб запобігти виконанню шкідливого коду.

3. **Принцип найменших привілеїв**. Переконайтеся, що десеріалізовані об'єкти мають лише необхідні дозволи та привілеї. Обмежте доступ та можливості десеріалізованих об'єктів, щоб мінімізувати потенційні наслідки експлуатації.

4. **Тестування безпеки**. Проводьте ретельне тестування безпеки, включаючи перевірку коду, сканування вразливостей та пенетраційне тестування, щоб виявити та усунути потенційні вразливості, пов'язані з використанням бібліотеки commons-collections.

Вирішення вразливості unserialize в бібліотеці commons-collections та дотримання цих заходів безпеки допоможе значно зменшити ризик експлуатації та захистить ваші Java-додатки та системи від можливих порушень безпеки [16].

3. Вразливість Spring4Shell в Spring Core є серйозною проблемою безпеки, яку необхідно врахувати

під час розроблення та використання додатків, що побудовані на цьому фреймворку.

Spring Core є одним із найпопулярніших фреймворків розроблення додатків мовою Java. Однак у версії Spring4Shell була виявлена вразливість, яка може дозволити зловмиснику виконувати небезпечний код через командний рядок.

Ця вразливість виникає через недостатню перевірку та санітизацію вхідних даних, які використовуються в командних рядках. Зловмисник може сконструювати спеціально сформований рядок, що містить шкідливий код, і передати його в додаток, підмінюючи правильні команди. Під час оброблення цього рядка в додатку виконується шкідливий код, що може призвести до виконання небезпечних дій або розкриття конфіденційної інформації.

Наслідки використання вразливості Spring4Shell можуть бути серйозними, включаючи виконання небезпечних команд на системі, зміну конфігурації додатка, доступ до конфіденційної інформації та потенційну компрометацію системи.

Для запобігання використанню вразливості Spring4Shell у Spring Core рекомендується дотримуватися таких заходів безпеки:

1. **Оновлення**. Переконайтеся, що ви використовуєте останню версію Spring Core, яка містить виправлення для вразливостей. Регулярно оновлюйте ваші залежності, щоб мати найсвіжіші виправлення безпеки.

2. **Перевірка та санітизація вхідних даних**. Упроваджуйте належну перевірку та санітизацію вхідних даних, особливо тих, які передаються через командний рядок. Використовуйте функції фільтрації та екранування, щоб усунути потенційно небезпечний код.

3. **Привілеї та обмеження**. Надайте додатку лише необхідні привілеї та обмеження. Зменшення прав доступу

до окремих компонентів та функцій додатка може допомогти зменшити можливі наслідки вразливості.

4. **Аудит безпеки**. Проводьте регулярну перевірку безпеки вашого додатка, включаючи аудит коду, сканування вразливостей та тестування на проникнення. Виявлення та виправлення потенційних вразливостей є важливими кроками для забезпечення безпеки вашого додатка.

Дотримуючись цих рекомендацій безпеки, ви зможете значно зменшити ризик експлуатації вразливості Spring4Shell в Spring Core та забезпечити безпеку вашого додатка на основі цього фреймворку [17, 18].

4. Вразливість RCE (виконання віддаленого коду) в Apache Struts 2 є однією з найвідоміших та серйозних вразливостей, які були виявлені в цьому фреймворку розроблення вебдодатків. Вона дозволяє зловмисникам виконувати віддалений код на сервері, зловживаючи певними недоліками в обробленні вхідних даних.

Ця вразливість виникає через неякісну або недостатню перевірку та санітизацію користувацьких вхідних даних, що передаються в параметрах URL або через HTTP-запити. Зловмисник може сконструювати спеціально сформований запит, включаючи в ньому шкідливий код, який потім буде виконуватися на сервері.

Наслідки використання вразливості RCE в Apache Struts 2 можуть бути надзвичайно серйозними. Зловмисник може виконати довільний код на сервері, отримати несанкціонований доступ до файлової системи, бази даних або інших ресурсів сервера. Це може призвести до викрадення чутливої інформації, компрометації додатків або навіть повного контролю над сервером.

Для запобігання використанню вразливості RCE в Apache Struts 2 рекомендується вживати такі заходи безпеки:

1. **Оновлення**. Переконайтеся, що ви використовуєте останню версію Apache Struts 2, яка містить виправлення для вразливостей. Регулярно оновлюйте ваші залежності, щоб мати найсвіжіші виправлення безпеки.

2. **Валідація вхідних даних**. Перевіряйте та санітизуйте всі вхідні дані, які передаються в параметрах URL або через HTTP-запити. Використовуйте механізми фільтрації та екранування, щоб запобігти виконанню шкідливого коду.

3. **Обмеження привілеїв**. Надайте додатку лише необхідні привілеї та обмеження. Зменшення прав доступу до окремих компонентів та функцій додатка може допомогти зменшити можливі наслідки вразливості.

4. **Аудит безпеки**. Регулярно перевіряйте безпеку вашого додатка, включаючи аудит коду, сканування вразливостей та тестування на проникнення. Виявлення та виправлення потенційних вразливостей є важливими кроками для забезпечення безпеки додатка на базі Apache Struts 2 [19].

2.4. Популярні фреймворки та бібліотеки Java, які допомагають забезпечити інформаційну безпеку додатків

У сучасному цифровому світі, де безпека інформації стає дедалі більшою проблемою, захист додатків від потенційних загроз і кібератак є надзвичайно важливим завданням. З метою забезпечення інформаційної безпеки, розробники програмного забезпечення шукають ефективні інструменти та рішення, які можуть допомогти їм у цьому процесі.

Один із напрямків, що стає дедалі більш популярним, – це використання фреймворків та бібліотек, які спеціалізуються на забезпеченні безпеки додатків. У

цьому розділі ми розглянемо кілька популярних фреймворків та бібліотек Java, які можуть бути використані для створення безпечних програмних рішень.

1. Перший фреймворк, на який варто звернути увагу, – це Spring Security. Spring Security є одним Із найпопулярніших фреймворків безпеки для додатків на платформі Java. Він надає широкий спектр інструментів для аутентифікації, авторизації та керування сеансами, що допомагає розробникам створювати безпечні додатки зі зручною інтеграцією в різні архітектури.

