

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

«На правах рукопису»

УДК 004.056

«До захисту допущено»

В.о. завідувача кафедри

_____ М.В.Грайворонський
“ ____ ” _____ 2019 р.

Магістерська дисертація
на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: Метод автоматичного виявлення помилок безпеки в програмному забезпеченні на основі глибинного навчання

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

Черноусов Артем Вікторович
(прізвище, ім'я, по батькові)

(підпис)

Науковий керівник к.т.н., доц. Стьопочкина Ірина Валеріївна _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2019 року

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

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (освітньо-професійна програма) – 125 Кібербезпека («Системи, технології та математичні методи кібербезпеки»)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)

«__» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Черноусов Артем Вікторович
(прізвище, ім'я, по батькові)

1. Тема дисертації: Метод автоматичного виявлення помилок безпеки в програмному забезпеченні на основі глибинного навчання

науковий керівник дисертації к.т.н., доц. Стьопочкіна Ірина Валеріївна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15» листопада 2019 р. № 3927-с

2. Термін подання студентом дисертації 10.12.2019 р.

3. Об'єкт дослідження _____

4. Вихідні дані _____

5. Перелік завдань, які потрібно розробити _____

6. Орієнтовний перелік ілюстративного матеріалу _____

7. Орієнтовний перелік публікацій _____

8. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент

(підпис)

(ініціали, прізвище)

Науковий керівник дисертації

(підпис)

(ініціали, прізвище)

РЕФЕРАТ

Обсяг роботи 85 сторінок, 18 ілюстрацій, 29 таблиць та 54 джерел літератури.

Об'єктом дослідження є процес розробки програмного забезпечення.

Предметом дослідження є методи виявлення помилок безпеки початкового коду програмного забезпечення.

Методи дослідження - представлення початкового коду у вигляді АСД, глибинні методи навчання для розв'язання проблем класифікації.

Наукова новизна полягає в тому, що метод детектування помилок безпеки на основі глибинного навчання отримав подальший розвиток шляхом використання представлення коду у вигляді абстрактного синтаксичного дерева, покращення процедури представлення код-гаджета у числовий вектор та покращений підбір параметрів топології нейронної мережі, що призвело до поліпшення точності знаходження помилок безпеки до 91%. Цей підхід є вдосконаленням технології, описаної раніше в статті [1].

Мета дослідження - дослідження та удосконалення методів та алгоритмів пошуку вразливих код-гаджетів.

Результати роботи викладені у третьому розділі, що демонструють систему знаходження помилок безпеки на основі глибинного навчання.

Результати роботи можуть бути використані для аналізу початкового коду на наявність помилок безпеки.

Ключові слова: АСД, статичний аналіз, початковий код, помилка безпеки.

ABSTRACT

Volume of work is 85 pages, 18 illustrations, 29 tables and 54 literature sources.

The object of the research is the software development process.

The subject of the study is the detection of security errors of the source code.

Research methods - presentation of the source code in the form of AST, neural network teaching methods for solving classification problems.

The purpose of the study is to investigate and improve the methods and algorithms for finding vulnerable code gadgets

The scientific novelty is that the method of detecting security errors based on deep learning has been further developed by using code representation in the form of an abstract syntax tree, improving the procedure for presenting the code gadget in a numerical vector and improving the selection of neural network topology parameters, which led to improved f1 score finding security errors up to 91%. This approach is an improvement on the technology described earlier in [1].

The results of the work are presented in the third section, which demonstrates the system of finding security errors based on deep learning.

The results can be used to analyze the source code for security errors.

Key words: AST, static analysis, source code, security error.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз існуючих систем для виявлення помилок безпеки	11
1.1 Поняття “помилка безпеки”	11
1.2 Існуючі методи виявлення помилок безпеки	20
1.3 Порівняння методів захисту	24
Висновки до розділу 1	25
2 Метод виявлення помилок безпеки в початковому коді.....	26
2.1 Постановка задачі	26
2.2 Загальна схема методу та основні принципи використання глибинного навчання	27
2.3 Метод перетворення моделі коду у вектор вхідних параметрів нейронної мережі.....	34
2.4 Архітектура нейронної мережі	35
2.5 Метод визначення помилок безпеки в вихідному коді.....	37
Висновки до розділу 2	39
3 Система виявлення помилок безпеки на основі глибинного навчання.....	41
3.1 Класифікація коду на основі нейронної мережі	41
3.2 Програмна реалізація.....	43
3.3 Навчання моделі.....	46
3.4 Результати дослідження	52
3.5 Обмеження підходу	54
Висновки до розділу 3	56

4 Розробка стартап-проекту	57
4.1 Опис ідеї проекту	57
4.2 Технологічний аудит ідеї проекту.....	59
4.3 Аналіз ринкових можливостей запуску стартап-проекту.....	61
4.4 Розроблення ринкової стратегії проекту	71
4.5 Розроблення маркетингової програми стартап-проекту.....	75
Висновки до розділу 4	79
Висновки	80
Перелік джерел посилання	81

ПЕРЕЛІК УМОВНИК ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

NN – neural network

DL – deep learning

SDL – Security Development Lifecycle

AST – Abstract Syntax Tree

CWE – Common Weakness Enumeration

CVE – Common Vulnerability Enumeration

TP – True positive

TN – True negative

FP – False positive

FN – False negative

FPR – False positive rate

FNR – False negative rate

UML – Unified Modeling Language

LSTM – Long Short-Term Memory

BLSTM – Bidirectional Long Short-Term Memory

ВСТУП

На сьогодні ведеться інтенсивна розробка програмного забезпечення, яке допомагає запобігати появі помилок безпеки в програмному забезпеченні. Великі компанії запроваджують процедуру Microsoft SDL [2], яка використовується для зменшення ризиків безпеки. Оскільки, найбільш ефективним етапом є проведення перевірок на етапі розробки програмного забезпечення, тому, в даній роботі ми пропонуємо продукт для виявлення помилок безпеки в програмному забезпеченні на основі глибинного навчання, який можливо використовувати на етапі розробки програмного забезпечення.

Найбільш відомими методами виявлення помилок безпеки програмного коду є статичні аналізатори. Основним недоліком статичних аналізаторів є те, що для кожної помилки безпеки експерт повинен запрограмувати правило або набір правил, які зможуть знаходити помилки безпеки в програмному забезпеченні, що є дуже ресурсномістким процесом. Водночас, існують бази даних, які містять як приклад коду з дефектами, так і вже виправленого. На підставі цього, можна скористатися перевагами глибинного навчання для автоматичної побудови правил аналізу початкового коду. Основною проблемою застосування глибинного навчання для аналізу початкового коду — є представлення коду, в такому вигляді, який би дозволив узагальнити варіанти реалізації специфічних функцій з урахуванням контексту використання. Внесок цієї роботи полягає у наданні методу подальшого розвитку шляхом використання іншого методу представлення коду, вдосконалення технології вилучення фрагментів коду (код-гаджетів), вдосконаленої процедури представлення код-гаджету у числовий вектор та вдосконалений підбір топології нейронної мережі. [1]

Об'єктом дослідження є процес розробки програмного забезпечення.

Предметом дослідження є методи виявлення помилок безпеки початкового коду програмного забезпечення.

Методи дослідження - представлення початкового коду у вигляді АСД, переведення тексту до числового вектору зі збереженням контексту та глибинні методи навчання для розв'язання проблем класифікації.

Мета дослідження - дослідження та удосконалення методів та алгоритмів пошуку вразливих код-гаджетів

Наукова новизна полягає в тому, що метод детектування помилок безпеки на основі глибинного навчання отримав подальший розвиток шляхом використання представлення коду у вигляді абстрактного синтаксичного дерева, покращення процедури представлення код-гаджета у числовий вектор та покращений підбір параметрів топології нейронної мережі, що призвело до поліпшення точності знаходження помилок безпеки до 91%. Цей підхід є вдосконаленням технології, описаної раніше в статті [1].

Практичне значення роботи полягає в готовому продукті VulDetect, який може бути з легкістю впроваджений в будь-яку систему.

Апробація результатів роботи – робота була оприлюднена на XVII Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики» (Україна, м. Київ, 25-26 квітня 2019 р.). Тема доповіді: “Методи виявлення помилок безпеки в програмному забезпеченні на основі глибинного навчання”. Матеріали конференції. – Київ : КПІ ім. Ігоря Сікорського, 2019. – С. 187–190. – Бібліогр.: 17 назв.

Публікації – робота була опублікована в науковому журналі «THEORETICAL AND APPLIED CYBERSECURITY», з темою “Deep learning based automatic software defects detection framework”. Vol 1, No 1 – pp. 68 - 74

1 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ДЛЯ ВИЯВЛЕННЯ ПОМИЛОК БЕЗПЕКИ

1.1 Поняття “помилка безпеки”

Безпека програмного забезпечення в широкому розумінні є властивістю певного програмного забезпечення функціонувати без різних негативних наслідків для конкретної комп'ютерної системи. Причини, що приводять до порушення безпеки, можуть бути різними: збої комп'ютерних систем, помилки програмістів і операторів, дефекти в програмах. При цьому дефекти прийнято розглядати двох типів: навмисні і ненавмисні. Перші є, як правило, результатом зловмисних дій, другі — помилкових дій людини.

Вразливе місце — це недолік програмного забезпечення, яким може скористатися зловмисник в своїх корисливих цілях. Вразливі місця в програмному забезпеченні, пов'язані з системною безпекою, варіюються від помилок в локальній реалізації і помилок між процедурних взаємозв'язків до недоліків більш високого рівня, допущених на стадії проектування.

Помилка — це похибка в програмному забезпеченні. Помилки програмного коду — це недоліки на рівні реалізації. Для виявлення помилок використовуються програми сканування коду. [3]

1.1.1 Класифікація програмних помилок

Виявити помилки в програмному забезпеченні, що впливають на правильне виконання завдання доволі важко, а виявити помилки в системі безпеки ще важче.

Ще складніше ситуація з вразливими місцями, внесеними на рівні проекту. Для виявлення помилки на рівні проекту програми потрібен величезний досвід. Тому дуже складно знайти такі помилки і ще складніше автоматизувати процес цього пошуку. Помилки на рівні проектування найбільш поширені, проте їм приділяється найменша увага при оцінюванні безпеки програмного коду.

Вразливості захисту можуть приводити до того, що одна програма може використовуватися для подолання обмежень захисту всієї системи в цілому.

Вразливості захисту залежно від програмних помилок можна класифікувати так [3]:

- Переповнювання буфера. Ця вразливість виникає через відсутність контролю за виходом за межі масиву пам'яті під час виконання програми. Коли при записуванні дуже великого пакету даних переповнюється буфер обмеженого розміру, вміст сторонніх елементів пам'яті перезаписується, і відбувається збій та аварійний вихід з програми. За місцем розташування буфера в пам'яті процесу розрізняють переповнювання буфера в стеку, купі і ділянки статичних даних.
- Вразливості "зіпсованого введення". Ці вразливості можуть виникати у випадках, коли дані, що вводяться користувачем, без достатнього контролю передаються на інтерпретатор деякої зовнішньої команди. В цьому випадку вхідні дані можуть бути задані так, що за запуском інтерпретатором виконається зовсім не та команда, яка передбачалася авторами вразливої програми.

- Вразливості рядків форматів. Даний тип вразливостей захисту є підкласом вразливості "зіпсованого введення". Він виникає внаслідок недостатнього контролю параметрів при використанні функцій введення-виведення стандартної бібліотеки мови Cі. Ці функції приймають в якості одного з параметрів символьний рядок, де задається формат введення або виведення подальших аргументів функції. Якщо користувач сам може задати вид форматування, то ця вразливість може виникнути в результаті невдалого використання функцій форматування рядків.
- Вразливості як наслідок помилок синхронізації. Проблеми, пов'язані з багатозадачністю, приводять до ситуацій, що називаються "стан гонки": в програмі, не розрахованій на виконання в багатозадачному середовищі, може бути не передбачено, що, наприклад, використовувані файли можуть бути змінені при виконанні іншої програми. Як наслідок, зловмисник, що вчасно підміняє вміст цих робочих файлів, може нав'язати виконання певних дій.

1.1.2 Common Weakness Enumeration

Common Weakness Enumeration (CWE) [4] - це формальний список або словник загальних недоліків програмного забезпечення, які можуть виникати в архітектурі, дизайні, коді або реалізації програмного забезпечення, що може призвести до уразливих зразків безпеки. CWE був створений для того, щоб слугувати спільною мовою для опису слабкості безпеки програмного забезпечення; слугують стандартною вимірювальною ручкою для програмних засобів безпеки, орієнтованих на ці недоліки; і забезпечити загальний базовий стандарт для виявлення слабкості, пом'якшення та профілактики.

Слабкі місця програмного забезпечення - це недоліки, помилки, помилки, уразливості та інші помилки в реалізації програмного забезпечення, коді, дизайні або архітектурі, які можуть призвести до того, що системи та мережі стануть вразливими до атак. Приклади слабкості програмного забезпечення включають: переповнення буфера, рядки формату тощо; проблеми структури та обґрунтованості; спільні маніпуляції спеціальними елементами; помилки каналів і шляхів; помилки обробника; помилки інтерфейсу користувача; помилки обходу шляху та еквівалентності; помилки автентифікації; помилки управління ресурсами; недостатня верифікація даних; кодова оцінка та ін'єкція; і випадковість і передбачуваність.

Прикладом може бути CWE-121, в якому зібрані вразливості Stack-based Buffer Overflow Weaknesses.

1.1.3 CVE

Common Vulnerabilities and Exposures (CVE) [5] - це список загальних ідентифікаторів загальновідомих вразливостей кібербезпеки.

Відмінність від CWE [4] полягає в тому, що CVE є більш вузьким, оскільки кожна CVE є реалізацією одного або декількох CWE.

1.1.4 Помилки на рівні проектування

Помилки, які виникають на етапі проектування — найнебезпечніші, та найбільш дорогі в плані виправлення, оскільки, архітектура є “скелетом” усього, тому при виправлення помилки в основному модулі тягнуть зміни

логіки в усіх успадкованих модулях. Тому, одним з прикладів правильного проектування та розробки застосунків було запропоновано компанією Microsoft, а саме Microsoft Security Development Lifecycle (SDL) [2].

1.1.5 Microsoft SDL

Security Development Lifecycle [2] - концепція розробки, яка полягає у формуванні вимог до додатка, безпечному програмуванні, тестуванні, сертифікації, експлуатації та оновленні. SDL - це процес, який дозволяє підтримувати необхідний рівень безпеки системи на етапі розробки, а потім протягом усього терміну експлуатації. Ця концепція фокусується на забезпеченні безпеки під час розробки застосунку, ідентифікації ризиків та керування ними.

