Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Розробка і дослідження методу автоматизованого тестування на проникнення з використанням Deep Learning
(тема)

Виконав:

студент II курсу, групи СПзм-20-1

Дубовик Т.І.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Кучук Г.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Дубовику Тарасу Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка і дослідження методу автоматизованого тестування на проникнення з використанням Deep Learning

затверджена наказом по університету від “ 25 ” березня 2022 р. № 33 СТз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022 р.

3. Вхідні дані до роботи система Shadov. інструмент Mulval, браузер Chrome, мобільні платформи IOS, Android.

4. Перелік питань, що потрібно опрацювати у роботі

1) Аналіз єтапів розробки додатків.

2) Особливості автоматизованого тестування на проникнення проникнення

3) Метод автоматизованного тестування на проникнення з використанням технології Deep Learning

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 13 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	28.03.22-05.04.22	
2	Збір і аналіз вимог для тестів на проникнення	05.0422-20.04.22	
3	Розробка методу автоматизованого тестування		
	на проникнення з використанням		
	технології deep learning	21.04.22-15.05.22	
4	Тестування розробки	16.05.22-20.05.22	
5	Оформлення матеріалів кваліфікаційної роботи	21.05.23-30.05.22	
6	Подання кваліфікаційної роботи керівникові та		
	її попередній захист	15.03.2023	
7	Подання кваліфікаційної роботи на рецензу-	20.03.2023	
	вання		

Дата видачі завдання 26 березня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Кучук Г.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 59 с., 17 рис., 7 табл., 1 дод., 16 джерел.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ІНТЕРНЕТ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБ ДОДАТКИ, DEEP LERNING.

Метою кваліфікаційної роботи є – розробка методу автоматизованого тестування на проникнення з використанням технології глибокого машинного навчання. Знаходження мінімальної критичної множини атак – безліч атак найменшої потужності, видалення яких з арсеналу порушника призведе до неможливості досягнення ним мети.

У ході виконання кваліфікаційної роботи застосовано методи порівняння та аналізу, а також інтеграції різного роду систем. У роботі удосконалено метод Deep Q – Lerning Network для аналізу матриці кібервторжень та знаходження оптимальної траєкторії атаки.

ABSTRACT

Master’s thesis: 59 pages, 17 figures, 7 tables, 1 appendices, 16 sources.

FIREWALL, GATE, INTERNET, PROTOCOL, ROUTER, SERVER, WI-FI, WIRELESS NETWORK, WLAN.

The major goal of this thesis is to analyze modern web applications, identify weaknesses. Finding the minimum critical set of attacks is the set of attacks of the least power, the removal of which from the intruder's arsenal will make it impossible for him to achieve the goal.

In the course of this thesis methods of comparison and analysis were applied, as well as the integration of various kinds of systems. In this work, the Deep Q - Lerning Network method has been improved to analyze the cyber intrusion matrix and find the optimal attack trajectory.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення;

QA – (Quality assurance) - тестувальник;

CSS – (Cascading Style Sheets) - каскадні таблиці стилів;

HTML – (HyperText Markup Language) - гіпертекстова мова розмітки;

UI – (User Interface) - інтерфейс користувача;

URI – (Uniform Resource Identifier) - уніфікований ідентифікатор ресурсів;

UX – (User Experience) - досвід користувача;

XML – (EXtensible Markup Language) - розширена мова розмітки;

JSON – (JavaScript Object Notation) - текстовий формат обміну даними;

GUI – (Graphical user interface) - графічний інтерфейс користувача

ВСТУП

Через технологічний розвиток веб програмування зростає потреба у якісному додатку. Використання комп'ютерних систем практично у всіх сферах життєдіяльності суспільства та збільшення кількості інцидентів інформаційної безпеки актуалізувало проблему захисту даних та програмного забезпечення. Одним із шляхів підвищення рівня кібербезпеки є використання методів та засобів тестування на проникнення. Це стосується як сфер реального виробництва та практичних послуг, так і сфери розробки програмного забезпечення.

При тестуванні додатків подібного роду потрібно досконально вивчити потреби та проблеми з якими людина може зіткнутися в процесі користування.

Тестування важливий єтап в розробці програмного забезпечення. Воно входить в усі єтапи розробки тому:

Економить гроші: без належного тестування кількість часу і ресурсів, необхідних для підтримки продукту в довгостроковій перспективі, набагато більше, ніж інвестиції в тестування, не кажучи вже про те, що з часом щось зламається.

Забезпечує безпеку коду при командній роботі: додаток часто створюється командами. Різні люди змінюють один і той же фрагмент коду з плином часу. Наявність тестів роблять цей процес більш безпечним, оскільки ніхто не зламає щось, що не дізнавшись про це. Це також відноситься і до майбутнього, тести забезпечує безпеку коду, коли ви повернетеся через рік або два для внесення змін.