Одним із ключових компонентів Spring Security є фільтр безпеки, який обробляє запити та виконує необхідні перевірки безпеки перед передачею їх до основної логіки додатка. Наприклад, він може перевіряти, чи користувач має необхідні права доступу до певних ресурсів або дозволяти виконання певних дій лише автентифікованим користувачам.

Одним із важливих аспектів Spring Security є його підтримка різних методів аутентифікації, включаючи базову аутентифікацію, форму аутентифікацію, аутентифікацію з використанням сторонніх служб (наприклад, OAuth) та багато інших. Для кожного методу аутентифікації існує відповідний провайдер, який виконує перевірку правильності інформації для аутентифікації (рис. 2.3.1).

Рисунок 2.3.1 – Приклад конфігурації Spring Security у спрощеному вебдодатку

У цьому прикладі ми налаштовуємо рівень доступу до ресурсів. **/public** доступний всім, а **/private** вимагає аутентифікації. Ми також налаштовуємо форму входу на **/login** і виходу на **/logout**.

Крім того, ми використовуємо **UserDetailsService**, який забезпечує інформацію про користувачів для аутентифікації. Ви можете створити власний клас, який реалізує інтерфейс **UserDetailsService** і містить логіку завантаження даних користувачів із бази даних або іншого джерела.

Це лише спрощений приклад конфігурації Spring Security. Фреймворк дає багато інших можливостей, таких як контроль доступу на основі ролей, захист від атак типу Cross-Site Request Forgery (CSRF), використання двофакторної аутентифікації та багато інших. Ви можете глибше дослідити ці можливості в офіційній документації Spring Security та додаткових ресурсах [20].

2. Другим інструментом, який варто врахувати, є Bouncy Castle з відкритою криптографічною бібліотекою для платформи Java, яка підтримує широкий спектр криптографічних алгоритмів, включаючи шифрування, підписи, геш-функції, генерацію ключів та багато інших. Вона надає реалізації алгоритмів, таких як AES, RSA, DSA, ECDSA, SHA, MD5 та багато інших (рис. 2.3.2). Вона є популярним вибором для розроблення безпечних додатків, які вимагають сильної криптографічної підтримки.

```java
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import java.security.Security;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import javax.crypto.Cipher;

public class BouncyCastleExample {
    public static void main(String[] args) throws Exception {
        Security.addProvider(new BouncyCastleProvider());

        // Генерація ключа
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance( algorithm: "RSA",  provider: "BC");
        keyPairGenerator.initialize( keysize: 2048);
        KeyPair keyPair = keyPairGenerator.generateKeyPair();

        // Шифрування даних
        Cipher cipher = Cipher.getInstance( transformation: "RSA/ECB/PKCS1Padding",  provider: "BC");
        cipher.init(Cipher.ENCRYPT_MODE, keyPair.getPublic());
        byte[] encryptedData = cipher.doFinal("Hello, World!".getBytes());

        System.out.println("Encrypted data: " + new String(encryptedData));
    }
}
```

Рисунок 2.3.2 – Приклад використання Bouncy Castle для генерації ключа та шифрування даних

У цьому прикладі ми спочатку додаємо провайдер Bouncy Castle (**BouncyCastleProvider**) до системи, щоб мати доступ до криптографічних алгоритмів, які надає ця бібліотека.

Далі ми використовуємо **KeyPairGenerator** для генерації ключів RSA. Потім ми створюємо об’єкт **Cipher** для шифрування даних, використовуючи одержаний публічний ключ. За допомогою **init** ми встановлюємо режим шифрування, а потім викликаємо **doFinal** для шифрування рядка "Hello, World!".

Bouncy Castle також надає підтримку для багатьох інших криптографічних алгоритмів та операцій, таких як підписи, гешування, генерація випадкових чисел і багато

інших. Ви можете досліджувати ці можливості у документації Bouncy Castle та прикладах, що надаються разом із бібліотекою [21].

3. Також варто згадати Apache Shiro, що є потужним фреймворком для безпеки та керування доступом у додатках на платформі Java. Він надає простий та елегантний спосіб виконання аутентифікації, авторизації, керування сеансами та інших аспектів безпеки в додатках.

Основні компоненти Apache Shiro:

1. **Subject**. Це представлення поточного користувача або системного об'єкта. Subject може бути користувачем, аутентифікованим користувачем, групою або будь-яким суб'єктом, якому необхідно надати доступ до ресурсів.

2. **SecurityManager**. Відповідає за керування всіма безпековими операціями, такими як аутентифікація, авторизація, керування сеансами. Він координує роботу різних компонентів безпеки та забезпечує інтеграцію з вашим додатком.

3. **Realm**. Realm використовується для отримання даних про користувачів, ролі, дозволи та іншу інформацію, необхідну для аутентифікації та авторизації. Ви можете створити власний Realm, щоб підключити вашу базу даних, LDAP або будь-яке інше джерело даних.

4. **SessionManager**. Відповідає за керування сеансами користувачів. Він дозволяє створювати, завантажувати та закінчувати сеанси, а також працювати з атрибутами сеансу.

Для створення прикладу використання Apache Shiro для аутентифікації та авторизації користувача потрібно налаштувати **SecurityManager** та **Realm** у вашому додатку (рис. 2.3.3).

```java
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.authc.*;
import org.apache.shiro.config.IniSecurityManagerFactory;
import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.subject.Subject;
import org.apache.shiro.util.Factory;

public class ShiroExample {
    public static void main(String[] args) {
        // Створення SecurityManager з конфігураційного файла shiro.ini
        Factory<SecurityManager> factory = new IniSecurityManagerFactory( iniResourcePath: "classpath:shiro.ini");
        SecurityManager securityManager = factory.getInstance();
        SecurityUtils.setSecurityManager(securityManager);

        // Отримання Subject (користувача)
        Subject currentUser = SecurityUtils.getSubject();

        // Аутентифікація користувача
        if (!currentUser.isAuthenticated()) {
            UsernamePasswordToken token = new UsernamePasswordToken( username: "johndoe",  password: "password");
            try {
                currentUser.login(token);
                System.out.println("Аутентифікація пройшла успішно!");
            } catch (UnknownAccountException uae) {
                System.out.println("Невідомий обліковий запис");
            } catch (IncorrectCredentialsException ice) {
                System.out.println("Неправильні облікові дані");
            } catch (LockedAccountException lae) {
                System.out.println("Обліковий запис заблокований");
            } catch (AuthenticationException ae) {
                System.out.println("Помилка аутентифікації");
            }
        }

        // Перевірка авторизації користувача
        if (currentUser.isAuthenticated()) {
            if (currentUser.hasRole( s: "admin")) {
                System.out.println("Користувач має роль admin");
            } else {
                System.out.println("Користувач не має ролі admin");
            }
        }

        // Вихід користувача
        currentUser.logout();
    }
}
```