Згідно до цієї концепції існують наступні поради, щодо проектування та розробки програмного забезпечення, а саме:

1. Підготовка та навчання

Навчання має на увазі дослідження підготовленості співробітників організації за темами безпеки і захисту даних. При необхідності рекомендується створити курси, розробити відповідні критерії якості навчання, визначити частоту тренінгів та забезпечити їх відвідування мінімально необхідному для підтримки безпеки кількості персоналу.

Базовий рівень тренінгу повинен включати в себе:

1. Безпечний дизайн;
2. Моделювання загроз;
3. Безпечне кодування;
4. Тестування безпеки;
5. Забезпечення приватності.

2. Визначення вимог безпеки до ПЗ

Очевидно, що оптимальним часом для визначення вимог безпеки є початкові етапи проектування та планування, тому, на цьому етапі команда розробки визначає лідерів і консультантів по темам безпеки, призначає відповідального за безпеку. Відповідальний перевіряє план розробки продукту, рекомендує зміни або встановлює додаткові вимоги до безпеки продукту, визначає пріоритет, процедуру відстеження та виправлення помилок.

3. Визначення звітності про показники та відповідність

Дуже важливо визначити мінімально прийнятні рівні якості безпеки та залучити інженерні групи до відповідальності за виконання цих критеріїв. Визначення цих раних допомагає групі розуміти ризики, пов'язані з проблемами безпеки, виявляти та виправляти дефекти безпеки під час розробки, а також застосовувати стандарти протягом всього проекту. Встановлення значущого рядка помилок включає чітке визначення порогів серйозності вразливостей безпеки (наприклад, всі відомі вразливості, виявлені з "критичним" або "важливим" рейтингом серйозності, повинні бути виправлені з певним часовим інтервалом) і ніколи не розслабляти його після встановлення.

Щоб відстежувати ключові показники ефективності (КПЕ) та забезпечити виконання завдань з безпеки, відстеження помилок та / або механізми відстеження роботи, що використовуються організацією (наприклад, Azure DevOps), повинні дозволяти чітко позначити дефекти безпеки та робочі елементи безпеки. Це дає можливість точного відстеження та звітування про роботу з безпеки.

4. Виконувати моделювання загроз

Моделювання загроз має використовуватися в середовищах, де існує значний ризик для безпеки. Моделювання загроз може застосовуватися на рівні компонента, програми або системи. Це практика, яка дозволяє групам розробників розглядати, документувати і (важливо) обговорювати наслідки для

проектів безпеки в контексті їх планового операційного середовища і структурованим чином.

Застосування структурованого підходу до сценаріїв загроз допомагає команді більш ефективно і менш дорого визначити уразливості безпеки, визначити ризики від цих загроз, а потім зробити вибір функцій безпеки та встановити відповідні пом'якшення.

5. Встановити вимоги до проектування

SDL зазвичай розглядається як діяльність з гарантування, що допомагає інженерам реалізовувати «безпечні функції», оскільки функції добре розроблені з точки зору безпеки. Щоб досягти цього, інженери, як правило, покладаються на функції безпеки, такі як криптографія, автентифікація, журналювання та інші. У багатьох випадках вибір або реалізація функцій безпеки виявився настільки складним, що вибір дизайну або реалізації може призвести до вразливості. Тому надзвичайно важливо, що вони застосовуються послідовно і з послідовним розумінням захисту, які вони надають.

6. Визначення та використання стандартів криптографії

Зі зростанням кількості мобільних і хмарних обчислень, надзвичайно важливо забезпечити захист всіх даних, включаючи інформацію, що стосується безпеки, та дані управління та контролю, від ненавмисного розкриття або зміни, коли вони передаються або зберігаються. Для досягнення цього зазвичай використовується шифрування. Внесення неправильного вибору при використанні будь-якого аспекту криптографії може бути катастрофічним, і краще розробити чіткі стандарти шифрування, які забезпечують специфіку кожного елемента реалізації шифрування. Це слід залишити фахівцям. Хороше загальне правило полягає в тому, щоб використовувати лише бібліотеки шифрування, перевірені промисловістю, і переконатися, що вони реалізовані таким чином, щоб вони могли бути легко замінені у разі потреби.

7. Управляти ризиком з використанням компонентів третьої сторони

Сьогодні переважна більшість програмних проєктів будується з використанням сторонніх компонентів (як комерційних, так і відкритих). При виборі компонентів третіх сторін для використання важливо розуміти вплив, який може мати вразливість системи безпеки на безпеку більшої системи, в яку вони інтегровані. Точна інвентаризація компонентів третьої сторони та план реагування на нові виявлені уразливості дасть змогу значно зменшити цей ризик, але слід враховувати додаткову перевірку, залежно від апетиту ризику вашої організації, типу використовуваного компонента та потенційний вплив уразливості системи безпеки. Дізнайтеся більше про управління ризиками безпеки використання сторонніх компонентів, таких як програмне забезпечення з відкритим кодом.

8. Використовуйте затверджені інструменти

Визначте та опублікуйте список затверджених інструментів та пов'язані з ними перевірки безпеки, такі як опції компілятора / лінкера та попередження. Інженери повинні прагнути до використання останньої версії затверджених інструментів, таких як версії компіляторів, і для того, щоб скористатися новими функціями та захистом аналізу безпеки.

9. Виконати перевірку безпеки статичним аналізатором (SAST)

Аналіз вихідного коду перед компіляцією забезпечує високо масштабований метод перевірки коду безпеки та допомагає забезпечити дотримання безпечних політик кодування. SAST, як правило, інтегрується в конвеєр фіксації для виявлення вразливостей кожного разу, коли програмне забезпечення будується або упаковано. Проте, деякі пропозиції інтегруються в середовище розробників, щоб виявити певні недоліки, такі як існування небезпечних або інших заборонених функцій і замінити тих, які мають більш безпечні альтернативи, оскільки розробник активно кодує. Існує не один розмір підходить всі рішення і команди розробки повинні вирішити оптимальну частоту для виконання SAST і, можливо, розгортання декількох тактик, щоб збалансувати продуктивність з відповідним охопленням безпеки.

10. Виконати перевірку безпеки динамічним аналізатором (DAST)

Виконання перевірки під час виконання повністю скомпільованого або упакованого програмного забезпечення перевіряє функціональність, яка проявляється лише тоді, коли всі компоненти інтегровані і працюють. Як правило, це досягається за допомогою інструменту або набору готових атак або інструментів, які спеціально відстежують поведінку програми для пошкодження пам'яті, проблеми з привілеями користувачів та інші критичні проблеми безпеки. Подібно до SAST, не існує універсального рішення, і хоча деякі інструменти, такі як інструменти для сканування веб-додатків, можуть бути більш легко інтегровані в континент безперервної інтеграції / безперервної доставки, інші тести DAST, такі як fuzzing, потребують іншого підхід.

11. Виконати тестування на проникнення

Тестування на проникнення - це аналіз безпеки програмної системи, що виконується кваліфікованими фахівцями з безпеки, що імітують дії хакера. Мета тесту проникнення полягає в тому, щоб розкрити потенційні уразливості, що виникають внаслідок помилок кодування, помилок конфігурації системи або інших слабких місць розгортання, і як такий тест, як правило, знаходить найширший вибір уразливостей. Тести на проникнення часто виконуються у поєднанні з автоматизованими та ручними переглядами кодів, щоб забезпечити більш високий рівень аналізу, ніж це було б зазвичай.

12. Створити стандартний процес реагування на інциденти

Підготовка плану реагування на інциденти має вирішальне значення для вирішення нових загроз, які можуть виникнути з часом. Вона повинна бути створена за погодженням із спеціальною групою реагування на інциденти з безпекою продуктів (PSIRT) вашої організації. План повинен включати тих, до кого слід звертатися у випадку аварійної ситуації з безпекою, і встановити протокол для обслуговування безпеки, включаючи плани щодо коду, успадкованого від інших груп в організації, і для коду третьої сторони. План реагування на інцидент повинен бути перевірений до того, як це необхідно!

1.2 Існуючі методи виявлення помилок безпеки

Для виявлення вразливостей захисту в програмах застосовують наступні інструментальні засоби:

- Динамічні аналізатори [6]. Інструменти, за допомогою яких можна налагоджувати програми в процесі її виконання.
- Статичні аналізатори (статичні налагоджувачі) [7]. Інструменти, за якими використовуються дані, накопичені в ході статичного аналізу програми.

1.2.1 Динамічні аналізатори

Динамічний аналіз коду [6] — аналіз програмного забезпечення, що виконується за допомогою виконання програм на реальному або віртуальному процесорі (на відміну від статичного аналізу). Утиліти динамічного аналізу можуть вимагати завантаження спеціальних бібліотек, перекомпіляцію програмного коду. Деякі утиліти можуть інструментувати код, що виконується, у процесі виконання або перед ним. Для більшої ефективності динамічного аналізу вимагається подача досліджуваної програмі достатньої кількості вхідних даних, щоб отримати повніше покриття коду. Також потрібно подбати про мінімізацію впливу інструментування на виконання програми, що досліджується (включаючи часові характеристики).

Прикладом таких аналізаторів можуть бути: Valgrind [8], Pin [9], DynamoRIO [10], Daikon [11], і т.д.

1.2.2 Статичні аналізатори

Статичний аналіз коду [7]- аналіз програмного забезпечення, який здійснюють (на відміну від динамічного аналізу [6]) без реального виконання програм, що досліджуються. Зазвичай аналізу піддають початковий код, хоча іноді аналізу піддається об'єктний код, наприклад, Р-код або код CIL. Термін зазвичай застосовують до аналізу, який проводить спеціальне програмне забезпечення (ПЗ), тоді як ручний аналіз називають «program understanding», «program comprehension» (розумінням або осягненням програми).

В даний час існує досить багато програмних застосунків для пошуку проблем безпеки програмного коду. Аналізатори працюють з різним поданням початкового коду: чистий початковий код, абстрактне синтаксичне дерево (АСД) [12] та виконуваний бінарний файл [13][14]. Зі всього списку програмних застосунків, можна виділити наступні статичні аналізатори: з відкритим початковим кодом [15][16], комерційні продукти [17][18] та деякі дослідницькі проекти[14][19]. Відносно принципу роботи статичних аналізаторів можна відмітити наступні 2 підходи:

- На основі правил [20]
- На основі подібності коду[21].

Кожний із підходів, має свої як переваги так і недоліки, які описані більш детально в розділі 1.2.3.

1.2.3 Підхід на основі правил

Підхід на основі правил [20] має два основних недоліки, які полягають в інтенсивній та монотонній роботі та в високих показниках помилок другого роду.

- Інтенсивна та монотонна робота

В даному випадку, завдання побудови правил, за якими будуть виявлятися вразливості конкретної функції покладаються на фахівця з безпеки. Ця задача достатньо стомлююча та суб'єктивна. Іноді виникають помилки через складність реалізованих функцій. Іншими словами, для виявлення ознак, що функція є вразливою необхідно враховувати багато аспектів. У принципі, розв'язання цієї проблеми складається з окремого написання однієї і тієї ж функції кількома експертами, а потім вибрати найбільш ефективну або застосувати комбінацію деяких функцій. Однак це призводить до ще більшої ручної праці. З цього, важливо, позбавити людей від стомлюючої і суб'єктивної задачі ручного визначення функцій, що будуть виявляти вразливості.

- Високі показники помилок другого роду

З іншої точки зору, рішення, що існують часто не враховують багатьох вразливостей. Іншими словами мають багато помилок другого роду. Згідно з результатами статті [22], ці показники для Clang Analyze [16] та CppCheck [15] склали 84% та 92% відповідно. Ці значення можуть бути виправдані акцентом на низький рівень помилок першого роду, але це досі не дуже гарний показник.

1.2.4 Підхід на основі подібності коду

Підхід на основі подібності коду [21] також має дві проблеми: відсутність достатнього набору даних та відсутність ефективного алгоритму.

- Відсутність достатнього набору даних

На даний момент не існує достатнього набору даних для такого роду досліджень. Ця проблема є актуальною, попри те, що National Vulnerability Database (NVD) [23] та NIST Software Assurance Reference (SARD) [24] стали відкритими. Так само в дослідженнях були створені бази даних, де зіставлені ідентифікатори Common Vulnerabilities [5] та Exposures (CVE-ID) з коммітами, де кожен коміт містить різницю початкового коду до виправлення та після. Але всіх цих даних все одно недостатньо для визначення усіх помилок безпеки.

- Відсутність ефективного алгоритму

В даний час не існує єдиного ефективного алгоритму подібності коду, який був би ефективним для всіх типів вразливостей, оскільки кожна вразливість має свої особливості, які слід брати до уваги.

Поліпшений підхід на основі подібності коду був опублікований у статті. Перевагою даного методу, щодо підходу на основі подібності коду полягає у ефективному алгоритмі, заснований на глибинному навчанні. У порівнянні з підходом, що ґрунтується на правилах, він має досить низький рівень помилок другого роду, приблизно 7%. Недоліки цього методу полягають в наступному: обмежений набір даних та доволі висока кількість помилок першого роду.

Можна сказати, що системи виявлення помилок безпеки з високими показниками помилок першого роду можуть виявитися непридатними для використання, а системи з високим показанням помилок другого роду можуть

виявитися марними. Це виправдовує важливість використання систем, які можуть забезпечити низький рівень помилок другого роду, поки рівень помилок першого роду не надто високий.

1.3 Порівняння методів захисту

Проаналізувавши ці методи захисту, можна зробити висновок, що жоден з методів не є ідеальним, та кожний має недоліки, а саме:

1. Динамічні аналізатори:
 - Потрібна велика кількість тестових даних;
 - Потрібно подбати про мінімізації впливу інструментування на виконання програми, що досліджується (включаючи часові характеристики);
 - Потрібно більше ресурсів;
 - Потребується багато часу на тестування;
 - Не вказується місце вразливого коду;
2. Статичні аналізатори:
 - Підхід на основі правил:
 - Потребує багато інтенсивної та монотонної роботи фахівцям;
 - Достатньо великі показники другого роду;
 - Підхід на основі подібності коду:
 - Відсутність достатнього набору даних;
 - Відсутність ефективного алгоритму;

Оцінюючи підхід на основі подібності коду, можна запропонувати підхід на основі глибинного навчання, оскільки глибинні методи навчання володіють гарною здатністю вилучати узагальнені структури з великого обсягу даних, що в даному випадку є накопиченою базою кодів.