Допомагає у створенні кращої архітектури: коли частина додатка важко тестувати, це зазвичай відбувається через те, що воно тісно пов'язане з іншими частинами або функціональність вашого застосування занадто складна. При їх тестуванні вам потрібно буде зробити їх слабо зв'язаної,

застосувати делегування і патерни проектування, щоб зробити додаток максимально простим і тестується.

Покращує якість коду: ваш продукт менш схильний до збоїв в роботі оскільки тести допомагає написати більш надійний і хороший код, який менш схильний до помилок.

Робить рефакторинг простим і безпечним: створення програмного забезпечення – це ітеративний процес. Вимоги змінюються з плином часу, отже, змінюється і функціональність. Наявність хорошого тестового покриття дозволяє вам модифікувати певний код, перевіряючи, що тести все ще успішно проходять. Якщо це не так, ви робите правки в код таким чином, щоб тести пройшли.

Метою роботи є – розробка методу автоматизованого тестування на проникнення з використанням технології глибокого машинного навчання.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1) дослідити можливості системи Shadov у збиранні фактичних даних для проектування дерев атак, а також платформи Mulval для генерації дерев атак;

2) розробити метод формування матриці кібервторжень із використанням інструменту Mulval;

3) удосконалити метод Deep Q – Lerning Network для аналізу матриці кібервторжень та знаходження оптимальної траєкторії атаки;

4) провести порівняльне дослідження методу автоматизованого тестування проникнення.

Об’єктом дослідження є процесс тестування.

Предмет дослідження. Метод автоматичного тестування на проникнення.

Методи дослідження. Застосовано методи порівняння та аналізу, а також інтеграції різного роду систем.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ВИРІШЕННЯ ЗАВДАНЬ ТЕСТУВАННЯ НА ПРОНИКНЕННЯ

Розробка программного забезпечення складається з багатьох єтапів: збір і аналіз вимог, проектування та дизайн, розробка (програмування), тестування, впровадження та підтримка. Тестування програмного забезпечення – перевірка відповідності між реальним і очікуваним поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином. Тестування входить в усі єтапи розробки ПЗ

1.1 Збір і аналіз вимог для тестів на проникнення

На першому єтапі QA інженер аналізує вимоги до ПЗ. Вони повинні бути систематизовані і ретельно задокументовані. Процедури забезпечення якості повинні вводитися до початку стадії розробки - на етапі збору та аналізу вимог до ПЗ. Тестуються вимоги до ПЗ на відповідність бізнес-цілям, повноту охоплення, доречність використання, цілісність і несуперечливість.

Крім цього, необхідно переконатися в тому, що всі учасники правильно зрозуміли поставлені завдання і те, як саме кожна вимога буде реалізовано на практиці.

Перший етап включає тестування документації вимог, яка описує функціональність проекту, користувальницький інтерфейс, апаратні і програмні інтерфейси, критерії ефективності, питання і ризики, пов'язані з реалізацією проектів, критерії безпеки і коректності системи.

Таким чином, цей етап передбачає збір вимог щодо розроблюваного програмного забезпечення, систематизацію, документування, аналіз, а також виявлення та розв'язання суперечностей.

1.2 Проектування застоснку тестування на проникнення

На етапі проектування програмісти та системні архітектори, керуючись вимогами, розробляють високорівневий дизайн системи. Створення і тестування прототипу із залученням вашої цільової групи - найкращий спосіб оцінити новий проект і зрозуміти, чи буде він успішним як комерційний продукт. Тестування прототипу надає можливість ґрунтовно вивчити проект на самому початковому етапі робіт і внести необхідні зміни відповідно до поставлених цілей.

Проведення тестування прототипу забезпечує повноцінну працездатність продукту після його випуску. Тестування прототипу дозволяє знизити ризики розробки шляхом раннього виявлення невідповідностей бізнес-вимогам, «вузьких місць» в структурі додатки, зручності для користувачів і дефектів логіки функціоналу додатка ще до початку розробки. Своєчасні зміни виконані на етапі прототипирования, допомагають запобігти дорогі переробки системи на стадіях розробки. Завдяки кваліфікованій роботі QA інженерів, тестування прототипу дозволить розрахувати потенційні витрати на кожному етапі створення продукту і визначитися з найбільш ефективною моделлю розробки.

На єтапі проектування QA інженер починає сворювани тестову документацію.

Створення тестової документації значно покращує якість продукту за рахунок більш тісної співпраці, уточнення деталей при розробці плану тестування і документації. Після завершення тестування наявність тестової документації дозволяє перевірити, наскільки успішно були проведені всі етапи тестування.

Документація може включати тестові специфікації - документи, що описують методику тестування і тестові сценарії з кроками для перевірки різних критеріїв оцінки якості програмного продукту. У тестових планах ми описуємо сценарії для виконання димового тестування (smoke testing),

регресійного і приймального тестування. Створюються звіти про результати тестування для кожної версії продукту і оточення. Як правило, в результаті аудиту проекту, ми надаємо замовнику таку документацію:

- список виявлених помилок за пріоритетністю і алгоритми відтворення неправильної поведінки ПО;
- звіт про стан продукту і його окремих компонентів;
- чек-листи з інформацією про якість кожної функції системи;
- аналіз по навантажувальних смок-тесту однієї сторінки для відображення ситуації по швидкості роботи програми та доцільності проведення повного тестування навантаження.