Рисунок 2.3.3 – Приклад використання Apache Shiro для аутентифікації та авторизації користувача

У цьому прикладі ми спочатку створюємо **SecurityManager** з конфігураційного файлу shiro.ini. Потім отримуємо поточного **Subject** (користувача). Якщо користувач не аутентифікований, ми виконуємо аутентифікацію з використанням **UsernamePasswordToken**. Після аутентифікації ми перевіряємо авторизацію користувача за допомогою **hasRole**. Наприкінці ми викликаємо **logout**, щоб вийти із системи.

Фреймворк також надає багато інших можливостей, таких як авторизація на основі ролей і дозволів, керування сеансами, криптографічні функції та багато інших. Ви можете дослідити ці можливості в документації Apache Shiro та офіційних прикладах [22].

4. Забезпечення захисту додатків від вразливостей типу Cross-Site Scripting (XSS) може бути досягнуто за допомогою OWASP Java Encoder. OWASP Java Encoder – це бібліотека, розроблена спільнотою OWASP (Open Web Application Security Project), яка надає безпечне кодування і екранування даних у додатках Java. Вона призначена для запобігання вразливостей, таких як XSS (міжсайтовий скриптинг) та SQL-ін’єкції, шляхом належного оброблення та екранування вхідних даних (рис. 2.3.4).

Основні можливості OWASP Java Encoder:

1. **Безпечне кодування**. Бібліотека дає методи для безпечного кодування різних типів даних, таких як рядки, URL-параметри, HTML-теги, JSON-рядки та інші. Це дозволяє запобігти XSS-атакам за допомогою перетворення потенційно небезпечних символів у безпечний формат.

2. **Екранування рядків для SQL-запитів**. OWASP Java Encoder дає методи для екранування рядків, що використовуються в SQL-запитах. Це допомагає уникнути SQL-ін’єкцій, де шкідливі дані можуть бути введені в SQL-запит і виконані небезпечним способом.

3. **Підтримка різних кодувань**. Бібліотека підтримує різні кодування, такі як HTML-ентиті, URL-кодування, Base64, JavaScript-рядки та інші. Це дозволяє вам вибрати потрібний тип кодування залежно від контексту і вимог безпеки вашого додатка.

4. **Надійна обробка некоректних вхідних даних**. Бібліотека володіє механізмами для надійного оброблення некоректних або невалідних вхідних даних. Вона спрощує оброблення помилок і може забезпечити вам контроль над тим, як обробляються неправильні дані.

```
import org.owasp.encoder.Encode;

public class EncoderExample {
    public static void main(String[] args) {
        String userInput = "<script>alert('XSS attack!');</script>";
        String encodedInput = Encode.forHtml(userInput);
        System.out.println(encodedInput);
    }
}
```

Рисунок 2.3.4 – Приклад використання OWASP Java Encoder для безпечного кодування HTML-рядків

У цьому прикладі ми використовуємо метод **Encode.forHtml()** для безпечного кодування рядка **userInput**, який містить потенційно небезпечний HTML-код. Результатом буде безпечно закодований рядок, що може бути виведений у HTML-сторінці без ризику XSS-атак.

OWASP Java Encoder дозволяє використовувати різні методи кодування та екранування для різних типів даних і контекстів безпеки. Ця бібліотека може стати корисним інструментом для покращання безпеки вашого додатка Java та запобігання різного роду вразливостям.

Більш детальну інформацію можна знайти в офіційній документації OWASP Java Encoder [23].

5. Наостанок, Keycloak, він є відкритим ідентифікаційним та управлінським рішенням, яке дає централізовану аутентифікацію, авторизацію та керування доступом для додатків та сервісів. Він побудований на основі стандартів безпеки, таких як OAuth 2.0 і OpenID Connect, і забезпечує безпечний доступ до ресурсів для користувачів, які використовують додатки на платформі Java (рис. 2.3.6).

Основні можливості Keycloak:

1. **Аутентифікація та авторизація**. Keycloak дає можливість аутентифікації користувачів, використовуючи різні методи, такі як ім'я користувача та пароль, соціальні мережі, віддалені системи і т. д. Він також дає механізми для авторизації, враховуючи керування ролями та дозволами.

2. **Single Sign-On (SSO)**. Keycloak підтримує SSO, що означає, що користувачі можуть використовувати один набір облікових даних для доступу до різних додатків без повторної аутентифікації. Це спрощує життя користувачам і поліпшує їх досвід використання додатків.

3. **Керування ресурсами**. Keycloak дозволяє визначати та керувати різними ресурсами, до яких користувачі мають доступ. Ви можете встановлювати різні рівні доступу для користувачів, ролей, груп та інших сутностей.

4. **Інтеграція з додатками Java**. Keycloak надає бібліотеки та розширення для інтеграції з додатками на платформі Java. Це дозволяє забезпечити безпеку додатків використанням стандартів безпеки, таких як OAuth 2.0 і OpenID Connect.

5. **Конфігурація та адміністрування**. Keycloak має вебінтерфейс, що дозволяє адміністраторам керувати

налаштуваннями системи, користувачами, ролями, дозволами та іншими параметрами безпеки. Він також надає API для автоматизації адміністрування.

```java
import org.keycloak.KeycloakPrincipal;
import org.keycloak.adapters.RefreshableKeycloakSecurityContext;
import org.keycloak.representations.AccessToken;

import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.Response;

@Path("/api")
public class MyResource {

    @GET
    @Path("/protected")
    public Response getProtectedResource(@Context HttpServletRequest request) {
        // Отримуємо аутентифікованого користувача
        KeycloakPrincipal<RefreshableKeycloakSecurityContext> principal = (KeycloakPrincipal<RefreshableKeycloakSecurityContext>) request.getUserPrincipal();
        // Отримуємо контекст безпеки
        RefreshableKeycloakSecurityContext securityContext = principal.getKeycloakSecurityContext();
        // Отримуємо токен доступу
        AccessToken accessToken = securityContext.getToken();

        // Перевіряємо роль користувача
        if (accessToken.getRealmAccess().isUserInRole("admin")) {
            // Доступ дозволено
            return Response.ok( entity: "Доступ дозволено").build();
        } else {
            // Доступ заборонено
            return Response.status(Response.Status.FORBIDDEN).entity("Доступ заборонено").build();
        }
    }
}
```

Рисунок 2.3.5 – Приклад використання Keycloak для аутентифікації користувача в додатку Java

У цьому прикладі ми маємо ресурс **/protected**, до якого доступ мають лише користувачі з роллю "admin". У методі **getProtectedResource** ми одержуємо аутентифікованого користувача з використанням **HttpServletRequest**. Потім ми одержуємо контекст безпеки та токен доступу, які надають інформацію про аутентифікованого користувача. Ми перевіряємо, чи має користувач роль "admin" і повертаємо відповідний результат.