Висновки до розділу 1

У цьому розділі були розглянуті основні поняття, що стосуються помилок безпеки програмного засобу, та підходи для виявлення помилок безпеки, такі як статичні та динамічні аналізатори. Недоліки та переваги кожного з них. Запропонований напрям розвитку у вдосконаленні шляхом використання алгоритмів глибинного навчання.

Також, були розглянуті основні практики процесу розробки програмного забезпечення відповідно до Microsoft SDL [2], особливо практики, щодо тестування ПЗ на наявність помилок безпеки.

2 МЕТОД ВИЯВЛЕННЯ ПОМИЛОК БЕЗПЕКИ В ПОЧАТКОВОМУ КОДІ

2.1 Постановка задачі

У цій роботі розглядається проблема побудови системи аналізу початкового коду, заснованого на глибинному навчанні, що дозволяє визначити, чи містить фрагмент коду помилку безпеки (витоки пам'яті, переповнення буфера, тощо). Перевагою такого підходу полягають у здатності автоматично формулювати правило для вирішення того, чи є фрагмент коду вразливим чи ні, на основі накопиченого досвіду написання коду та виправлення дефектів. Глибинні методи навчання володіють гарною здатністю вилучати узагальнені структури з великого обсягу даних, що в даному випадку є накопиченою базою кодів. Тобто, запропонований метод повинен усунути деякі недоліки динамічних та статичних аналізаторів.

Об'єктом дослідження є процес розробки програмного забезпечення.

Предметом дослідження є методи виявлення помилок безпеки початкового коду програмного забезпечення.

Методи дослідження - представлення початкового коду у вигляді АСД, переведення тексту до числового вектору зі збереженням контексту та глибинні методи навчання для розв'язання проблем класифікації.

Наукова новизна полягає в тому, що метод детектування помилок безпеки на основі глибинного навчання отримав подальший розвиток шляхом використання представлення коду у вигляді абстрактного синтаксичного дерева, покращення процедури представлення код-гаджета у числовий вектор та покращений підбір параметрів топології нейронної мережі, що призвело до поліпшення точності знаходження помилок безпеки до 91%. Цей підхід є вдосконаленням технології, описаної раніше в статті [1].

Нашою метою є розробка системи виявлення вразливостей (VulDetect), яка може автоматично визначити, чи є дана програма в вигляді початкового C/C++ коду вразливою, і якщо так, то місця розташування вразливостей. Цього слід досягти, не вимагаючи від людини визначати функції вручну і без високих показників помилок другого роду (до тих пір, поки рівень помилок першого роду не достатньо високий). У цьому розділі ми опишемо загальну модель VulDetect. Почнемо з основних питань, які виникли при побудові моделі, а також основних ідей їх вирішення.

2.2 Загальна схема методу та основні принципи використання глибинного навчання

У цьому розділі ми пропонуємо розглянути деякі основні принципи використання глибинного навчання для виявлення вразливостей. Деякі принципи можуть бути необхідні для вдосконалення або вивченні більш детально, але цього достатньо для поточних досліджень з виявлення вразливостей. При використанні нейронних мереж стандартні основні питання завжди виникають перед початком розробки, і в цьому дослідженні виникли наступні питання:

1. Як представити програму, яка тестується для системи виявлення вразливостей на основі глибинного навчання?
2. Як локалізувати вразливість?
3. Яку архітектуру нейронної мережі вибрати?

2.2.1 Методи представлення коду

Оскільки нейронна мережа приймає числові вектори на вході, необхідно перетворити програму таким чином, щоб зберегти зв'язок між вектором і семантичною інформацією про програму. [12][13][14], були також пропозиції щодо роботи з бінарними даними програми і АСД, а також з вихідним кодом програми. Іншими словами, нам потрібен метод, який створить зв'язок між презентацією програми та векторним поданням, що є входом до глибинного навчання. Ми вирішили використовувати репрезентацію програмного коду до АСД, та працювати вже з цим представленням.

1. Програма спочатку перетворюється на АСД
2. Від усіх витягується код-гаджет (частина коду, що посилається на виклик певної функції або аргументи виклику функції)
3. За допомогою word2vec [25] наведемо гаджет коду, перетворений у вектор, який є вхідними даними для нейронної мережі.

2.2.2 Абстрактне синтаксичне дерево

Абстрактне синтаксичне дерево (АСД) [12]- в інформатиці кінцеве позначене орієнтоване дерево, в якому внутрішні вершини зіставлені (позначені) з операторами мови програмування, а листя - з відповідними операндами. Таким чином, листя є порожніми операторами і представляють тільки змінні і константи.

Синтаксичні дерева використовуються в парсерах для проміжного представлення програми між деревом розбору (конкретним синтаксичним деревом) і структурою даних, яка потім використовується в якості внутрішнього уявлення в компіляторі або інтерпретатор комп'ютерної програми

для оптимізації і генерації коду. Можливі варіанти подібних структур описуються абстрактним синтаксисом.

Прикладом може слугувати алгоритм Евкліда, що наведений на рисунку 2.1, а АСД на рисунку 2.2.

```

1 while b ≠ 0
2     if a > b
3         a := a - b
4     else
5         b := b - a
6 return a
7

```

Рисунок 2.1 – Псевдокод для алгоритму Евкліда



Рисунок 2.2 – Абстрактне синтаксичне дерево для алгоритму Евкліда

Для того, щоб отримати АСД для початкового C/C++ коду одним з варіантів є використання clang compiler toolkit [26]. Для того, щоб отримати АСД потрібно ввімкнути дамп-режим(-ast-dump). Для цього потрібно ввести наступну команду (рисунок 2.3):

```
1 clang -ast-dump test.cc
```

Рисунок 2.3 – Команда для отримання АСД для файлу test.cc

Для файлу test.cc, зміст якого зображений на рисунку 2.4 буде наступний вивід:

```
int f(int x) {
    int result = (x / 42);
    return result;
}
```

Рисунок 2.4 – Програмний код функції, що ділить число на 42

```
TranslationUnitDecl 0x5aea0d0 <<invalid sloc>>
... cutting out internal declarations of clang ...
^-FunctionDecl 0x5aeab50 <test.cc:1:1, line:4:1> f 'int (int)'
| -ParmVarDecl 0x5aeaa90 <line:1:7, col:11> x 'int'
| ^-CompoundStmt 0x5aead88 <col:14, line:4:1>
| | -DeclStmt 0x5aead10 <line:2:3, col:24>
| | | ^-VarDecl 0x5aeac10 <col:3, col:23> result 'int'
| | | | ^-ParenExpr 0x5aeacf0 <col:16, col:23> 'int'
| | | | | ^-BinaryOperator 0x5aeacc8 <col:17, col:21> 'int' '/'
| | | | | | -ImplicitCastExpr 0x5aeacb0 <col:17> 'int' <LValueToRValue>
| | | | | | | ^-DeclRefExpr 0x5aeac68 <col:17> 'int' lvalue ParmVar 0x5aeaa90 'x' 'int'
| | | | | | | ^-IntegerLiteral 0x5aeac90 <col:21> 'int' 42
| ^-ReturnStmt 0x5aead68 <line:3:3, col:10>
| | ^-ImplicitCastExpr 0x5aead50 <col:10> 'int' <LValueToRValue>
| | | ^-DeclRefExpr 0x5aead28 <col:10> 'int' lvalue Var 0x5aeac10 'result' 'int'
```

Рисунок 2.5 – Приклад виводу АСД за допомогою Clang AST

Декларація верхнього рівня в блоці перекладу завжди є декларацією одиниці перекладу (англ. translation unit declaration). У цьому прикладі нашим першим письмовим декларацією користувача є оголошення функції "f". Тіло "f" - це складна заява (англ. compound statement), дочірні вузли якої - заява декларації (англ. declaration statement), яка оголошує нашу змінну результату, і оператор return.

Вузли clang AST моделюються за ієрархією класів, які не мають спільного предка. Натомість, існує декілька великих ієрархій для основних типів вузлів, таких як Decl і Stmt. Багато важливих вузлів AST походять від Type, Decl, DeclContext або Stmt, а деякі класи походять від Decl і DeclContext.

У AST також існує безліч вузлів, які не є частиною більшої ієрархії, і доступні лише для певних інших вузлів, таких як CXXBaseSpecifier.

Таким чином, щоб пройти повний AST, людина починає з TranslationUnitDecl, а потім рекурсивно проходить все, що може бути досягнуто з цього вузла,

Два найбільш основних вузли Clang AST - це заяви (Stmt) та декларації (Decl). Зауважте, що вирази (Expr) також є твердженнями в AST Кланг [26].

2.2.3 Алгоритм представлення коду

Працюючи з початковим кодом, ми повинні вирішити, з якою частиною коду ми будемо працювати, це може бути функції, деякі рядки або ж увесь файл. Нам потрібно вилучити з початкового коду деякий фрагмент, який буде пов'язаний з потенційною вразливістю та з яким ми будемо працювати надалі.

Цей фрагмент коду буде назватися код-гаджет за аналогією зі статтею [1]. Код-гаджет – це набір операторів, пов'язані з потоком даних. Вилучати код-гаджет ми будемо не з початкового коду, а з заздалегідь перетвореним в АСД.

Алгоритм вилучення код-гаджетів складається з наступних етапів (рисунок 2.7):

1. Попередня обробка. На основі початкового коду, використовуючи вбудовану попередню обробку clang, створюється новий файл без директив препроцесора, які були визначені користувачем.
2. Побудова АСД [12]. Для побудови АСД ми використовуємо clang compiler toolkit.
3. Пошук початкових точок. Під початковою точкою розуміється місце в початковому коді, з якого починається аналіз. Ми використовуємо виклики функцій зі стандартної бібліотеки C/C++ як початкову точку (наприклад, malloc, memcpy, fopen і т.д.)
4. Створення графу залежностей. На цьому етапі ми будуємо граф залежності аргументів (або повернених значень) від початкової точки.
5. перейменування змінних або функцій (рисунок 2.6). На цьому кроці ми позбуваємося залежностей від користувацьких імен функцій і змінних. Ми замінюємо всі імена функцій і змінних символічними іменами, такими як: <<VAR_1>>, <<VAR_2>>, і т.д та <<METHOD_1>>, <<METHOD_2>>, і т.д. Водночас, це призводить до зіставлення усіх код-гаджетів в один вигляд.
6. Створення код-гаджету. На основі графу залежностей, код-гаджет формується з токенів, взятих з вузла графу.

Демонстрація алгоритму вилучення код-гаджетів зображена на рисунку 2.7


```
public Integer getMinElement(List myList) {
    if(myList.size() >= 0) {
        return ListManager.getFirst(myList);
    }
    return 0;
}
```



```
public TYPE_1 METHOD_1 ( TYPE_2 VAR_1 )
{ if ( VAR_1 . METHOD_2 ( ) >= INT_1 )
{ return TYPE_3 . METHOD_3 ( VAR_1 ) ; }
return INT_1 ; }
```

Рисунок 2.6 – Приклад позбуття коду від користувацьких змінних

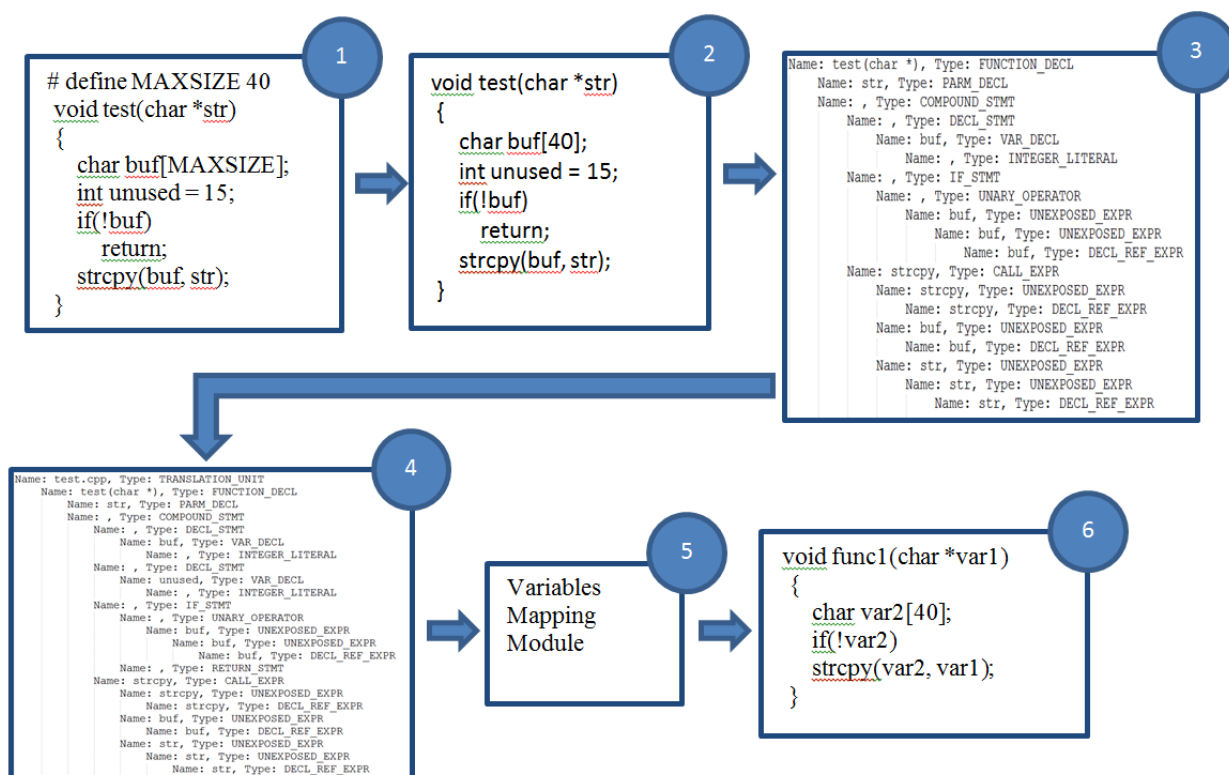


Рисунок 2.7 – Алгоритм вилучення код-гаджету

2.2.4 Локалізація вразливості

Виявлення вразливого коду не є єдиною проблемою, яку потрібно вирішити. Важливим завданням є також пошук місця вразливості. Це означає, що виявлення вразливостей не повинно здійснюватися на рівні програми або функції, що є надто абстрактним. Це призвело до наступних дій: для того, щоб визначити більш точне місце вразливості, програма повинна бути представлена з більш високим ступенем деталізації, ніж програма або функція в цілому. Дійсно, презентація код-гаджету призводить до більш точного виявлення вразливості, оскільки в більшості випадків код-гаджет складається лише з декількох рядків.