1.3 Принципи розробки застоснку тестування на проникнення

Після того як вимоги та дизайн продукту затверджені, відбувається перехід до наступної стадії життєвого циклу – безпосередньо розробці. Тут починається написання програмістами коду програми відповідно до раніше визначених вимог.

Системні адміністратори налаштовують програмне оточення, front-end програмісти розробляють призначений для користувача інтерфейс програми і логіку її взаємодії з сервером.

Крім того, програмісти пишуть Unit-тести для перевірки правильності роботи коду кожного компонента системи, проводять рев'ю написаного коду, створюють білди і розгортають готове ПЗ в програмному середовищі. Цей цикл повторюється до тих пір, поки всі вимоги не будуть реалізовані.

Програмування передбачає чотири основні стадії:

- розробка алгоритмів – фактично, створення логіки роботи програми;
- написання вихідного коду;
- компіляція - перетворення в машинний код;
- тестування та налагодження – мова, головним чином, про юніт-тестування.

У план процедур щодо забезпечення якості, можуть бути включені різні види тестування. Для зручності і залученості в процес, результати тестування обробляються у вигляді звітів про виконану роботу з описом знайдених дефектів. Розподіляють декілька рівнів тестування, кожен з яких має окремий вигляд, власну специфіку та призначення. Наприклад, модульне функціональне тестування. Даний вид тестування дозволяє грунтовно перевірити кожен окремий компонент (модуль, об'єкт, клас, функцію і ін.) Програмного забезпечення, щоб переконатися в коректності її роботи. Це першорядне, негативний і позитивний тестування знову з'являється функціональності, яке проводиться в ході розробки ПО. Модульне функціональне тестування передбачає тестування кожного окремого компонента ізольовано в штучно створеному середовищі. Даний тип тестування проводиться на основі затверджених вимог до тестування з використанням автоматизованих і ручних тестів.

Частіше використовують підхід test first approach, коли автоматизовані тести створюються до початку розробки ПО. В даному випадку тести, написані до початку розробки, запускаються навпаки створених і інтегрованих шматків коду. Розробка ведеться до тих пір, поки не будуть успішно пройдені всі тести.

Модульне функціональне тестування проводиться під час розробки кожного окремого модуля системи. Тому в разі виявлення недоліків буде необхідно провести редизайн тільки окремого модуля, що тестується, а не всієї системи. Крім того, модульне тестування дозволяє виявити недоліки в технічному завданні або архітектурі проекту, а також працездатність окремих шматків програми на кожному етапі розробки. Після компонування цих модулів досить провести лише системне і регресійні тестування, а в роботі самих модулів помилок вже не буде. Це заощадить ресурси, витрачені на баг-фіксинг вже після релізу продукту.

Так інтеграційне тестування дозволяє зробити тестування логіки взаємозв'язків між більшими частинами програми та виявити дефекти, що

виникли при об'єднанні модулів. Інтеграційне тестування дозволяє переконатися в тому, чи можуть об'єднані модулі працювати без помилок. Як правило, інтеграційне тестування проводять після модульного тестування.

Перевірка, як взаємодіють між собою компоненти системи після проведеного модульного тестування, використовуючи такі підходи:

– висхідний (спершу збираються воєдино і тестуються низькорівневі модулі, а потім поступово додаються модулі більш високого рівня);

– спадний (спершу тестуються високорівневі модулі, а потім додаються всі низькорівневі);

– комплексний, званий також «великий вибух» (модулі всіх рівнів збираються воєдино і тестуються).

В процесі інтеграційного тестування виявляються помилки взаємодії між уже протестованими на попередньому етапі тестування модулями згідно тест-плану. Тому будь-які проблеми, що виникають в результаті інтеграції модулів можуть бути пов'язані з особливостями взаємодії цих модулів.

Інтеграційне тестування включає в себе наступні етапи: складання тест-плану, створення тест-кейсів і юз-кейсів, виконання тестів після інтеграції модулів, виявлення помилок і повторне тестування після їх виправлення.

Після того, як інтеграційне тестування завершено успішно, робиться системне тестування. Його головна мета – вивчити функціональність системи на етапах складання кожної версії продукту, а також на етапі випуску ПО, у вигляді альфа- і бета-тестування.

Системне тестування проводиться за таких умов:

- модульне і інтеграційне тестування успішно завершено;
- розробка ПЗ відповідно до вимог специфікації завершена;
- створена відповідна тестова середовище для проведення системного тестування.

На даному етапі тестування досліджуються поведінкові аспекти, дизайн і відповідність системи передбачуваним очікуванням користувача.

Зазвичай використовуеться два підходи:

- підхід, заснований на базі вимог, коли для кожного окремого вимоги пишуться тест-кейси;

- підхід, заснований на базі юз-кейсів, коли для перевірки способів використання системи пишуться юз-кейси, або сценарії, які перевіряються тест-кейсами.