Цей приклад показує, як Keycloak може використовуватися для захисту ресурсів із використанням ролей та авторизації. Зверніть увагу, що конфігурація Keycloak, налаштування ролей та дозволів, інтеграція з додатком та інші аспекти пов'язані з використанням Keycloak будуть вимагати додаткової настройки та налаштування.

Функціональність Keycloak набагато ширша, ніж у прикладі, і може бути налаштована та розширена для ваших потреб. Детальніше про Keycloak можна дізнатися з його офіційної документації та прикладів [24].

Окрім згаданих фреймворків та бібліотек, існує ще багато інших інструментів у світі Java, які можуть допомогти забезпечити безпеку додатків. Залежно від специфічних потреб роботи, можна розглянути такі фреймворки та бібліотеки:

1. **Java Web Application Firewalls (WAF)**. WAF – це програмне забезпечення, яке використовується для захисту вебдодатків від уразливостей та кібератак. Деякі популярні Java WAF включають ModSecurity, OWASP ModSecurity Core Rule Set (CRS) та другорядні продукти, які надають спеціалізований захист.

2. **Apache Santuario**. Це бібліотека, яка надає підтримку для безпеки XML та цифрового підпису. Apache Santuario може бути використана для роботи з електронними підписами, шифруванням та перевіркою цифрових підписів у XML-документах.

3. **Hibernate Validator**. Ця бібліотека надає інструменти для валідації вхідних даних у додатках, що використовують фреймворк Hibernate. Вона дозволяє встановлювати правила перевірки та забезпечує безпеку даних на рівні введення.

4. **Apache Commons Crypto**. Це бібліотека, яка надає інтерфейс для криптографічних операцій, таких як

шифрування та гешування даних. Вона побудована на основі Java Cryptography Architecture (JCA) та Java Cryptography Extension (JCE), забезпечуючи зручний спосіб використання криптографії в додатках.

Ці інструменти представляють лише деякі з можливих варіантів для забезпечення безпеки додатків на платформі Java. Однак під час вибору будь-якого фреймворку або бібліотеки важливо звернути увагу на їх функціональність, надійність та підтримку спільнотою [25, с. 41–51, 139–172].

3. НАБУТТЯ ПРАКТИЧНИХ НАВИЧОК БЕЗПЕЧНОГО ПРОГРАМУВАННЯ JAVA-ДОДАТКІВ

У цьому розділі пропонуються тренувальні вправи, спрямовані на підвищення рівня навичок у безпечному програмуванні додатків мовою Java. Ці вправи розроблені спеціально для студентів середнього рівня знань та досвіду, які мають базові знання мови Java та програмування загалом.

Кожне завдання дає можливість практичного використання концепцій та методів безпечного програмування. Вони охоплюють такі важливі аспекти, як перевірка введених даних, оброблення виключень, захист від SQL-ін'єкцій, захист від переповнення буфера, аутентифікація та авторизація. А також основи використання фреймворків для забезпечення безпеки додатків на Java.

Кожна вправа має чіткі постановки завдань та орієнтована на розвиток конкретних навичок. Для успішного виконання вправ потрібно застосовувати набуті знання з безпечного програмування, а також розробляти творчі підходи та аналітичні навички для виявлення потенційних проблем та їх вирішення. Засвоєні навички будуть корисними в галузі кібербезпеки та розробки програмного забезпечення.

3.1. Приклад виконання типового завдання

Розглянемо детальний алгоритм вирішення типового завдання. Тренуючи практичні навички безпечного програмування не варто забувати про інші важливі аспекти, такі як документування коду та дотримання відповідної стилістики. Адже лише за повного розуміння процесу написання коду можна буде створити дійсно безпечний програмний додаток.

Розглянемо завдання, суть якого – навчитися боротися з переповненням буфера та повністю керувати цим процесом.

Умова завдання:

1. Напишіть програму, яка приймає вхідні дані від користувача та перевіряє їх довжину перед збереженням у масив. Обмежте довжину введеного рядка, щоб запобігти переповненню буфера.

- Створіть метод, який приймає рядок, введений користувачем, та перевіряє його довжину.
- Якщо довжина рядка перевищує задане значення, виведіть повідомлення про помилку.
- В іншому разі збережіть введений рядок у масив або відповідну змінну.

2. Створіть функцію, яка отримує рядок та перевіряє, чи його довжина не перевищує заданий ліміт. Якщо рядок занадто довгий, обріжте його до максимально допустимої довжини.

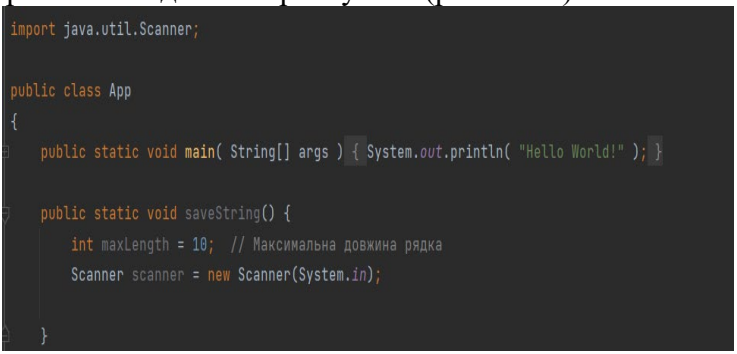
- Створіть метод, який приймає рядок та максимально допустиму довжину.
- Якщо довжина рядка перевищує задане значення, обріжте його до максимально допустимої довжини.
- Поверніть обрізаний рядок як результат функції.