2.3 Метод перетворення моделі коду у вектор вхідних параметрів нейронної мережи

Після видалення код-гаджетів з початкового коду, ми перетворюємо його у числовий вектор. Щоб досягти бажаного результату, ми використали word2vec [25]. Word2vec – набір моделей для аналізу семантики природних мов, що представляє собою технологію, яка заснована на дистрибутивній семантиці і векторному поданні слів.

Робота цієї технології здійснюється наступним чином: word2vec приймає великий текстовий корпус в якості вхідних даних і зіставляє кожному слову вектор, видаючи координати слів на виході. Спочатку він створює словник, «навчаючись» на вхідних текстових даних, а потім обчислює векторне подання слів. Векторне подання ґрунтується на контекстній близькості: слова, що зустрічаються в тексті поруч з однаковими словами (а отже, мають схожий

зміст), у векторному поданні матимуть близькі координати векторів-слів. Отримані вектори-слова можуть бути використані для обробки природної мови та машинного навчання.

Прикладом роботи word2vec [25] може слугувати рисунок 2.8:

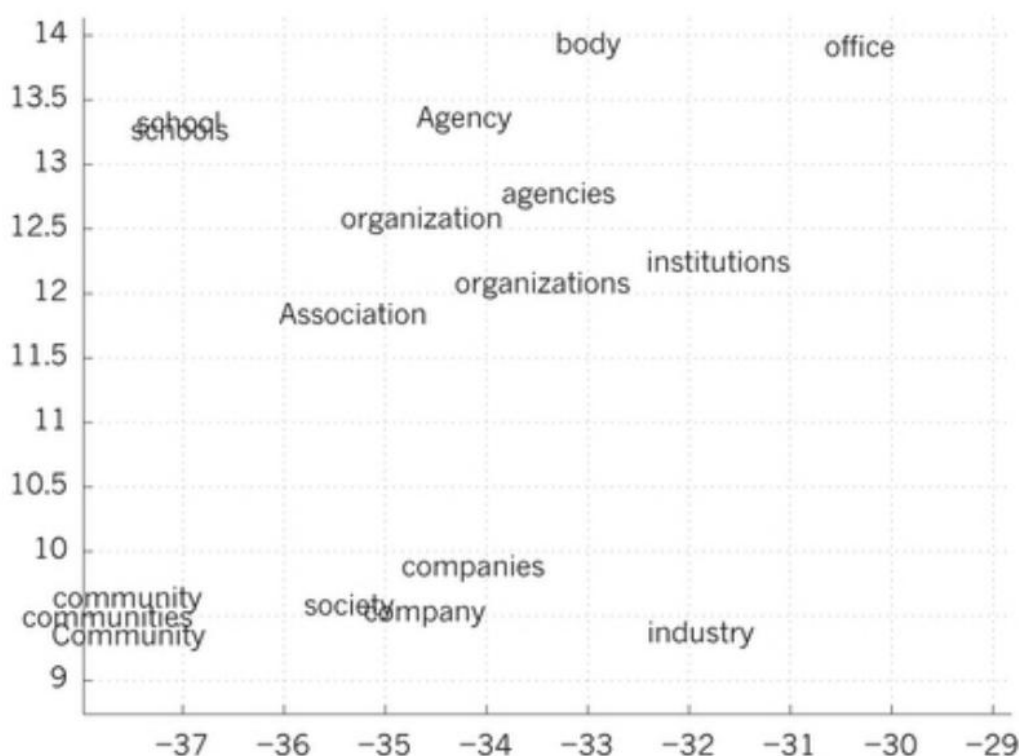


Рисунок 2.8 – Приклад роботи технології word2vec

Як можна побачити з рисунку технологія працює таким чином, що близькі слова мають дуже близьке числове представлення, а слова що суттєво відрізняються мають різне числове представлення.

Отже, результатом роботи word2vec [25] є числовий вектор, в якому зберігається семантика та контекст близьких слів, оскільки вразливий та не вразливий код можуть відрізнятися незначно, тому дана технологія добре підходить для наших цілей.

2.4 Архітектура нейронної мережі

Нейронні мережі зарекомендували себе в наступних областях: обробка зображень, розпізнавання мови, кластеризація, які відрізняються від пошуку вразливостей. З цього випливає, що багато типів нейронних мереж не підходять для наших цілей і що нейронні мережі, які нам потрібні, повинні мати специфічні властивості. Це призвело до таких випадків: оскільки певна лінія коду вразливості може залежати від контексту програмного коду, нейронні мережі, які можуть працювати з контекстом, підійдуть, наприклад, нейронні мережі для обробки людської мови. Можна бачити, що на аргументи виклику функцій найчастіше впливають більш ранні або пізні операції в програмі. Розглянемо нейронну мережу зі зворотним зв'язком, тобто рекурентну нейронну мережу (RNN), що називається [27]. Але ця нейронна мережа має один головний недолік, а саме: зникаюча проблема градієнта [28]. Для вирішення цієї проблеми було обрано більш складну архітектуру, а саме нейронну мережу з довгою короткочасною пам'яттю (LSTM) [29]. Однак навіть нейронна мережа LSTM у стандартній формі не підходить, оскільки мережа однонаправлена, але аргументи функції можуть впливати як на раніше, так і на більш пізні операції в програмі. З цього випливає, що односпрямованої мережі LSTM може виявитися недостатньо. Тому було прийнято рішення використовувати двонаправлену LSTM, а саме –BLSTM [30].

На рисунку 2.9 видно блок-схему нейронної мережі BLSTM з декількома шарами BLSTM, щільним шаром і шаром softmax. Вхід до нейронної мережі на етапі навчання є специфічним векторним поданням. Самі шари BLSTM мають два напрямки: вперед і назад і містять кілька складних клітин LSTM. Щільний шар зменшує кількість векторів, отриманих в результаті рівня BLSTM, і шар softmax приймає вектори з щільного шару в якості вхідних даних і відповідає за презентацію і форматування результату класифікації, який забезпечує зворотний зв'язок для оновлення нейронної мережі. Результатом фази навчання

є нейронна мережа BLSTM з точно налаштованими параметрами моделі, а вихідною фазою виявлення є результати класифікації.

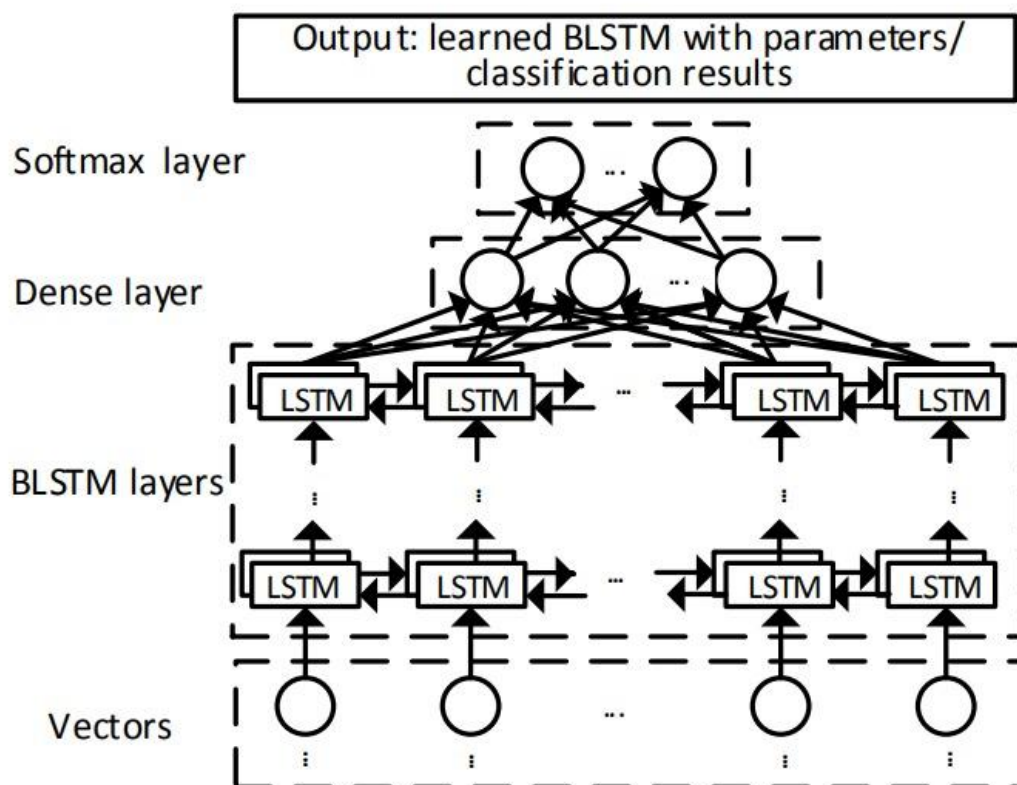
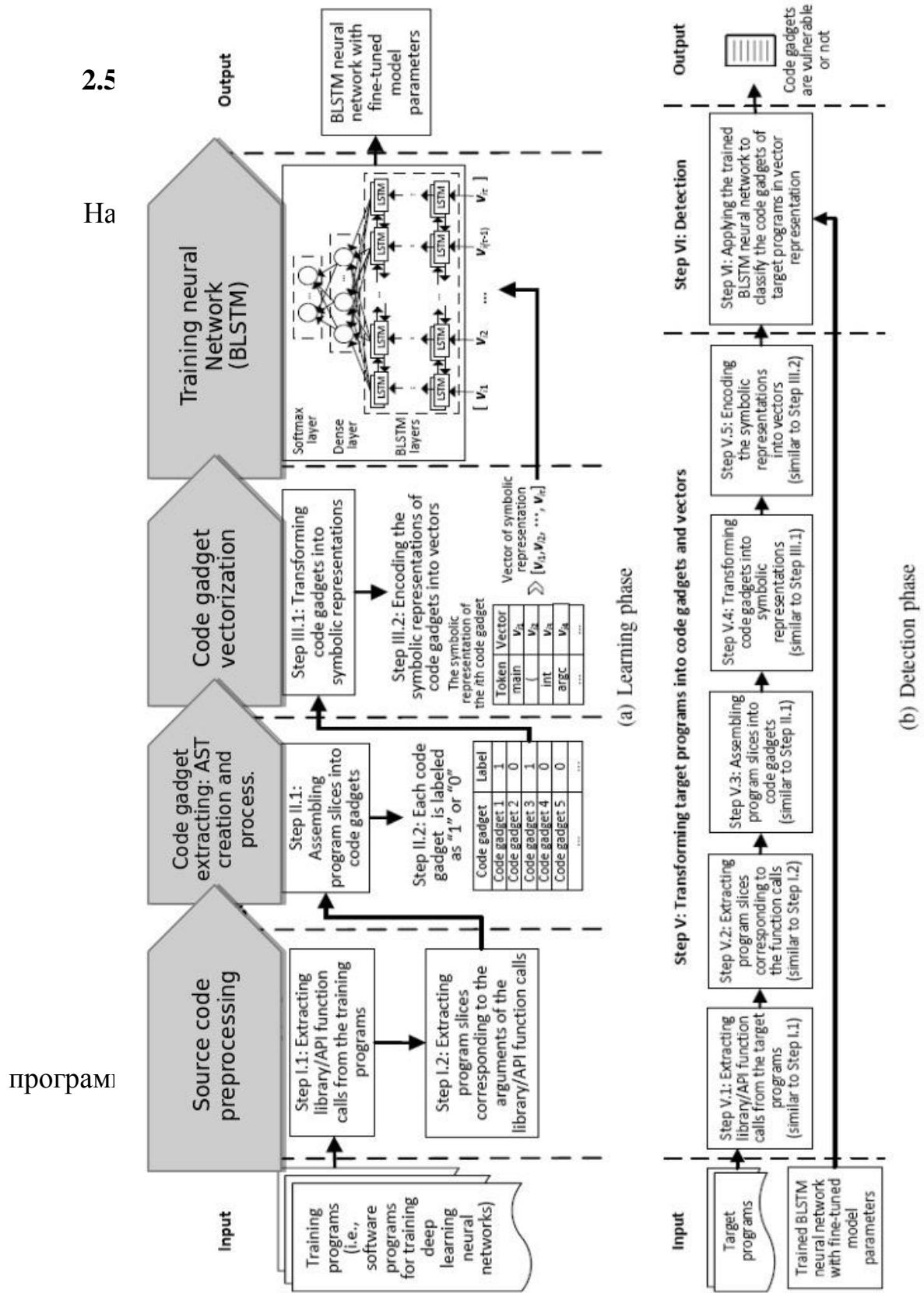


Рисунок 2.9 – Структурна схема нейронної мережі BLSTM

2.5 Метод визначення помилок безпеки в вихідному коді

В результаті, аналіз вразливостей проекту, складався з двох кроків, що проілюстровані на рисунку 2.10 [1]:

1. Фаза виявлення
2. Фаза виявлення



2. Вилучення код-гаджетів

- a. Вилучення усіх початкових точок вразливих функцій в код-гаджети
- b. Маппінг та позначення код гаджетів як вразливий або ні

3. Векторизація код-гаджетів

- a. Перетворення текстових код-гаджетів в числовий вектор за допомогою технології word2vec

4. Навчання нейронної мережі

Виходом з цієї фази є навчена модель нейронної мережі з підібраними гіперпараметрами.

2.5.2 Фаза виявлення

На вхід до другої фази потребується навчена модель нейронної мережі, яка була отримана в результаті першої фази та початковий код програми, яку потрібно перевірити на наявність в ній помилок безпеки. Після чого виконуються наступні кроки:

1. Попередня обробка початкового коду, відповідно до п.1 першої фази
2. Вилучення код-гаджетів, відповідно до п.2 першої фази
3. Векторизація код-гаджетів, відповідно до п.3 першої фази
4. Використання готової моделі нейронної мережі для отримання результатів класифікації

Виходом з цієї фази є число в межах від 0 до 1, яке показує на скільки той чи інший код-гаджет є вразливим, де 0 – не вразливий, 1 – вразливий.

Висновки до розділу 2

В даному розділі був запронований вдосконалений метод виявлення помилок безпеки в початковому коді, а саме: описана постановка задачі, загальна схема методу та основні принципи використання глибинного навчання. Також, були розглянуті методи представлення початкового коду. Розглянули поняття абстрактного синтаксичного дерева та запропонували алгоритм представлення початкового коду з використанням абстрактного синтаксичного дерева. В цьому розділі детально був описаний метод перетворення моделі коду у вектор вхідних параметрів для нейронної мережі, та, власне, розібрана архітектура самої нейронної мережі та вибрана така, яка найбільш задовольняє наші вимоги. Останнім етапом була запропонована загальна архітектура системи, яка складається з двох етапів: етап навчання моделі нейронної мережі та етапу виявлення помилок безпеки в початковому коді.