Системне тестування проводиться наступним чином. Спершу створюється тестовий план, потім тест і юз-кейси. Наступним кроком створюються тестові дані, які використовуються для системного тестування, і проводиться автоматизований прогін тест-кейсів, потім звичайний. За результатами тестування формується звіт по виявлених помилок і проводиться регресійне тестування після їх виправлення. Цикл повторюється до виключення всіх помилок.

Системне тестування також включає тести на відповідність функціональним і нефункціональним вимогам, адаптаційне тестування, навантажувальний і стресове тестування, юзабіліті тестування, тестування сумісності, тестування користувальницького інтерфейсу, тестування продуктивності тощо.

На етапі тестування користувальницького інтерфейсу (UI) перевіряться, наскільки він зручний у використанні і чи відповідає заданим вимогами і затвердженому прототипу.

Тестуються наскільки очікувано поводиться програма і як відображаються елементи інтерфейсу на різних пристроях при здійсненні користувачем певних дій. Це дозволяє оцінити наскільки ефективна робота користувача з додатком.

Наше завдання на даному етапі полягає у виявленні структурних і візуальних недоліків в інтерфейсі додатка, перевірці зручності інтерфейсу для навігації і можливості повного використання функціоналу програми.

Перевіряються, яким чином елементи інтерфейсу реагують на дії користувача і як додаток обробляє дії, пов'язані з використанням клавіатури і мишки або тач-дій. Для цих цілей ми використовуємо кілька підходів:

– ручне тестування, що дозволяє звірити UI додаток на відповідність макетів дизайну і прототипу;

– автоматизоване тестування після кожної збірки продукту для виявлення помилок інтерфейсу і регресійних багів;

– проведення фокус-груп.

Використання різних підходів в тестуванні UI дозволяє протестувати призначений для користувача інтерфейс більш ретельно, поліпшити якість програми та зробити його зручнішим для його використання.

Під час тестування враховуються критерії створення якісного інтерфейсу, а саме:

– мінімальний час на виконання завдань користувачем;

– мінімальна кількість помилок, які допускає користувач при роботі з додатком;

– повне розуміння інтерфейсу користувачами і відсутність неоднозначностей при роботі з ним;

– мінімальний обсяг введеної користувачами інформації;

простота і візуальна привабливість інтерфейсу.

Для тестування UI проводиться кроссбаузерное і Мультиплатформенне тестування, після завершення якого ви отримаєте висококласне додаток, що працює на всі види пристроїв і у всіх сучасних браузерах.

1.4 Впровадження та підтримка застоснку тестування на проникнення

Коли продукт або система практично готова до релізу або експлуатації, проводиться стабілізаційний тестування. Цей етап проводиться в найбільш наближених до реальних умовах або в умовах експлуатації. Деякі з функціональних можливостей можна перевірити тільки на цій стадії, наприклад, взаємодія великих баз даних, оплата в режимі реального часу тощо.

Коли програма протестована і в ній більше не залишилося серйозних

дефектів, приходить час релізу і передачі її кінцевим користувачам.

Після випуску нової версії програми в роботу включається відділ технічної підтримки. Його співробітники забезпечують зворотний зв'язок з користувачами, їх консультування та підтримку.

У разі виявлення користувачами тих чи інших пост-релізних багів, інформація про них передається у вигляді звітів про помилки команді розробки, яка, в залежності від серйозності проблеми, або негайно випускає виправлення (hot-fix), або відкладає його до наступної версії програми.

Крім того, команда технічної підтримки допомагає збирати і систематизувати різні метрики – показники роботи програми в реальних умовах.

2 ОСОБЛИВОСТІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ НА ПРОНИКНЕННЯ

2.1 Види тестування застосунків

Розглянемо види тестування програмного забезпечення, в залежності від переслідуваних цілей, можна умовно розділити на наступні групи:

- функціональні;
- нефункціональні;
- пов'язані зі змінами.

Функціональні види тестування. Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному, інтеграційному, системному і приймальному. Функціональні види тестування розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів:

- функціональне тестування;
- тестування безпеки;
- тестування взаємодії;
- GUI тестування.

Далі розглянемо нефункціональні види тестування. Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, «Як» система працює.

Далі перераховані основні види не функціональних тестів. Всі види тестування продуктивності, тобто, тестування навантаження, стресове тестування, тестування стабільності або надійності, об'ємне тестування, тестування установки, тестування зручності користування, тестування на відмову і відновлення, конфігураційне тестування [28].

2.2 Аналіз конфігурацій

Аналіз виконувався за допомогою statcounter додатку. Проаналізуємо популярні браузери та девайси в Україні за останній рік. Перелік браузерів для десктопів представлено на рисунку 2.1. Перелік браузерів для мобайл девайсів відображено на рисунку 2.2. Також проаналізовано роздільну здатність екрану десктопів, результати представлено на рисунку 2.3.