Алгоритм вирішення

Крок 1. Насамперед варто встановити IDE, що вам подобається та підходить для написання програм мовою Java. Відповідно до IDE варто створити проєкт, у рамках роботи стандартного буде достатньо, також можливе використання проєктів на основі архетипу Maven, в якому буде виконуватися завдання.

Крок 2. Розглянемо першу частину розглянутого завдання. Створимо метод saveString(), який не повертає значення і не має аргументів. У ньому створимо локальну змінну цілочислового типу, що характеризуватиме

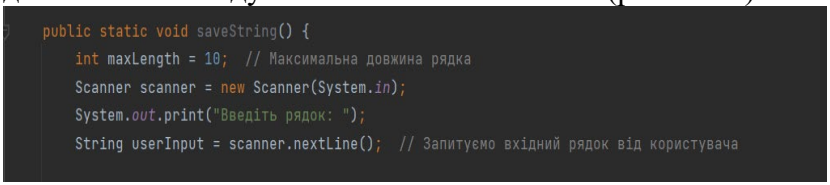
максимальна довжина рядка maxLength (у цьому разі – 10). Та створюємо об’єкт класу Scanner (не забуваючи імпортувати необхідну бібліотеку до проєкту) для одержання введення користувача (рис. 3.1.1).



```
import java.util.Scanner;

public class App
{
    public static void main( String[] args ) { System.out.println( "Hello World!" ); }

    public static void saveString() {
        int maxLength = 10;  // Максимальна довжина рядка
        Scanner scanner = new Scanner(System.in);

    }
```

Рисунок 3.1.1 – Створення методу saveString та створення в ньому потрібних локальних змінних

Крок 3. Створюємо примітивний інтерфейс для взаємодії з користувачем, що складається з виведення на екран за допомогою System.out.print з проханням ввести необхідні дані та зчитування до новоствореної змінної рядкового типу userInput внесених користувачем даних за допомогою методу nextLine об’єкта scanner (рис. 3.1.2).



```
public static void saveString() {
    int maxLength = 10;  // Максимальна довжина рядка
    Scanner scanner = new Scanner(System.in);
    System.out.print("Введіть рядок: ");
    String userInput = scanner.nextLine();  // Запитуємо вхідний рядок від користувача
```

Рисунок 3.1.2 – Додавання до методу saveString мінімалістичного інтерфейсу

Крок 4. Реалізуємо оброблення зчитаних даних відповідно до умов завдання. Перевіряємо довжину введеного рядка за допомогою логічного оператора if/else, методу length() та змінної maxLength. Якщо довжина рядка перевищує maxLength, виводиться повідомлення про

помилку. Інакше, створюється масив рядків myArray розміром 1 та введений рядок зберігається в першому елементі масиву (рис. 3.1.3).

```
public static void saveString() {
    int maxLength = 10;  // Максимальна довжина рядка
    Scanner scanner = new Scanner(System.in);
    System.out.print("Введіть рядок: ");
    String userInput = scanner.nextLine();  // Запитуємо вхідний рядок від користувача

    if (userInput.length() > maxLength) {
        System.out.println("Помилка: рядок занадто довгий!");  // Виводимо повідомлення про помилку
    } else {
        String[] myArray = new String[1];  // Створюємо масив рядків розміром 1
        myArray[0] = userInput;  // Зберігаємо введений рядок у масиві
        System.out.println("Рядок збережено у масиві.");
    }
}

}
```

Рисунок 3.1.3 – До методу saveString додано оброблення введених даних відповідно до умови завдання

Крок 5. Викликаємо створену функцію в main для перевірки її працездатності (рис. 3.1.4).

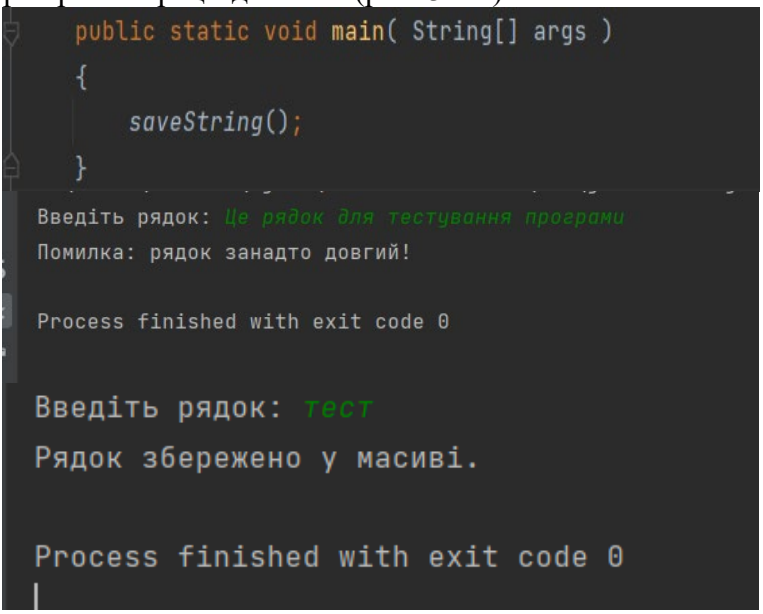


Рисунок 3.1.4 – Додаємо створену функцію в main та тестуємо

Крок 6. Перейдемо до реалізації 2, запропонованої в завдані функції. Для цього створюється метод truncateString(), який приймає рядок (string) і максимально допустиму довжину (maxLength) та повертає обрізаний рядок (рис. 3.1.5).

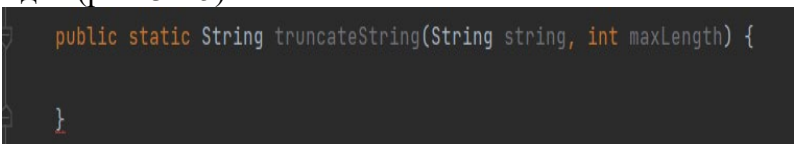


Рисунок 3.1.5 – Створення методу truncateString

Крок 7. За аналогією до кроку 3 створимо примітивний інтерфейс програми, використовуючи об’єкт класу Scanner для одержання введення користувача.