3 СИСТЕМА ВИЯВЛЕННЯ ПОМИЛОК БЕЗПЕКИ НА ОСНОВІ ГЛИБИННОГО НАВЧАННЯ

Відповівши на усі запитання, був реалізований програмний продукт на мові програмування Python [31], з використанням keras [32], tensorflow [33], word2vec [25], sklearn [34], pandas [35], numpy [36]. Продукт, ім'я котрого було дано VulDetect (Vulnerability Detection) має користувацький інтерфейс, може виявляти помилки безпеки з будь-якого GitHub [37] репозиторію. Продукт використовує нейронну мережу для задачі класифікації. Результатом роботи продукту є вивід код-гаджету, ймовірність його небезпечності та позицію в коді. Даний розділ описує програмну реалізацію, збір даних та навчання моделі нейронної мережі.

3.1 Класифікація коду на основі нейронної мережі

Для початку, потрібно було побудувати модель нейронної мережі, тому була використана модель нейронної мережі з вчителем, де кожен вхідний вектор був позначений як 1 або 0, відповідно, вразливий, або ні. Була обрана наступна структура моделі, відношення кількості епох навчання до кількості навчальних прикладів, дорівнювало 100 та складалась з частин що зображені на рисунку 2.9:

1. п'ять BLSTM [30] шарів, на вході і виході якого вектор розміру N
2. один dense layer типу “relu”[38]
3. один dense layer типу “softmax”[39], який приймає на вхід вектор розмірності N_features та скорочує їх до 2-х

4. останній прошарок використовував функцію “binary_crossentropy” [40] та результатом є число типу “float” в діапазоні від 0 до 1.

```

1 from keras.models import Sequential
2 from keras.layers import Dense, LSTM, Bidirectional
3 from keras.layers import TimeDistributed
4
5 class BaseModel:
6     def __init__(self, n_features):
7         self.n_features = n_features
8         self.n_hidden = int(n_features / 2)
9
10    def model_get(self, config):
11        model = Sequential()
12        for _ in range(5):
13            model.add(Bidirectional(
14                LSTM(self.n_features, return_sequences=True),
15                input_shape=(1, self.n_features))
16            )
17        model.add(TimeDistributed(
18            Dense(self.n_hidden, input_dim=self.n_features, activation='relu'))
19        )
20        model.add(TimeDistributed(
21            Dense(2, input_dim=self.n_hidden, activation='softmax'))
22        )
23        model.compile(
24            loss="binary_crossentropy",
25            optimizer="binary_crossentropy",
26            metrics=["acc"]
27        )
28        return model

```

Рисунок 3.1 – Програмна реалізація BLSTM

На рисунку 3.1 зображена програмна реалізація нейронної мережі з використанням мови програмування Python3 та модуля для роботи с методами машинного навчання – keras [32]. В класі BaseModel є конструктор з параметром, який приймає число з розмірністю вхідного розміру вектору та функція яка повертає модель нейронної мережі. Реалізація keras виконана відповідно до шаблону програмування: будівник. В строках 12-16 добавляються шари BLSTM, в строках 17-19 – шар dense layer типу “relu” [38], в строках 20-22 - один dense layer типу “softmax” [39], та відповідно використовуємо binary_crossentropy optimizer [40] для отримання результатів від 0 до 1.

3.2 Програмна реалізація

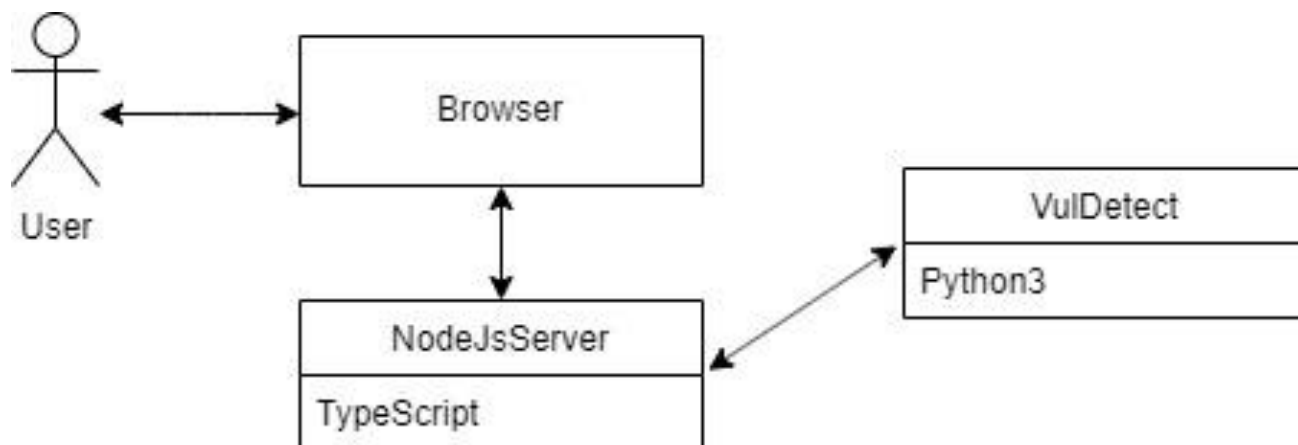


Рисунок 3.2 – Дизайн високого рівня для системи виявлення помилок безпеки

На рисунку 3.2 зображена дизайн високого рівня (High Level Design) [41], в якій виконується огляд всієї системи, виявлення основних компонентів, які будуть розроблені для продукту. Даний дизайн коротко описує всі платформи, системи, та їх взаємодію. На даному дизайну можна побачити, що існують наступні компоненти продукту: NodeJsServer [42] для правильного відображення користувацького інтерфейсу та коректного виклику процедур з VulDetect, який реалізований на Python3 [31].

На рисунку 3.3 зображена діаграма класів [44] для VulDetect- статичне представлення структури моделі. На даній діаграмі зображені наступні статичні елементи, такі як: класи, типи даних, функції, що реалізовані в певних класах, їх зміст та відношення. Діаграма виконана відповідно до Unified Modeling Language (UML) [43].



Рисунок 3.3 – Діаграма класів для серверної частини продукту

На рисунку 3.4 зображена діаграма класів для NodeJsServer [42]. Діаграма також виконана відповідно до UML [43].

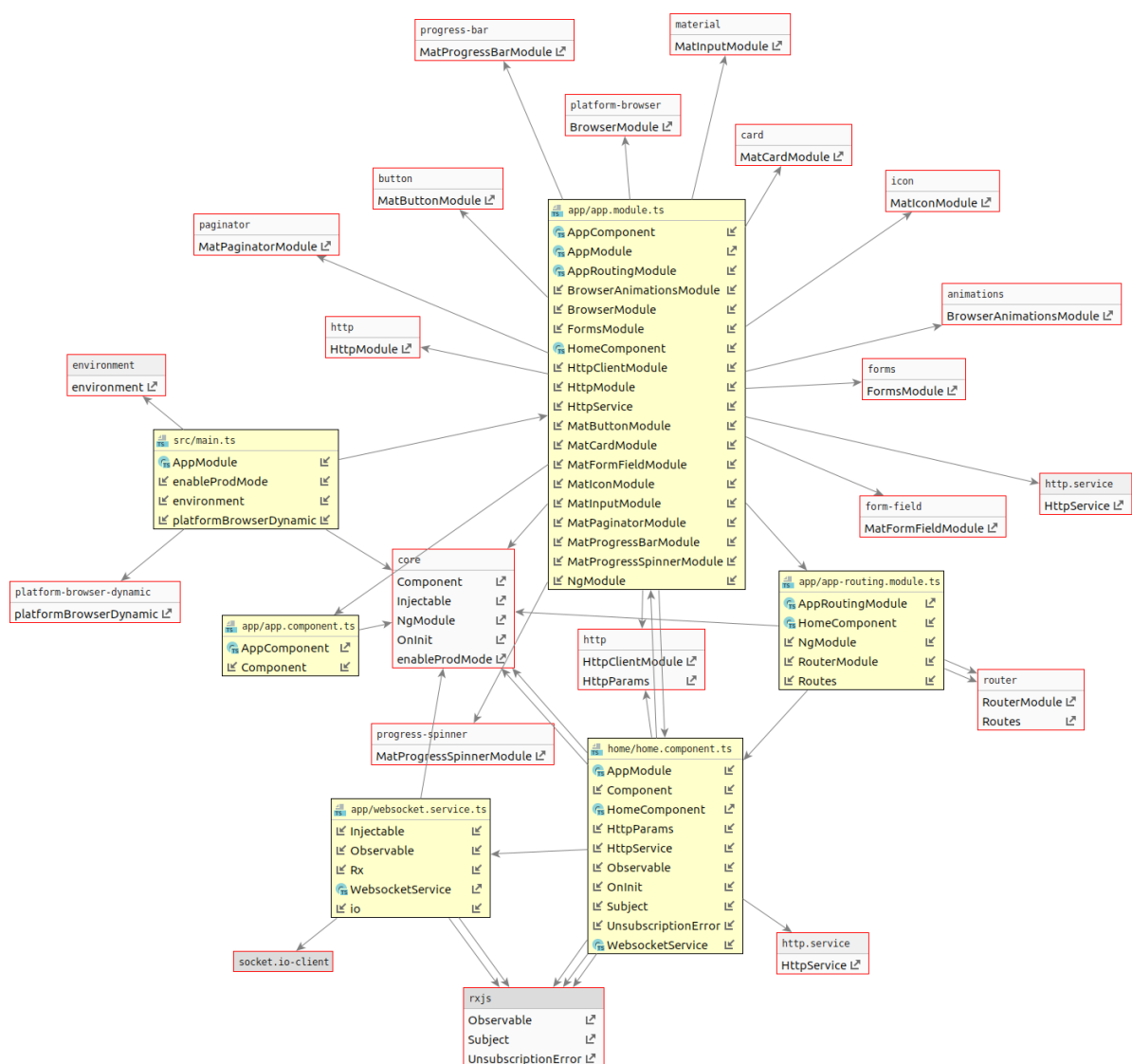


Рисунок 3.4 – Діаграма класів для клієнтської частини продукту

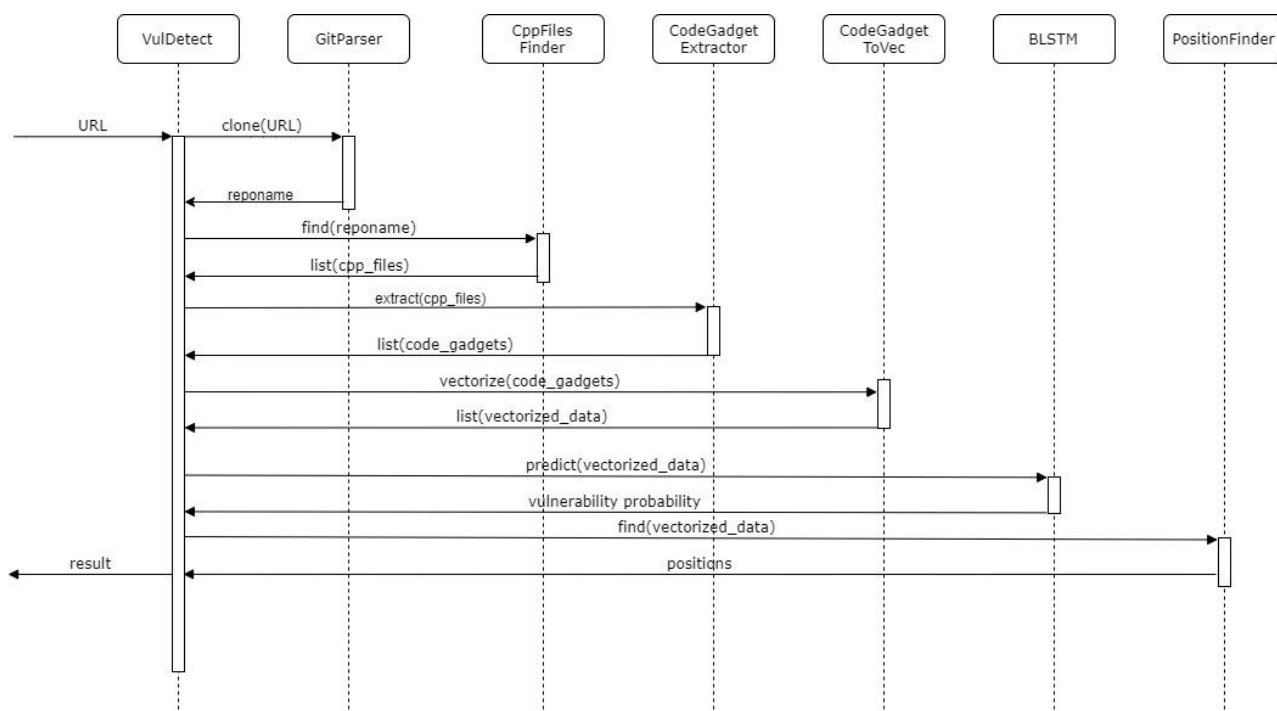


Рисунок 3.5 – Діаграма послідовностей для моделі VulDetect

На рисунку 3.5 зображена діаграма послідовностей [45] для моделі VulDetect на фазі детектування. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовності відправлених повідомлень. На діаграмі показані у вигляді вертикальних ліній різні процеси або об'єкти, що існують водночас. Надісланні повідомлення зображуються у вигляді горизонтальних ліній, в порядку відправлення.

3.3 Навчання моделі

Навчання моделі нейронної мережі складалось з наступних кроків:

1. Пошук навчального набору даних
2. Виконання усіх пунктів з першої фази

3.3.1 Збирання даних

Для підготовки моделі VulDetect в якості навчального набору даних були створені власні код-гаджети з використанням вищезазначених алгоритмів. Набір даних було створено на основі початкових кодів, взятих з National Vulnerability Database (NVD) [23], а також з NIST Software Assurance Reference (SARD) [24]. Набір даних містить 61640 зразків коду з наявністю вразливості переповнення буфера, а також зразки коду з вразливостями, пов'язаними з неправильним управлінням ресурсами.