Рисунок 2.1 – Графік популярних браузерів для десктопів

Рисунок 2.2 – Графік популярних браузерів для мобайл девайсів

Таблиця 2.1 – Сумарна таблиця конфігурацій

	Десктоп девайс	Мобайл девайс
Браузери	−Chrome −Safari	Chrome Safari
Версія браузеру	Остання	Остання
Роздільна здатність	1366 px і вище	360 px і вище

З огляду на окреслені критерії – приватність, зручність і доступність – насправді, існує лише один браузер, який задовольняє всім вимогам, і це Firefox. Справжня його відмінність полягає не у функціональності, а у приватності. Safari – найприватніший браузер, який не замикає вас в екосистемі. Користуйтеся ним на будь-якій операційній системі, на всіх своїх пристроях і відчувайте себе захищеними під час роботи.

Браузери пройшли тривалий шлях з моменту представлення і завоювання ринку браузером Chrome. Більшість сучасних браузерів скоротили розрив щодо доступності й функціональності, а в деяких сферах, як швидкість та приватність, фактично перевершили Chrome.

Було проведено тестування навантаження 80 користувачів та 10 сек відклику (рисунок 2.3).

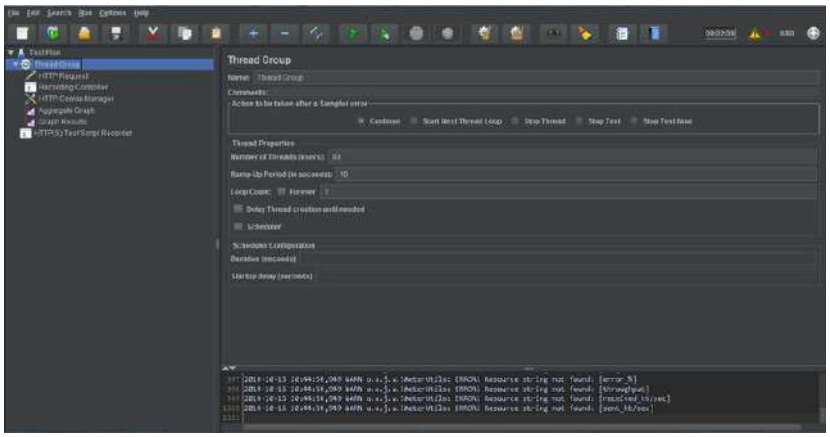

Рисунок 2.3 – Налаштування групи потоків

Задаємо методом GET запит для отримання файлу з головної страниці додатку (рисунок 2.4) та результати тестування відображаються у Aggregate Graph таблиці.

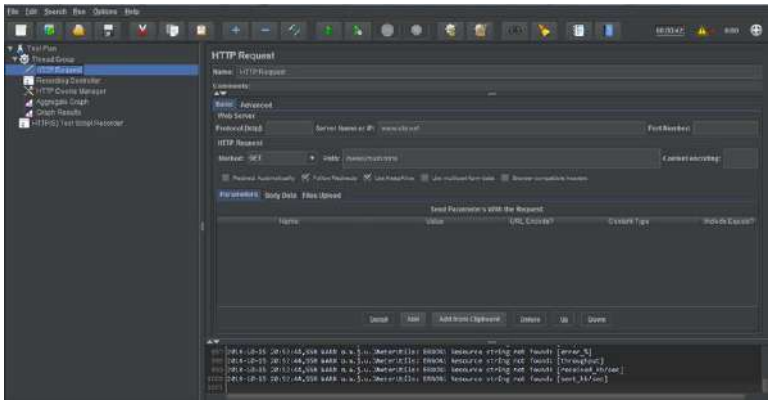

Рисунок 2.4 – Отримання файлу з головної страниці

Для отримання більш точних результатів дослідження було проведено аналогічне тестування навантаження 80 користувачів та 3 сек відклику. Задаємо методом GET запит для утримання файлу з головної страниці додатку та отримуємо результати тестування відображаються у Aggregate Graph таблиці (рисунок 2.5).