Введений рядок від користувача отримується за допомогою методу nextLine(). Установлюється максимальна допустима довжина рядка maxLength (у цьому разі – 5) (рис. 3.1.6).

```
Scanner scanner = new Scanner(System.in);
System.out.print("Введіть рядок: ");
String userInput = scanner.nextLine();  // Запитуємо вхідний рядок від користувача
int maxLength = 5;  // Максимальна допустима довжина рядка
```

Рисунок 3.1.6 – Додає примітивний інтерфейс для використання зчитаних даних у truncateString

Крок 8. У методі truncateString реалізуємо оброблення вхідних даних відповідно до другої частини завдання за допомогою логічного оператора if/else. Якщо довжина рядка string перевищує maxLength, використовується метод substring() для обрізання рядка до максимально допустимої довжини. Якщо довжина рядка не перевищує maxLength, повертається вихідний рядок string (рис. 3.1.7).

```
public static String truncateString(String string, int maxLength) {
    if (string.length() > maxLength) {
        return string.substring(0, maxLength);  // Обрізаємо рядок до максимально допустимої довжини
    } else {
        return string;
    }
}
```

Рисунок 3.1.7 – До методу truncateString додано оброблення введених даних відповідно до умови завдання

Крок 9. Викликаємо створену функцію в main, для перевірки її працездатності з аргументами userInput і maxLength для обрізання рядка (рис. 3.1.8).

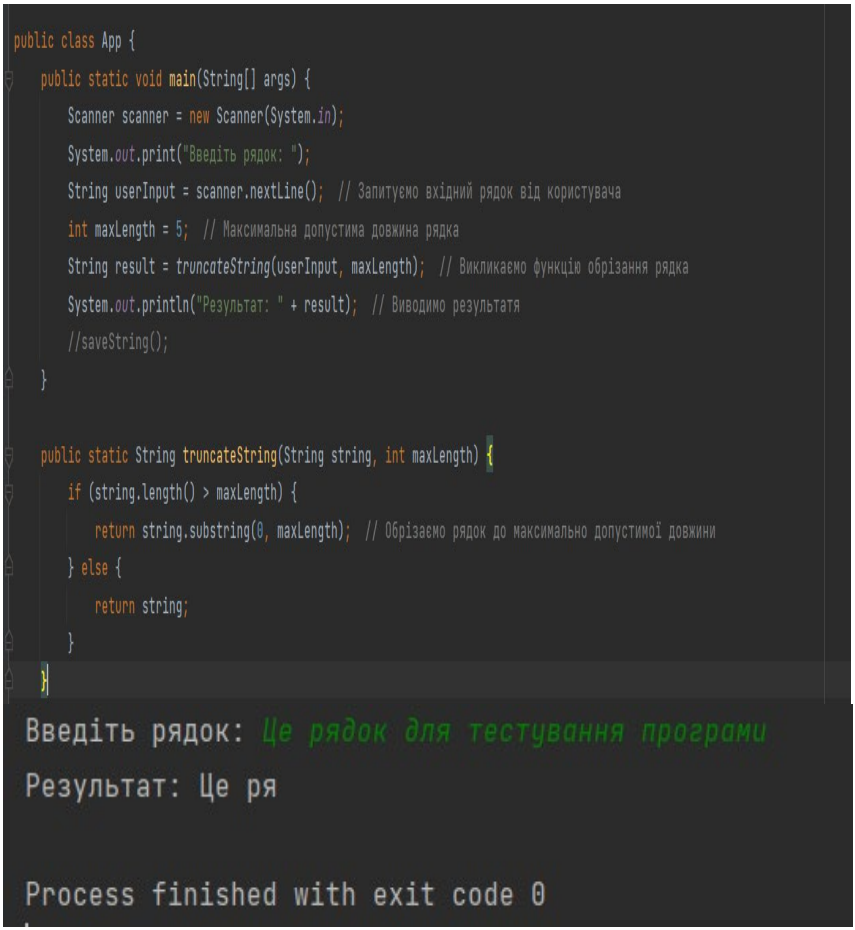


Рисунок 3.1.8 – Викликаємо створену функцію в main та тестуємо

3.2. Завдання для самостійного виконання

Пропонуємо тренувальні вправи з безпечного програмування для розробників програмного забезпечення базового рівня. Вправи допоможуть покращити навички у захисті програмного забезпечення від злому та інших безпекових загроз.

3.2.1. Завдання з перевіркою введених даних

1. Створіть програму, яка перевіряє коректність введеного паролю. Пароль повинен задовольняти такі критерії:

- Мінімальна довжина пароля повинна бути не менше ніж 8 символів.
- Пароль повинен містити принаймні одну велику літеру, одну малу літеру, одну цифру та один спеціальний символ.

2. Напишіть функцію, яка перевіряє коректність введеного імені користувача. Перевірте такі критерії:

- Довжина імені повинна бути в межах від 3 до 20 символів.
- Ім’я користувача може містити лише літери (як великі, так і малі), цифри, символи "_" і "-". Пробіли не допускаються.

3.2.2. Завдання з обробленням винятків

1. Напишіть програму, яка зчитує дані з файлу та обробляє виняткові ситуації, такі як:

- Перевірте наявність файлу перед спробою його зчитування. Якщо файл відсутній, викиньте виняток FileNotFoundException і виведіть відповідне повідомлення.
- Обробіть виняток IOException, якщо виникли проблеми зі зчитуванням файлу, і виведіть відповідне повідомлення про помилку.

2. Створіть клас, який представляє просту арифметичну операцію (наприклад, додавання) і містить метод для її виконання. Обробіть виняток, якщо передані значення аргументів не відповідають очікуваним типам даних. Наприклад, якщо введені значення не є числами, викиньте виняток NumberFormatException і виведіть відповідне повідомлення про помилку.

3.2.3. Завдання із захисту від SQL-ін'єкцій

1. Створіть програму, яка виконує запити до бази даних із використанням підготовлених виразів замість конкатенації рядків. Переконайтеся, що ваша програма відповідно обробляє введені дані та захищає їх від SQL-ін'єкцій.

- Створіть базу даних та таблицю, до якої будуть виконуватися запити.
- Реалізуйте методи для додавання, оновлення та видалення даних із бази даних, використовуючи підготовлені вирази.
- Напишіть тестову програму, яка додає дані до бази даних, використовуючи введені користувачем значення. Переконайтеся, що ваша програма належним чином обробляє введені дані та запобігає SQL-ін'єкціям.