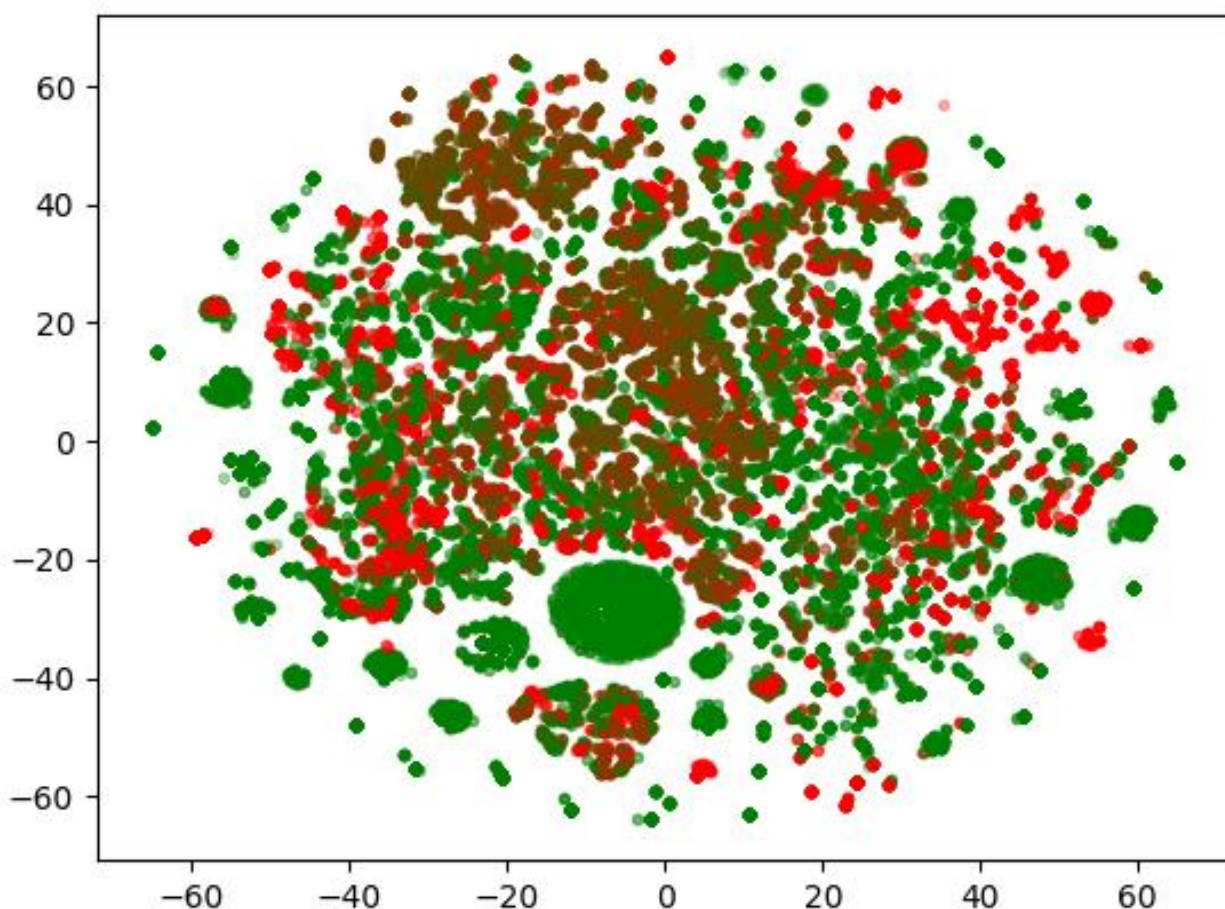


Рисунок 3.6 – Розподілення вразливих та не вразливих код-гаджетів

Після перетворення усіх код-гаджетів було використано t-SNE [46], для переведення в дво-мірний простір, та зроблені графіки розподілу вразливих та не вразливих код-гаджетів (рисунок 3.6). Опис рисунка наведено в таблиці 3.1

Таблиця 3.1 – Розподілу вразливих та не вразливих код-гаджетів

Метрика	Кількість
Загальна кількість код-гаджетів	61638 (100%)
Не вразливі (зелені)	43913 (71%)
Вразливі (червоні)	17725 (29%)

З рисунку 3.2 можна побачити, що результат роботи word2vec [25], scaler [47] є доволі непоганим, оскільки код, є доволі синтетичний, та відрізняється лише декількома строками, тому вразливий та не вразливий код проектується майже в однаковий вектор, з незначними відмінностями. Але, існує набір даних, в яких майже немає вразливих код-гаджетів (зелені області), що призводить до однотипного результату при виконанні.

3.3.2 Результати навчання

Для оцінювання систем знаходження помилок безпеки зазвичай використовують 5 наступних метрик [48]: хибно позитивні спрацювання (FPR), хибно негативні спрацювання (FNR), точність (A), влучність (P), F1-міра (F1). Нехай, істинно позитивна метрика (TP) – кількість вразливих зразків які були виявленні як вразливі, хибно позитивні (FP) – кількість не вразливих зразків, але які були виявленні як вразливі, істинно негативні (TN) – кількість зразків, які

не вразливі, але були виявленні як вразливі та хибно негативні (FN) – кількість вразливих зразків, які були виявленні не вразливими. Таблиця узагальнює визначення.

Набір даних поділили для тренування і тестування у співвідношенні 90% і 10% відповідно.

Навчання виконувалось на 2 x Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz, та зайняло приблизно 300 секунд на одну епоху тренування. Загальний час навчання, на 500 епох, зайняло 42 години.

Таблиця 3.2 – Опис основних метрик для оцінювання систем знаходження помилок безпеки

Метрика	Формула	Значення
Хибно позитивні спрацювання	$FPR = \frac{FP}{FP + TN}$	Частка хибно позитивних зразків у загальній кількості вибірок, які не є вразливими.
Хибно негативні спрацювання	$FNR = \frac{FN}{TP + FN}$	Частка хибно негативних зразків у загальній кількості вибірок, які є вразливими.
Точність	$A = \frac{TP + TN}{TP + FP + TN + FN}$	Правильність всіх виявлених зразків.
Влучність	$P = \frac{TP}{TP + FP}$	Правильність виявлених вразливих зразків.
F1-міра	$F1 = \frac{2 * P * (1 - FNR)}{P + (1 - FNR)}$	Загальна ефективність враховуючи як точність, так і хибно негативні спрацювання.

Та отримали наступні результати, що наведені в таблиці 3.3, 3.4:

Таблиця 3.3 – Точність нейронної мережі на різних етапах

Фаза	Точність
Тренування	88%
Тестування	87%

Таблиця 3.4 – Результати передбачення навченої моделі

Назва метрики	Кількість	% відношення
Усього код-гаджетів	6164	100%
Істинно позитивні (true positive)	4091	66%
Істинно негативні (true negative)	1257	20%
Усього правильних (accuracy)	5348	~87%
Хибно позитивні (false positive)	524	8.5%
Хибно негативні (false negative)	292	~5%
Усього неправильних	816	~13%

В таблицях 3.5, 3.6 розраховані метрики, що приведені в таблиці 3.2:

Таблиця 3.5 – Результати основних метрик навченої моделі

Метрика	Значення
Хибно позитивні спрацювання	0.29
Хибно негативні спрацювання	0.07
Точність	0.87
Влучність	0.89
F1-міра	0.91

Таблиця 3.6 – Результати продуктивності навченої моделі нейронної мережі

Метрика	Кількість
Загальна кількість код-гаджетів	6164 (100%)
Істинно позитивні (зелені)	4091 (66%)
Істинно негативні (червоний)	1257 (20%)
Хибно позитивні + хибно негативні (синій)	816 (14%)

Для дослідження результатів був створений графік з використанням t-SNE [46], в яких місцях виникають хибні результати на даних для тестування

З рисунку 3.7 можна побачити, що вразливий та не вразливий код гарно виявляються, якщо вони суттєво відрізняються один від одного, а коли, код дуже схожий то відрізнити вразливий код від не вразливого доволі важко, але згідно до результатів тестової вибірки, така кількість не перевищує 14%.

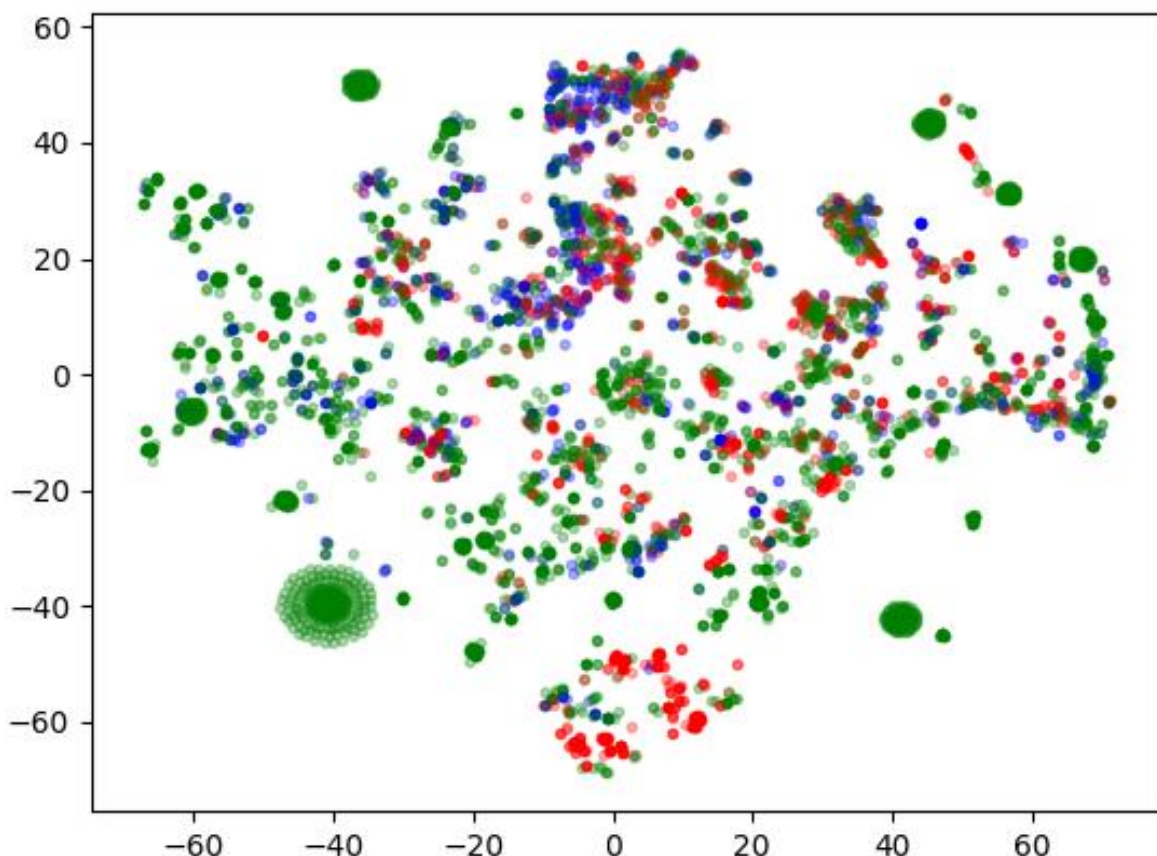


Рисунок 3.7 – Продуктивність нейронної мережі на тестових даних навченої моделі

3.4 Результати дослідження

Результатом дослідження є програмний продукт, VulDetect, який може бути використано в корпоративних цілях, а таблиця 3.7 підсумовує результати дослідження в порівнянні з іншими аналізаторами, такими як Flawfinder [49], RATS [50], Checkmarx [51], VUDDY [52] та VulDeePecker [1]

Таблиця 3.7 – Порівняння існуючих підходів за основними метриками

Метод	FRP (%)	FNR (%)	A (%)	P (%)	F1 (%)
Flawfinder	21.6	70.4	69.8	22.8	25.7
RATS	21.5	85.3	67.2	12.8	13.7
Checkmarx	20.8	56.8	72.9	30.9	36.1
VUDDY	4.3	90.1	71.2	47.7	16.4
VulDeePecker	2.5	41.8	92.2	78.0	66.6
VulDetect	29	7	87	89	91

Програмні продукти з відкритим початковим кодом Flawfinder [49] та RATS [50] мають високі FRP та FNR. Checkmarx [51] краще, ніж Flawfinder та RATS, але все ще мають високі FPR та FNR. VUDDY [52], як відомо, має високий FNR при низькому FPR, тому що він може виявляти лише вразливості, які майже однакові як і в навчальних парограмах. Метод VulDetect набагато ефективніший, ніж VulDeePecker, оскільки VulDeePecker має більший FNR та менший F1.

У програмного продукту VulDetect є графічний інтерфейс, який показано на рисунку 3.8 де зображено результат виявлення помилок безпеки на основі проекту <https://github.com/open-source-parsers/jsoncpp> [53].

<https://github.com/open-source-parsers/jsoncpp>

Source file: src/lib_json/json_value.cpp

Target call: free

Positions in file: [210:63]

```
static inline void releaseStringValue(char* value, unsigned) { free(value); }
```

Source file: src/lib_json/json_value.cpp

Target call: free

Positions in file: [209:61]

```
static inline void releasePrefixedStringValue(char* value) { free(value); }
```

Source file: src/lib_json/json_value.cpp

Target call: strlen

Positions in file: [183:36]

```
inline static void decodePrefixedString(bool isPrefixed,
char const* prefixed,
unsigned* length,
char const** value) {
if (!isPrefixed) {
```

Рисунок 3.8 – Графічний інтерфейс програмного продукту VulDetect

3.5 Обмеження підходу

Оскільки продукт є дослідницьким, то в ньому є декілька обмежень:

- Тексти програм з навчальної вибірки містили специфічні для систем Windows і *nix бібліотеки, тому для коректного створення код гаджетів була потрібна мануальна установка відсутніх залежностей. В іншому випадку ділянки коду, пов'язані з невідомими бібліотеками, не

потрапляли в AST, що значно зменшувало об'єм і якість навчальної вибірки.

- Також, важливим аспектом процесу обробки вихідних кодів є те, що для коректної побудови код гаджетів потрібно вручну скласти список потенційно уразливих функцій і задавати список їх особливостей, наприклад, чи є ця функція `forward` або `backward`, тобто чи потрібно відстежувати аргументи функції потрібно відстежувати повертаються функцією значення.
- Для того щоб код гаджети не включали в себе інформацію зі сторонніх файлів і бібліотек, потрібно вручну задавати каталог з файлами, на основі яких буде будуватися код гаджети. Інакше, в результаті проходу по вкладеним функцій, код гаджет матиме не релевантну, з точки зору користувача, інформацію і, потенційно, може вносити похибка в ході аналізу.
- При відстеженні аргументів або повертаються значень функції, не рідко зустрічаються випадки великої кількості вкладень, в результаті чого код гаджети виходять досить об'ємними. Дана проблема тягне за собою ряд наслідків: збільшується похибка результатів моделі; з точки зору користувача, інформативність результатного звіту падає, так як опис уразливого ділянки коду занадто розпливчасто; продуктивність системи, в цілому, падає, так як потрібно обробляти більше інформації. Від даної проблеми частково вдалося позбутися шляхом мануального задавання порогового значення рівня вкладеності. Дане рішення дозволило скоротити розмір код-гаджетів, а також збільшити точність аналізу, але воно потенційно може випускати важливі деталі код-гаджета.