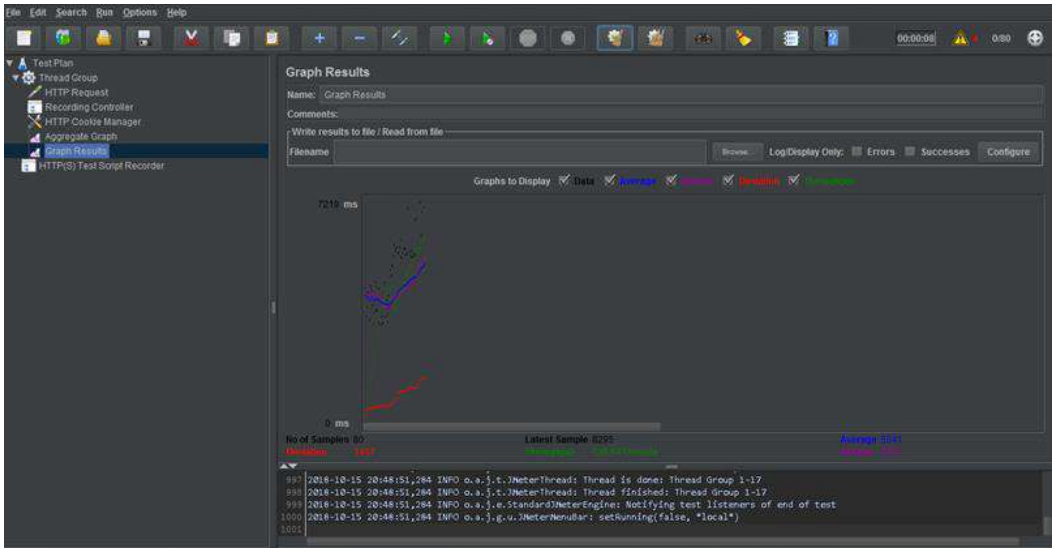

Рисунок 2.5 – Графік роботи додатку

Безпека програмного забезпечення узагалом базується на 3-х

принципах: конфіденційності, цілісності, доступності.

Конфіденційність – це приховування певних ресурсів або інформації. Під конфіденційністю також інколи можна розуміти обмеження прав доступу до ресурсу деякої категорії користувачів. Наприклад: неавторизований користувач наділяється частковими правами, порівнюючи з авторизованим користувачем;

Перевірка цілісності інформації – описує характеристики здатності системи до самовідновлення при пошкодженні окремих сегментів програми або даних, через внесення неправильних змін у їх масиви авторизованими чи неавторизованими користувачами;

Доступність – це, по ідеї, безперешкодний доступ до ресурсу, внутрішнього об’єкту або пристрою авторизованим користувачем. Як правило, чим більш критичний ресурс, тим вищий рівень доступності повинен бути. Поняття критичний означає, що деякий програмний або апаратний ресурс, в кожен момент часу може використовуватися одним і тільки одним процесом, потоком або перериванням.

Які точки будуть вразливими у конкретній програмній структурі – наперед складно передбачити, ми вже з Вами це обговорювали – «людська фантазія та жадоба не мають меж», «закриються одні двері чи друге віконечко пролізти» хакери завжди перебуватимуть у пошуках альтернативного.

По стратегіях Тестування Безпеки Програмного Забезпечення існує кілька міжнародно-визнаних класифікацій вразливостей програмного забезпечення.

Розглянемо загальні терміни, які використовуються в тестуванні безпеки.

Визначення – Метою даного етапу є визначення системи в межах її обсягу та послуг, які при цьому задіяні. Цей етап не призначений для виявлення вразливих місць. Визначення версії допомагає знайти зміни з попередніх версій програмного забезпечення і таким чином вказати на

потенційні вразливості.

Пошук вразливостей. Після етапу виявлення цей етап перевіряє на наявність відомих проблем безпеки за допомогою автоматизованих засобів відповідно до умов. Рівень повідомлень ризику встановлюється автоматично за допомогою інструменту, без ручної перевірки або втручання тестувальника. Також, цей етап може бути доповнений певним скануванням, яке базується на ідентифікації. Воно слідкує за деякими помилковими спрацьовуваннями системи за допомогою облікових даних для перевірки (наприклад, локальні облікові записи Windows).

Оцінка вразливості, цей етап визначає і використовує сканування для пошуку вразливостей, які згодом розміщує в контексті навколишнього середовища. Прикладом може служити видалення загальних помилкових спрацьовувань із звіту та виявлення рівня ризиків, які повинні бути застосовані до кожного спрацьовування для покращення розуміння системи та її контексту.

Оцінка безпеки базується на оцінці вразливості, за рахунок додавання ручної перевірки для підтвердження експозиції, але не включає експлуатацію вразливостей, щоб отримати більш широкий доступ. Перевірка може проводитися у вигляді несанкціонованого доступу до системи, щоб підтвердити налаштування та захист системи, задіяти системний лог, повідомлення про помилки, коди тощо. За допомогою оцінки безпеки тестувальники прагнуть отримати широке представлення системи, але без подробиць про її внутрішню функціональність.

Випробування на проникнення – це тест проникнення імітує напад зловмисників. Ґрунтуючись на попередніх етапах та включаючи в себе використання знайдених вразливостей, тест намагається отримати більш широкий доступ до системи. При використанні цей підхід приводить до розуміння того, як зловмисник отримає доступ до конфіденційної інформації та повпливає на цілісність даних. Кожен тест розрахований на використання постійної та повноцінної методології таким чином, що дозволяє тестеру

використовувати свої здібності, щоб вирішувати проблеми, які виходять за межі дії їх інструментів, їх власних знань та досвіду. Це допомагає знайти ті вразливості, які б не були ідентифіковані ні одним автоматизованим засобом. Цей підхід ґрунтується на типі атаки в порівнянні з оцінкою безпеки системи, яка охоплює більш широку зону системи.

Аудит безпеки – це функція, яка дивитися на конкретний елемент керування або дотримання його специфіки. Охарактеризований вузькою сферою, цей тип взаємодії може використовувати як будь-який з підходів, що обговорювався раніше (оцінка вразливості, оцінка безпеки, тест проникнення).