2. Розробіть функцію, яка здійснює пошук даних у базі даних із використанням параметризованих запитів. Перевірте, чи коректно обробляються спроби впровадження SQL-ін'єкцій.

- Створіть метод для виконання параметризованого запиту до бази даних.
- Реалізуйте функцію пошуку, яка приймає введені користувачем параметри та виконує запит до бази даних, використовуючи параметризований запит.
- Напишіть тестовий код, що демонструє виконання пошукового запиту з використанням параметрів та

переконайтеся, що ваша програма відповідно обробляє дані та захищається від SQL-ін'єкцій.

3.2.4. Завдання з аутентифікацією та авторизацією

1. Реалізуйте систему аутентифікації для додатка. Створіть клас користувача з полями для імені, пароля та ролі. Напишіть функції для перевірки введеного пароля та перевірки прав доступу користувача до певних ресурсів.

- Створіть клас «Користувач» із полями для імені, пароля та ролі.
- Реалізуйте метод для перевірки коректності введеного пароля користувачем.
- Реалізуйте метод для перевірки прав доступу користувача до певних ресурсів.

2. Створіть програму, яка дозволяє реєструвати нових користувачів з унікальним іменем та паролем. Переконайтеся, що паролі зберігаються в безпечному форматі, наприклад із хешуванням.

- Реалізуйте метод для реєстрації нових користувачів.
- Перевірте, що ім'я користувача є унікальним, перш ніж додати його до системи.
- Зберігайте паролі в безпечному форматі, наприклад, хешуючи їх перед збереженням у базі даних.

Ці завдання допоможуть студентам поглибити свої знання та навички в безпечному програмуванні додатків мовою Java.

3.2.5. Завдання з використанням Spring Security

1. Створіть сторінку вітального повідомлення, до якої мають доступ усі користувачі.

- Створіть контролер для оброблення запиту на цю сторінку.
- Налаштуйте маршрут (URL) до цієї сторінки у конфігурації Spring MVC.

2. Створіть сторінку зі списком продуктів, до якої мають доступ лише аутентифіковані користувачі.

- Створіть контролер для оброблення запиту на цю сторінку.
- Додайте анотацію @PreAuthorize до методу контролера для обмеження доступу лише для аутентифікованих користувачів.

3. Реалізуйте форму авторизації з полями для введення імені користувача та пароля.

- Створіть HTML-сторінку з формою авторизації.
- Створіть контролер для оброблення запитів на аутентифікацію.

4. Налаштуйте Spring Security для оброблення авторизації і перенаправлення на відповідні сторінки після авторизації або відмови в доступі.

- Додайте залежність на Spring Security у вашому проєкті.
- Налаштуйте конфігурацію Spring Security для оброблення аутентифікації та авторизації.
- Використайте UserDetailsService або AuthenticationProvider для проведення аутентифікації користувачів.
- Налаштуйте перенаправлення після успішної аутентифікації та відмови в доступі.

5. Додайте можливість виходу (логауту) користувача з додатка.

- Створіть контролер для оброблення запитів на логаут.
- Додайте посилання або кнопку, за допомогою яких користувач може вийти з додатка.

Це завдання дозволить студентам крок за кроком реалізувати основні функціональність Spring Security: обмеження доступу до сторінок, аутентифікацію та авторизацію користувачів, а також логаут.

3.2.6. Завдання з використанням фреймворку Bouncy Castle

1. Підключіть фреймворк Bouncy Castle до свого проєкту. Додайте необхідні залежності та налаштування для коректної роботи фреймворку.

2. Розробіть функцію, що генерує випадковий пароль із використанням функцій Bouncy Castle.

- Використовуйте генератор випадкових чисел із фреймворку Bouncy Castle для створення випадкових значень.
- Визначте параметри паролю, такі як мінімальна та максимальна довжина, набір допустимих символів і т. д.
- Згенеруйте випадковий пароль, використовуючи зазначені параметри.

3. Реалізуйте відображення згенерованого пароля на екрані.

- Виведіть згенерований пароль у зручному для користувача форматі.
- Забезпечте можливість повторного генерування пароля за запитом користувача.

4. Забезпечте безпеку згенерованого пароля.

- Рекомендуйте користувачеві добре зберігати та не розголошувати пароль.
- Використовуйте методи та рекомендації з фреймворку Bouncy Castle для забезпечення безпеки паролю, наприклад хешування або шифрування.

Це завдання дозволить студентам ознайомитися з використанням фреймворку Bouncy Castle для генерації безпечних випадкових паролів.

3.2.7. Завдання з використанням фреймворку OWASP Java Encoder

1. Підключіть фреймворк OWASP Java Encoder до свого проєкту. Додайте необхідні залежності та налаштування для коректної роботи фреймворку.

2. Розробіть функцію, яка приймає рядок введених даних від користувача та застосовує функції з фреймворку OWASP Java Encoder для екранування даних.

- Використовуйте функцію Encode.forHtml для екранування рядка перед виведенням його на вебсторінку.
- Переконайтеся, що всі HTML-спеціальні символи у введеному рядку замінені на їх еквіваленти, що запобігає виконанню небезпечного скрипту в браузері.

3. Реалізуйте можливість введення даних користувача та відображення їх на екрані.

- Запросіть від користувача рядок даних, який потрібно екранувати.
- Використайте функції з фреймворку OWASP Java Encoder для екранування введених даних перед виведенням їх на екран.
- Відобразіть екранований рядок на екрані для перевірки результату.

4. Забезпечте безпеку оброблення даних користувача.

- Поясніть користувачеві, що екранування даних є важливим кроком для запобігання XSS або HTML-ін’єкціям.
- Наголосіть на важливості використання функцій із фреймворку OWASP Java Encoder для безпечного виведення даних на вебсторінках.
- Порадьте користувачеві завжди екранувати дані перед їх виведенням на стороні клієнта.

Це просте завдання дозволить студентам ознайомитися з використанням фреймворку OWASP Java

Encoder для безпечного оброблення та виведення даних користувача, запобігаючи потенційним атакам XSS та HTML-ін'єкцій.