Висновки до розділу 3

В даному розділі була описана система виявлення помилок безпеки на основі глибинного навчання, а саме описана топологія вибраної нейронної мережі, та її програмна реалізація мовою Python3. Також, в цьому розділі були описані результати дослідження даних, на яких було вирішено навчити модель нейронної мережі, та, власне, результати навчання моделі. В цьому розділі була зроблена візуалізація результатів нейронної мережі, для кращого розуміння проблем детектування та підходу. Згідно до програмної реалізації були наведенні наступні діаграми: дизайн високого рівня, діаграма класів для серверної частини, діаграма класів для клієнтської частини та діаграма послідовностей для продукту. В решті-решт були пораховані основні показники точності методу та проведено порівняння з існуючими підходами. В результаті чого, що даний метод є найбільш точним з усіх методів, що існують, але, даний метод також має і недоліки і обмеження, які були описанні в цьому ж розділі.

4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

Розділ має на меті проведення маркетингового аналізу стартап проекту задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження [54].

4.1 Опис ідеї проекту

В цьому підрозділі аналізується зміст ідеї, що пропонується, можливі напрямки застосування, основні вигоди, що може отримати користувач товару (за кожним напрямком застосування) (табл. 4.1) та відмінність від існуючих аналогів та замінників (табл. 4.2)

Таблиця 4.1 – Опис ідеї стартап-проекту

<i>Зміст ідеї</i>	<i>Напрямки застосування</i>	<i>Вигоди для користувача</i>
Система виявлення помилок безпеки у початковому коді на основі глибинного навчання	1. ІТ компанії, що займаються розробкою ПЗ	1. Швидкість перевірки, в порівнянні зі статичними та динамічними аналізаторами
	2. ІТ компанії, що займаються тестуванням ПЗ	2. Велика точність виявлення помилок безпеки
	3. Програмісти	3. Наявність графічного інтерфейсу
	4. Люди, що вивчають програмування	4. Легка інтеграція з git репозиторіями

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ n/ n	Техніко- економічні характери стики ідеї	(потенційні) товари/концепції конкурентів				W	N	S
		Мій проект	CppChes k	ClangA nalyzer	Svace			
1.	Швидкіст ь роботи	не сильно залежит ь від розміру проекту	залежит ь від розміру проекту експоне нціальн о	залежи ть від часу компіл яції проект у	залеж ить від компі ляції проек ту	дуже залежи ть від розмір у проект у	ліній на залеж ність	виконує ться за констан тний час
2.	Точність виявленн я	91%	36.1%	16.4%	66.6%	<30%	30% <50% <75%	>75%
3.	Веб інтерфейс	+	-	-	+	-	-	+
4.	Інтеграці я з системам и контролю версій	GitHub	тільки локальні проекти	тільки локаль ні проект и	GitHu b	-	тіл ьк и локал ьні проект и	локальн і проекти та GitHub
5.	Необхідн ість в компіляці ї проекту	потрібе н тільки початко вий код	потрібе н тільки початко вий код	потріб на компіл яція коду	потріб на компі ляція коду	потріб на компіл яція коду	-	потрібе н тільки початко вий код

Кінець таблиці 4.2

6.	Виявленн я точного місця помилки безпеки	до виклику функції	до виклику функції	до виклик у функці ї	до викли ку функц ії	факт помил ки безпек и	функ ція в цілом у	місце виклику функції
----	--	--------------------------	--------------------------	----------------------------------	----------------------------------	------------------------------------	-----------------------------	-----------------------------

4.2 Технологічний аудит ідеї проекту

В межах даного підрозділу був проведений аудит технології, за допомогою якої можна реалізувати ідею проекту та заповнена таблиця 4.3

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

<i>№ п / п</i>	<i>Ідея проекту</i>	<i>Технології її реалізації</i>	<i>Наявність технологій</i>	<i>Доступність технологій</i>
1	Використання перетворення початкового коду	Використання перетворення початкового коду по типу абстрактного синтаксичного дерева.	Наявні деякі алгоритми для побудови абстрактного синтаксичного дерева.	Clang compiler tool знаходиться в відкритому доступі.

Кінець таблиці 4.3

2	Впровадження програмної реалізації нейронної мережі, що спеціалізується на класифікації	Використання сучасних фреймворків для побудови моделі нейронної мережі.	Нейронна мережа повинна бути побудована та натренована.	Фреймворки tensorflow, keras знаходяться в відкритому доступі.
3	Використання коду з систем контролю версій	Використання початкових кодів з віддалених репозиторіїв	Використання задокументованого API	Системи контролю версій, такі як GitHub, GitLab, BitBucket мають відкриті API
4	Взаємодія з іншими продуктами	Впровадження задокументованого API, що дозволяє використовувати результати виявлення	Співробітництво з виробниками патчів, аналізу проектів.	Співробітництво можливе за умови двосторонньої вигідності партнерства
5	Використання напрацьованих змін початкового коду	Використання бази кодів, що були з вразливістю та без вразливості	Збирання вразливих та не вразливих початкових кодів	Існують загальновідомі бази даних з початковими кодами
Обрана технологія реалізації ідеї проекту: так як для реалізації ідеї проекту, всі технології є наявними та доступними, тому обираються всі вище описані технології.				

4.3 Аналіз ринкових можливостей запуску стартап-проекту

В межах данного підрозділу було визначено ринкові можливості які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів (табл. 4.4, табл. 4.5).

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

<i>№ п/ п</i>	<i>Показники стану ринку (найменування)</i>	<i>Характеристика</i>
1	Кількість головних гравців, од	20
2	Загальний обсяг продаж, грн/ум.од	20 млн ум. од.
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Потреба у базі даних початкових кодів, що є вразливими та кодів, що не є вразливими
5	Специфічні вимоги до стандартизації та сертифікації	Не потребує стандартизації та сертифікації
6	Середня норма рентабельності в галузі (або по ринку), %	Не менше 150

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

<i>№ n/n</i>	<i>Потреба, що формує ринок</i>	<i>Цільова аудиторія (цільові сегменти ринку)</i>	<i>Відмінності у поведінці різних потенційних цільових груп клієнтів</i>	<i>Вимоги споживачів до товару</i>
	Виявлення помилок безпеки, що можуть призвести до втрат даних або відмові в обслуговуванні цільового програмного забезпечення	1. ІТ компанії, що займаються розробкою ПЗ 2. ІТ компанії, що займаються тестуванням ПЗ 3. Програмісти 4. Люди, що вивчають програмування	Для кожної з категорій існують окремі цінності пропозицій. Пропозиція відрізняється для кожної цільової групи.	<ul style="list-style-type: none"> • точність виявлення помилок безпеки • швидкість роботи • покриття усіх помилок безпеки • зручний користувацький інтерфейс

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 4.6-4.7)

Таблиця 4.6 – Фактори загроз

<i>№ n/n</i>	<i>Фактор</i>	<i>Зміст загрози</i>	<i>Можлива реакція компанії</i>
1	Цінова конкуренція	Деякі продукти наявні в вільному доступі	Покращення точності виявлення помилок безпеки відносно відкритих продуктів
2	Низька точність виявлення помилок безпеки	Деякі помилки знаходяться хибно	Покращення методу, зниження помилок першого та другого роду
3	Низька швидкість роботи	Програма працює дуже довго та займає багато ресурсів	Оптимізація програмного коду, використання більших ресурсів комп'ютера
4	Мала кількість можливих виявлених помилок	Деякі помилки безпеки не знаходяться	Збирання або знаходження початкових кодів з вразливостями та без них

Таблиця 4.7 – Фактори можливостей

<i>№ n/n</i>	<i>Фактор</i>	<i>Зміст можливості</i>	<i>Можлива реакція компанії</i>
1	Поява нових топологій нейронних мереж	Підвищення результатів точності виявлення та часу роботи для класифікації	Впровадження нової топології нейронної мережі для класифікації, розширення функціоналу, проведення маркетингової компанії

Кінець таблиці 4.7

2	Поява нового методу представлення початкового коду	Збільшення можливих точок входу, взяття до уваги додаткових факторів	Підвищення результатів виявлення помилок безпеки
3	Поява відкритої бази даних з початковими кодами з та без вразливостей	Збільшення точності виявлення помилок безпеки, виявлення нових типів помилок безпеки	Розширення типів помилок безпеки, що можуть бути виявленні
4	Залучення нових відомих компаній у якості клієнтів чи постачальників	Підвищення фінансування	Підвищення якості програмного коду, оптимізації, шляхом найму додаткових спеціалістів
5	Поява нової мови програмування	Вийти на ринок виявлення помилок безпеки першими	Вивчення особливостей мови та запровадження даного методу для іншої мови програмування

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (табл. 4.8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

<i>Особливості конкурентного середовища</i>	<i>В чому проявляється дана характеристика</i>	<i>Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)</i>
Олігополістична конкуренція	Статичні аналізатори дуже дорогі для розробки, оскільки потребується багато ресурсів для опису кожного правила виявлення, тому існуючі рішення є тільки в великих ІТ компаніях або точність аналізатору не дуже високий	Вдосконалення рішення для підвищення якості надаваних послуг для клієнтів при невеликих затратах та невеликій ціні продукту
За рівнем конкурентної боротьби: національний	Надання послуг для клієнтів в Україні та за її межами	Застосування новітніх технологій машинного навчання для підвищення конкурентоспроможності
За галузевою ознакою: внутрішньогалузева	Рішення можуть використовуватися тільки компаніям, що займаються розробкою або тестування програмного	Застосування маркетингових методів, які дозволять користуватися продуктом невеликим компаніям, або клієнтам, що навчаються програмуванню

	забезпечення	
--	--------------	--

Кінець таблиці 4.8

Конкуренція за видами товарів: товарно-видова	Рішення, що використовується для задоволення потреб клієнтів, але істотно відрізняються від рішень конкурентів.	Надання послуг з виявлення помилок безпеки на основі машинного навчання
За характером конкурентних переваг: цінова	Ціна запропонованого рішення є меншою за рахунок використання вже готових доступних результатів наукових досліджень та розробок.	Надання порівняного рівня послуг, проте з наголосом на інформаційну безпеку та за меншу ціну.
За інтенсивністю: марочна	Сукупність характеристик та властивостей рішення	Підвищення якості роботи програмного рішення для клієнтів

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі (за моделлю 5 сил М. Портера) (табл. 4.9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	VulDee Pecker	CppCheck ClangAnalyzer Svace Dexter Valgrind Pin Dmalloc	Досконалі продукти, підтримка зі сторони розробки, рішення перевірене часом	Великі ІТ компанії мають великі фінанси та підтримують високу якість розроблююмих продуктів	Товари замінники відсутні
Висновки:	Продукт є дослідицьким, тому не є конкурентом	Є можливості входу на ринок, але існує чимало серйозних конкурентів, що вже працюють на цьому ринку.	Постачальники диктують умови роботи на ринку; компанії, які мають найбільше ресурсів, мають найбільш якісний продукт.	Клієнти диктують умови на ринку; при організації вибору послуги, відчутний рівень відношення до вартості рішення та його якості.	Обмежень на ринку через товари замінники немає.

На основі проведеної роботи проводиться обґрунтування факторів конкурентоспроможності (табл. 4.10)

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

<i>№ п/п</i>	<i>Фактор конкурентоспроможності</i>	<i>Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)</i>
	Якість продукту	Продукт працює швидко та показує гарні показники виявлення, без великої кількості помилок першого та другого роду.
	Рівень ціни	Ціна розробки програмного продукту не велика, оскільки використовуються новітні технології машинного навчання, які здатні замінити деяку працю людини.
	Наявність конкурентів	Конкуренти – великі ІТ компанії, або відкриті продукти, тому продукт або комерційний або недостатньої якості
	Масштабованість продукту	Продукт не потребується в масштабованості, оскільки одна й та ж копія може бути проданою без змін

За визначеними факторами конкурентоспроможності (табл. 4.10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 4.11).

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін

№ п/ п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з Svace, CppCheck, ClangAnalyzer						
			-3	-2	-1	0	+1	+2	+3
1	Якість продукту	20			+				
2	Рівень ціни	10	+						
3	Масштабованість продукту	10							+

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (табл. 4.12)

Таблиця 4.12 – SWOT- аналіз стартап-проекту

Сильні сторони: використання машинного навчання для питання класифікації, мала кількість людської роботи для реалізації підходу, швидкість роботи, низька ціна розробки	Слабкі сторони: низький рівень маркетингу, наявність конкурентів у даній сфері.
Можливості: залучення великих ІТ компаній в якості постачальників чи клієнтів	Загрози: цінова конкуренція; зниження доходів потенційних споживачів; конкуренція в якості продукту та підтримці.

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації (табл 4.13)

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

<i>№ n/n</i>	<i>Альтернатива (орієнтовний комплекс заходів) ринкової поведінки</i>	<i>Ймовірність отримання ресурсів</i>	<i>Строки реалізації</i>
	Створення угоди з певною ІТ компанією	Висока ймовірність отримання ресурсів	до 1 року
	Впровадження рекламних засобів для таргетингової категорії	Середня ймовірність отримання ресурсів	до 6 місяців
	Участь в ІТ виставках, публікація в журналу	Середня ймовірність отримання ресурсів	до 1 року
	Організування патенту	Мала ймовірність отримання ресурсів	до 1 року
	Додання до функціоналу специфічних функцій	Мала ймовірність отримання ресурсів	до 3 місяців

4.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл. 4.14)

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

<i>№ п / п</i>	<i>Опис профілю цільової групи потенційних клієнтів</i>	<i>Готовність споживачів сприйняти продукт</i>	<i>Орієнтовн ий попит в межах цільової групи (сегменту)</i>	<i>Інтенсивніс ть конкуренції в сегменті</i>	<i>Простота входу у сегмент</i>
1	ІТ компанії, що займаються розробкою ПЗ	Клієнти потребують продукт такого типу та готові	Високій рівень попиту	Існує конкуренці я з	Існує складність входу, оскільки великі компанії мають власнорозроблен і продукти
2	ІТ компанії, що займаються тестуванням ПЗ	ним користувати сь	Високій рівень попиту	подібними функціями	Існують конкуренти з подібним функціоналом

Кінець таблиці 4.14

3	Програмісти	Низька зацікавленість	Середній рівень попиту		Не велика зацікавленість, оскільки перевіркою якості є сама компанія
4	Люди, що вивчають програмуван ня	Низька зацікавленість	Низький попит, пов'язаний з невеликою обхідністю вивчати вразливості тієї або іншої функції	Існує конкуренція з подібними функціями	Існують безкоштовні та відкриті продукти, але з меншим функціоналом
Які цільові групи обрано: вибрані групи з високим рівнем попиту, а саме ІТ компанії, що займаються розробкою ПЗ та ІТ компанії, що займаються тестуванням ПЗ					

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку (табл. 4.15) та вибір стратегії конкурентної поведінки (табл. 4.16).