Огляд безпеки – підтвердження того, що галузеві стандарти або внутрішня безпека були застосовані до компонентів системи або продукту. Це, як правило, робиться в ході аналізу недоліків і використовує побудову / огляд коду або виконується шляхом перегляду проектної документації та архітектури діаграм. Ця дія не використовує будь-яку з попередніх підходів (оцінка вразливості, оцінка безпеки, тест проникнення, аудит безпеки).

2.3 Автоматизоване тестування застоснку

Сучасне програмне забезпечення – складний багатофункціональний об’єкт. Його якість забезпечується щедрим тестуванням на кожному етапі життєвого циклу SDLC. Розробниками в загальному перевіряється робота коду. QA-спеціалістами — перевіряється все, що можна уявити. І навіть щоразу коли кінцевий користувач взаємодіє з програмою – виявляється він теж її тестує, бо у результаті ймовірно ще можуть повилазити непередбачені доти баги. Хоча у цьому випадку, кінцевий споживач не завжди повідомить про інцидент службі підтримки продукту, якщо не використані інструменти моніторингу. Як бачите тестування ПЗ — значний кусень роботи, значить його доцільно проводити ефективно: якомога швидше із мінімальними витратами.

Таблиця 2.2 – Переваги та недоліки автоматизованого тестування

Переваги	Недоліки
Підвищує точність та швидкість виявлення помилок порівняно з ручним тестуванням.	Вибір правильного інструменту вимагає значних зусиль, часу і планування.
Економія часу та зусиль.	Потреба в знаннях засобів тестування.
Збільшує охоплення тестуванням, оскільки багато інструментів тестування можна використовувати одночасно, що дозволяє паралельне тестування та різні сценарії тестування.	Вартість придбання інструменту для тестування та тестового обслуговування.
Повторюваний сценарій.	Потрібна певна кваліфікація для написання сценарію.

Автоматизоване тестування – це процес, що використовує програмне забезпечення, відмінне від програмного забезпечення, що тестується, для контролю виконання перевірок та порівняння фактичних результатів із очікуваними. Засоби автоматизації використовуються для автоматизації певних розділів ручного тестування, але не всіх. Автоматизоване тестування, як правило, економить час. Тестувальник може протягом короткого періоду ефективно виконувати велику кількість перевірок та важливих і повторюваних завдань, разом з тим, можна автоматизувати тестування, яке важко було б зробити вручну. Окрім економії часу, автоматичне тестування економить значну кількість грошей та зусиль, підвищує якість тестування, а також сприяє підвищенню точності програмного забезпечення.

Здебільшого засоби автоматизації тестування актуально застосувати в наступних випадках:

- для рутинних операцій — людині вони не цікаві, у людини настає втома, розсіяність. Наприклад, форми, в яких міститься велика кількість полів для набору даних (перебір даних). Автоматизований тестовий процес дозволяє автоматично виконувати безпомилкове заповнення полів, а також після збереження внесених даних здійснювати їх перевірку. Заповнюючи некоректними даними форму, одночасно проводиться перевірка різної валідації;
- коли необхідне повторне використання тестів при коригуванні ПЗ;
- довгих сценаріях (від кінця до кінця);
- з бекендом, що є найбільш важкодоступним;
- при роботі із базами даних, логуванням файлів, даними бази де присутній високий потенційний ризик припуститися помилки, і ці помилки вкрай небажані;
- тестування даних, для яких необхідні точні математичні розрахунки;
- добра практика автоматизовувати пошук даних у додатку, бо так гарантовано забезпечується швидке знаходження помилок.

Навпаки не варто застосовувати автоматизоване тестування:

- для перевірки нової функціональності програми — очевидно, ми не знаємо ще її поведінки;
- у випадках, де обов’язково необхідна участь людини — автоматизоване тестування — відмінний інструмент для перевірки коду, але для комп’ютера і його коду залишаються незбагненними, «просте людське співчуття», наприклад, тестування користувацького інтерфейсу чи «почуття прекрасного» у поєднані кольорів верстки (всі чергові розробки нейронних мереж це підтверджують);
- на малесенькому проекті вкладення коштів в автоматизацію 100% буде невиправданою розкішшю. Ручне тестування у цьому випадку організувати легше і швидше. Теж саме на великому проекті, якщо проект «з’їдає» занадто ресурсів, то це вірна ознака того, що щось не те відбувається з автоматизацією на проекті;

- якщо проект має часті релізи, теж цілком можлива ситуація, що здійснити ручне тестування можна швидше, ніж написати тести. Автоматизоване тестування вимагає фахівців, які у ньому тямлять, часу щоб вони вникнули у суть проекту, плюс саме написання скрипту вимагагає часу.

2.4 Автоматизоване тестування на проникнення

Використання комп'ютерних систем практично у всіх сферах життєдіяльності суспільства та збільшення кількості інцидентів інформаційної безпеки актуалізувало проблему захисту даних та програмного забезпечення. Одним із шляхів підвищення рівня кібербезпеки є використання методів та засобів тестування на проникнення. Це стосується як сфер реального виробництва та практичних послуг, так і сфери розробки програмного забезпечення.