3.2.8. Завдання з використанням фреймворку Apache Shiro

1. Підключення Apache Shiro:

- Додайте залежність Apache Shiro до вашого проєкту (наприклад, додавши відповідний репозиторій та версію залежності у файл pom.xml для Maven проєкту).
- Налаштуйте конфігураційний файл Shiro (shiro.ini або shiro.xml) для визначення налаштувань аутентифікації та авторизації.

2. Створення класу користувача (User):

- Створіть клас User з полями, такими як username (ім'я користувача) і password (пароль користувача).
- Реалізуйте методи доступу (getters / setters) до цих полів.

3. Аутентифікація користувача:

- Запитайте в користувача його ім'я та пароль.
- Створіть об'єкт UsernamePasswordToken з уведеними даними.
- Створіть об'єкт SecurityManager та налаштуйте його з використанням конфігураційного файлу Shiro.
- Викличте метод login на об'єкті SecurityManager, передаючи UsernamePasswordToken.
- Перевірте, чи пройшла аутентифікація успішно (з використанням методу isAuthenticated()).
- Виведіть повідомлення про успішну або невдалу аутентифікацію користувача.

4. Перевірка роботи програми:

- Створіть декілька об'єктів User із різними іменами та паролями.
- Викличте функцію аутентифікації для кожного створеного об'єкта User.

- Виведіть повідомлення про успішну або невдалу аутентифікацію для кожного користувача.

Це завдання дозволить студентам поглибитись у використання фреймворку Apache Shiro для аутентифікації користувачів у своїх додатках мовою Java.

СПИСОК ЛІТЕРАТУРИ

1. Java security (java security). документация – unix.org.ua. URL: https://docstore.mik.ua/orelly/java-ent/security/index.htm (data of access: 07.06.2023).

2. Каплун В. А., Дмитришин О. В., Баришев Ю. В. Захист програмного забезпечення. Частина 2 : навчальний посібник. Вінниця : ВНТУ, 2014. 105 с.

3. Горбенко І. Д., Олешко О. І., Герцог А. М. Безпека програмного забезпечення та безпечне програмування. *Прикладная радиоэлектроника* : науч.-техн. журн. Харьков : ХНУРЭ, 2011. Т. 10, № 2. С. 244–248.

4. Якименко І. З. Опорний конспект лекцій із дисципліни «Безпека програм та даних» для студентів спеціальності «Кібербезпека». Тернопіль, 2019. 50 с.

5. Seacord R. Secure coding in C and C++. Pearson Education, Limited.

6. Stuttard D. The web application hacker’s handbook. New York : John Wiley & Sons, Ltd., 2008.

7. Dowd M., Schuh J., McDonald J. Art of software security assessment: identifying and preventing software vulnerabilities. Pearson Education, Limited.

8. OWASP foundation, the open source foundation for application security / OWASP foundation. OWASP Foundation, the Open Source Foundation for Application Security / OWASP Foundation. URL: https://owasp.org/ (data of access: 11.07.2023).

9. National institute of standards and technology. NIST. URL: https://www.nist.gov/ (data of access: 11.07.2023).

10. Security week home. SecurityWeek. URL: https://www.securityweek.com/ (data of access: 11.07.2023).

11. Home. Security Boulevard. URL: https://securityboulevard.com/ (data of access: 11.07.2023).

12. Де використовується Java і чому він такий популярний?. SpaceLab. URL: https://spacelab.ua/articles/gde-ispolzuetsa-java-i-pochemu-on-takoy-populyarniy/ (дата звернення: 07.06.2023).

13. Правила безпечного кодування та найкращі практики для розробників – Інший. *Огляди, Ігри, Розваги, Червень 2023*. URL: https://uk.myservername.com/secure-coding-guidelines (дата звернення: 07.06.2023).

14. Carr G. Що таке вразливість програм та вразливість нульового дня в комп'ютерах? – Чайові 2023. *Begin IT*. URL: https://uk.begin-it.com/4790-what-is-vulnerability-in-computer-security (дата звернення: 07.06.2023).

15. Log4Shell, критична вразливість у Apache Log4j 2, яка впливає на багато проектів Java. *Desde Linux*. URL: https://blog.desdelinux.net/uk/log4shell-–-критична-вразливість-у-apache-log4j-2,-яка-впливає-на-багато-проектів-Java/ (дата звернення: 07.06.2023).

16. What do weblogic, websphere, jboss, jenkins, opennms, and your application have in common? This vulnerability. URL: https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/#thevulnerability (data of access: 07.06.2023).

17. Pentestit. Spring4Shell RCE – критическая уязвимость в java spring framework. *Хабр*. URL: https://habr.com/ru/companies/pentestit/articles/658421/ (data of access: 07.06.2023).

18. Spring framework RCE, early announcement. *Spring Framework RCE, Early Announcement*. URL: https://spring.io/blog/2022/03/31/spring-framework-rce-early-announcement (data of access: 07.06.2023).

19. A vulnerability in apache struts could allow for remote code execution. *CIS*. URL: https://www.cisecurity.org/

advisory/a-vulnerability-in-apache-struts-could-allow-for-remote-code-execution_2022-056 (data of access: 07.06.2023).

20. Spring security. *Spring Security*. URL: https://spring.io/projects/spring-security#overview (date of access: 07.06.2023).

21. Introduction to BouncyCastle with Java | Baeldung. *Baeldung*. URL: https://www.baeldung.com/java-bouncy-castle (data of access: 07.06.2023).

22. Apache shiro / simple. java. security. *Apache Shiro*. URL: https://shiro.apache.org/ (data of access: 07.06.2023).

23. OWASP java encoder / OWASP foundation. *OWASP Foundation, the Open Source Foundation for Application Security / OWASP Foundation*. URL: https://owasp.org/www-project-java-encoder/ (data of access: 07.06.2023).

24. Securing applications and services guide. *Keycloak*. URL: https://www.keycloak.org/docs/latest/securing_apps/ (data of access: 07.06.2023).

25. Oaks S. Java security. O'Reilly Media, Incorporated, 2001.

Електронне навчальне видання

Любчак Володимир Олександрович,
Савостян Віталій Вікторович,
Козолуп Павло Дмитрович

БЕЗПЕКА JAVA-ДОДАТКІВ

Навчальний посібник

Редактор Н. З. Клочко
Комп’ютерне верстання І. О. Пугача

Формат 60×84/16. Ум. друк. арк. 4,19. Обл.-вид. арк. 4,03.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб’єкта видавничої справи ДК № 3062 від 17.12.2007.