Таблиця 4.15 – Визначення базової стратегії розвитку

<i>№ п/ п</i>	<i>Обрана альтернатива розвитку проекту</i>	<i>Стратегія охоплення ринку</i>	<i>Ключові конкурентоспромо жні позиції відповідно до обраної альтернативи</i>	<i>Базова стратегія розвитку*</i>
	Флангова атака	Стратегія концентрован ого маркетингу	Сильні сторони та можливості рішення	Стратегія спеціалізації

Таблиця 4.16 – Стратегія конкурентної поведінки

<i>№ п/п</i>	<i>Чи є проект «першопрохідцем» на ринку?</i>	<i>Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?</i>	<i>Чи буде компанія копіювати основні характеристики товару конкурента, і які?</i>	<i>Стратегія конкурентної поведінки*</i>
1	Частково	Так	Так, загалом це будуть взаємодія товару з	Стратегія лідера

			інтерфейсом, можливості конфігурації, тощо.	
--	--	--	--	--

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту (див. табл. 4.5), а також в залежності від обраної базової стратегії розвитку (табл. 4.15) та стратегії конкурентної поведінки (табл. 4.16) розробляється стратегія позиціонування (табл. 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 – Визначення стратегії позиціонування

<i>№ n/ n</i>	<i>Вимоги до товару цільової аудиторії</i>	<i>Базова стратегія розвитку</i>	<i>Ключові конкурентно спроможні позиції власного стартап- проекту</i>	<i>Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)</i>
1	Отримання готового продукту зі зручним користувацьким інтерфесом, що легко налаштовується на будь-який продукт, можливість працювати з віддаленими	Стратегія спеціалізації	Сильні сторони та можливості рішення	Швидкість, точність виявлення, доступна ціна

	репозиторіями, точність знаходження місця вразливості			
--	---	--	--	--

4.5 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 4.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 – Визначення ключових переваг концепції

<i>№ n/n</i>	<i>Потреба</i>	<i>Вигода, яку пропонує товар</i>	<i>Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)</i>
1	Знаходження можливих вразливостей програмного забезпечення	Висока точність та швидкість виявлення	Отримання комплексного продукту, який буде точно та швидко знаходити можливі вразливості використання функцій
2	Зручний користувацький інтерес	Наявність користувацького інтерфейсу або іншого зручного подання результатів	Існує веб інтерфейс продукту, де показуються рядки коду, в яких може міститися вразливість
3	Точне вказання місця	Вказується точне місце вразливості	Вказується саме місце в початковому коді, де виникає

	вразливості		помилка безпеки
4	Можливість конфігурації відносно до проекту	Під різні проекти можна налаштувати виявлення помилок безпеки	Можливість налаштувати систему виявлення відповідно до кожного проекту, з врахуванням його специфіки

Надалі розробляється тривірнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості проце-су його надання (табл. 4.19)

Таблиця 4.19 – Опис трьох рівнів моделі товару

<i>Рівні товару</i>	<i>Сутність та складові</i>		
I. Товар за задумом	Система виявлення помилок безпеки в початковому коді на основі глибинного навчання VulDetect.		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Використовує глибинне навчання для питань класифікації, що призводить до високої точності	М	Тх
	2. Швидко працює	М	Тх
	3. Має веб-інтерфейс	М	Тх
	4. Працює с системами контролю версій		
	Якість: виявляються помилки безпеки відносно CWE-119 CWE-399		
Пакування: Надання користувачу електронної ліцензії (ключа доступу) до хмарного сервісу (якщо побудова системи відбувається в хмарному середовищі			
Марка: VulDetect			

III. Товар із підкріпленням	До продажу: початок рекламної компанії, аналіз open source кодів
	Після продажу: продовження рекламної компанії
За рахунок чого потенційний товар буде захищено від копіювання: користувач не має початкового коду, а має лише доступ до веб інтерфейсу	

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субституту, а також аналіз рівня доходів цільової групи споживачів (табл. 4.20). Аналіз проводиться експертним методом.

Таблиця 4.20 – Визначення меж встановлення ціни

№ n/n	Рівень цін на товари- замінники	Рівень цін на товари- аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
	10000	9000	12000	3000-4000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 4.21)

Таблиця 4.21 – Формування системи збуту

№ n/n	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
----------	--	--	----------------------------	--------------------------------

	мінімальна кількість посередників	організувати широку мережу збуту товару	3	не пряма
--	-----------------------------------	---	---	----------

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.22).

Таблиця 4.22 – Концепція маркетингових комунікацій

<i>№ п/п</i>	<i>Специфіка поведінки цільових клієнтів</i>	<i>Канали комунікацій, якими користуються цільові клієнти</i>	<i>Ключові позиції, обрані для позиціонування</i>	<i>Завдання рекламного повідомлення</i>	<i>Концепція рекламного звернення</i>
	Дослідження переваг та властивостей продукту для точного та швидкого виявлення помилок безпеки своїх продуктів	Інформаційні канали	Знаходження помилок безпеки в початковому коді	Донести переваги та доступність до можливих клієнтів	

Висновки до розділу 4

В цьому розділі, відносно до його мети, було проведено маркетинговий аналіз стартап проекту задля визначення принципової можливості його ринкового впровадження та можливих напрямів реалізації цього впровадження. Для цього був описана основна ідея проекту, проведений технологічний аудит ідеї. Додатково, проаналізовані ринкові можливості стартап проекту та розроблена ринкова стратегія проекту. В решті-решт була розроблена маркетингова програма стартап-проекту.

На основі проведеної роботи можна зробити наступні висновки, а саме, що продукт VulDetect має деякі цільові категорії, які можуть бути заінтересовані в даному продукті, а саме ІТ компанії, що займаються розробкою або тестуванням програмного забезпечення, що складають основні цільові категорії. Також, додатково, після аналізу конкурентів, можна зробити висновок, що даний продукт має суттєві переваги відносно конкурентів, та може користуватися попитом у цільовій аудиторії, тому продукт є рентабельним для реалізації, як стартап-проект.

ВИСНОВКИ

У даній роботі представлена нова система виявлення помилок безпеки у початковому коді на основі глибинного навчання, що називається VulDetect. Ця система є вдосконаленням технології VulDeePecker. Модель використовує представлення початкового коду у вигляді АСД. У ході дослідження було розроблено нову систему вилучення код-гаджетів. Було порівняно обидві системи за результатами виявлення помилок у початковому коді, а також було порівнянно з існуючими підходами, а саме статичними та динамічними аналізаторами. У моделі VulDetect знизився показник помилок першого роду, що призвело до збільшення помилок другого роду. Досі існує проблема з правильним набором даних, вирішення якої дозволить підвищити точність прогнозування за допомогою нейронної мережі і збільшити кількість виявлених вразливостей. У майбутньому планується зробити більш ефективним алгоритм вилучення код-гаджетів, що дозволить підвищити точність виявлення вразливостей у початковому коді.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 Li Z. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection [Текст] / Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, Yuyi Zhong // Proceedings 2018 Network and Distributed System Security Symposium. 2018.
- 2 Microsoft SDL [Електронний ресурс] – Режим доступу: <https://www.microsoft.com/en-us/securityengineering/sdl/practices>
- 3 Ярочкин В.И. Информационная безопасность: Учебник. / В.И. Ярочкин // Академический проект: Трикста – 2005. – 544 с.
- 4 Common Weakness Enumeration [Електронний ресурс] – Режим доступу: <https://cwe.mitre.org/>
- 5 Common Vulnerabilities and Exposures [Електронний ресурс] – Режим доступу: <https://cve.mitre.org/>
- 6 Динамічний аналіз коду [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Динамічний_аналіз_коду
- 7 Статичний аналіз коду [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Статичний_аналіз_коду
- 8 Valgrind [Електронний ресурс] – Режим доступу: <http://valgrind.org/info/>
- 9 Pin - A Dynamic Binary Instrumentation Tool [Електронний ресурс] – Режим доступу: <https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>
- 10 DynamoRIO [Електронний ресурс] – Режим доступу: <https://www.dynamorio.org/>
- 11 The Daikon invariant detector [Електронний ресурс] – Режим доступу: <https://plse.cs.washington.edu/daikon/>
- 12 Abstract syntax tree [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Abstract_syntax_tree

- 13 Chua Z. Neural Nets Can Learn Function Type Signatures From Binaries [Текст] / Zheng Leong Chua* Shiqi Shen* Prateek Saxena Zhenkai Liang // 26th USENIX Security Symposium 2017.
- 14 Yamaguchi F. Generalized vulnerability extrapolation using abstract syntax trees [Текст] / Yamaguchi F., Lottmann M., Rieck K. // Proceedings of the 28th Annual Computer Security Applications Conference on - ACSAC '12. 2012.
- 15 Cppcheck [Электронный ресурс] – Режим доступа: <https://sourceforge.net/projects/cppcheck/>
- 16 Clang Static Analyzer [Электронный ресурс] – Режим доступа: <https://clang-analyzer.llvm.org/>
- 17 COVERITY SCAN STATIC ANALYSIS [Электронный ресурс] – Режим доступа: <https://scan.coverity.com/>
- 18 Dexter [Электронный ресурс] – Режим доступа: <https://github.com/Samsung/Dexter>
- 19 Li Z. VulPecker An Automated Vulnerability Detection System Based on Code Similarity Analysis [Текст] / Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Hanchao Qi, Jie Hu // Proceedings of the 32nd Annual Conference on Computer Security Applications - ACSAC '16. 2016.
- 20 Static Code Analysis [Электронный ресурс] – Режим доступа: https://www.owasp.org/index.php/Static_Code_Analysis
- 21 Bellon S. Comparison and Evaluation of Clone Detection Tools [Текст] / Stefan Bellon, Rainer Koschke // IEEE Transactions on Software Engineering. 2007. № 9 (33). С. 577–591.
- 22 Ribeiro A. Ranking Source Code Static Analysis Warnings for Continuous Monitoring of FLOSS Repositories [Текст] / Athos Ribeiro, Paulo Meirelles, Nelson Lago, Fabio Kon // Open Source Systems: Enterprise Software and Solutions.
- 23 NATIONAL VULNERABILITY DATABASE [Электронный ресурс] – Режим доступа: <https://nvd.nist.gov/>

- 24 Software Assurance Reference Dataset [Электронный ресурс] – Режим доступа: <https://samate.nist.gov/SARD/>
- 25 Word2vec [Электронный ресурс] – Режим доступа: <https://code.google.com/archive/p/word2vec/>
- 26 Clang AST [Электронный ресурс] – Режим доступа: <https://clang.llvm.org/docs/IntroductionToTheClangAST.html>
- 27 Mani A. Solving Text Imputation Using Recurrent Neural Networks [Электронный ресурс] – Режим доступа: <https://cs224d.stanford.edu/reports/ManiArathi.pdf>
- 28 Hochreiter S. Untersuchungen zu dynamischen neuronalen Netzen / Sepp Hochreiter // Institut fur Informatik – 1991.
- 29 Hochreiter S. Long Short-term Memory [Текст] / Sepp Hochreiter, Jurgen Schmidhuber // Neural computation. 9 :1735-80 - 1997.
- 30 Ray A. Text recognition using deep BLSTM networks [Текст] / Anupama Ray, Sai Rajeswar, Santanu Chaudhury // 2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR). 2015.
- 31 Python3 [Электронный ресурс] – Режим доступа: <https://docs.python.org/3/>
- 32 Keras: The Python Deep Learning library [Электронный ресурс] – Режим доступа: <https://keras.io/>
- 33 TensorFlow [Электронный ресурс] – Режим доступа: <https://www.tensorflow.org/>
- 34 scikit-learn Machine Learning in Python [Электронный ресурс] – Режим доступа: <https://scikit-learn.org/stable/>
- 35 Python Data Analysis Library [Электронный ресурс] – Режим доступа: <https://pandas.pydata.org/>
- 36 NumPy [Электронный ресурс] – Режим доступа: <https://numpy.org/>
- 37 GitHub [Электронный ресурс] – Режим доступа: <https://github.com/>
- 38 Rectifier (neural networks) [Электронный ресурс] – Режим доступа: [https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

- 39 Softmax function [Электронный ресурс] – Режим доступа:
https://en.wikipedia.org/wiki/Softmax_function
- 40 Understanding binary cross-entropy / log loss: a visual explanation
[Электронный ресурс] – Режим доступа:
<https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a> - 24.02.2018 г.
- 41 High-level design [Электронный ресурс] – Режим доступа:
https://en.wikipedia.org/wiki/High-level_design
- 42 Node.js [Электронный ресурс] – Режим доступа:
<https://nodejs.org/en/docs/guides/getting-started-guide/>
- 43 Unified Modeling Language [Электронный ресурс] – Режим доступа:
https://en.wikipedia.org/wiki/Unified_Modeling_Language
- 44 Class diagram [Электронный ресурс] – Режим доступа:
https://en.wikipedia.org/wiki/Class_diagram
- 45 Sequence diagram [Электронный ресурс] – Режим доступа:
https://en.wikipedia.org/wiki/Sequence_diagram
- 46 t-distributed stochastic neighbor embedding [Электронный ресурс] – Режим
доступа: https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
- 47 StandardScaler [Электронный ресурс] – Режим доступа: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>
- 48 Metrics to Evaluate your Machine Learning Algorithm [Электронный ресурс]
- Режим доступа: <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234> - 24.02.2018
- 49 Flawfinder [Электронный ресурс] – Режим доступа:
<https://dwyer.com/flawfinder/>
- 50 The Rough Auditing Tool for Security [Электронный ресурс] – Режим
доступа:
<https://security.web.cern.ch/security/recommendations/en/codetools/rats.shtml>

- 51 Checkmarx [Електронний ресурс] – Режим доступу: <https://www.checkmarx.com/>
- 52 Kim S. VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery [Текст] / Seulbae Kim, Seunghoon Woo, Heejo Lee. Hakjoo Oh // 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, 2017, pp. 595-614.
- 53 JsonCpp [Електронний ресурс] – Режим доступу: <https://github.com/open-source-parsers/jsoncpp>
- 54 Розроблення стартап-проекту [Електронний ресурс] : Методичні рекомендації до виконання розділу магістерських дисертацій для студентів інженерних спеціальностей / За заг. ред. О.А. Гавриша. – Київ : НТУУ «КПІ», 2016. – 28 с.