Донедавна тестування на проникнення виконувалося вручну. При цьому насамперед, ґрунтуючись на власному досвіді та ерудиції, тестувальникам необхідно було провести аналіз комп'ютерної системи для виявлення вразливості. Тільки після цього проникнути в систему та скомпрометувати програмне забезпечення. Це досить трудомістка задача, що з одного боку потребує великого обсягу знань тестувальника, і з іншого боку має безліч ризиків суб'єктивного характеру. Тому останнім часом все більше організацій використовують варіанти планувальників атак на проникнення на основі моделей автоматизованої цільової системи. Так, наприклад, компанія Core Security використовуючи цю ідею з 2010 року у своєму інструменті Core IMPACT використовує планувальник атак MetricFF. Крім того, Core Security розпочала практику реалізації окремих етичних кібератак відповідно до відомих експлойтів програмних уразливостей. Однак уніфікованого рішення для процедур тестування на проникнення не було отримано.

Одним із шляхів вирішення даної суперечності є використання методів дерев атак. Дані методи у своїй основі використовують положення теорії

графів Брюса Шнайєра [31], який ввів неформальне поняття дерев атак для систематизації та категоризації різних сценаріїв атак на комп'ютерні системи.

Аналізуючи дерево атак пентестувальники можуть візуалізувати та розібратися у взаємозв'язках між моделями атаки. Так, наприклад, у роботі [32] пропонується спосіб моделювання та виявлення атак, заснований на графі. При цьому, переходи зі стану в стан розширеного дерева атак характеризуються атрибутами часу життя TTL і ступеня впевненості PC. Перший атрибут відображає часові залежності між етапами атаки і дозволяє зменшити кількість помилкових спрацьовувань системи виявлення кібервторжень. Другий характеризує можливість досягнення мети при досягнутих підцілях. Слід зазначити, що такий прототип етичних кібервторжень ілюструє сценарії атак, проте кількість хибних спрацьовувань не знижує.

Проведені дослідження показали, що одночасно з дослідженнями моделей та методів синтезу дерев атак доцільно проводити їхній аналіз. Це є необхідною умовою, оптимізації та зниження їх візуальної складності. Результат аналізу визначається призначенням граф атак. Для прикладів тестування на проникнення такий аналіз має знайти найімовірніші сценарії нападу атакуючого.

Серед багатьох методів аналізу графів атак можна назвати такі. Автори роботи [33] визначають найкоротші шляхи до цілей алгоритмом Дейкстри. Також обчислюється безліч шляхів найменшої вартості за алгоритмом Наора-Брутлага. Показується, що завдання визначення множини ефективних заходів захисту (у сенсі вартості) є NP-важким. Пропонується наступна інтерактивна техніка: адміністратор відповідно до вжитих заходів захисту змінює модель мережі в конфігураційному файлі, а потім виконує перерахунок найкоротших шляхів, щоб побачити чи підвищили ці зміни рівень захищеності мережі чи ні. Однак автори статті створили графік атак реалістичного розміру на основі 10 або 20 шаблонів, і не вирішили всіх проблем, пов'язаних із зіставленням шаблонів з конфігурацією та профілем зловмисника. Крім того, наявність

шаблонів атак і конфігураційного файлу також може бути ще однією вразливістю. Якщо вони потраплять у чужі руки, можуть стати цінними інструментами для зловмисника.

На роботі [34] автор проводить аналіз, метою якого є знаходження мінімальної критичної множини атак – безліч атак найменшої потужності, видалення яких з арсеналу порушника призведе до неможливості досягнення ним мети. Показується, що завдання пошуку такої множини є NP-повною, і пропонується “жадібний” евристичний алгоритм її розв'язання складності O(mn), де m – число станів та ребер, n – число атак. У той же час, як зауважують самі автори, представлена технологія не охоплює всього різноманіття методів аналізу та оптимізації графів, що надає можливість дослідникам шляхи подальшого вдосконалення.

У роботі [7] для вирішення завдання аналізу дерева атак автори застосували метод машинного навчання. При цьому за основу взяли Q - навчання для пошуку траєкторії атак. Проте незначність простору дій та простору вибірки знизили практичну цінність зазначеної розробки. На думку ряду авторів корисним удосконаленням у питаннях аналізу дерева атак для тестування на проникнення стала технологія глибокого машинного навчання з підкріпленням. Наприклад, у роботі виконано завдання аналізу та оптимізації графа атак за допомогою даної інтелектуальної технології. Водночас широта спектру можливих уразливостей, а також недоліки методів автоматизації тестування на проникнення для забезпечення безпеки комп'ютерних систем залишає актуальним питання моделювання та розробки відповідних методів.

Загальну структуру методу можна у вигляді комплексу методів та інструментів побачити рисуноку 2.6. Як видно з рисунка, структура методу передбачає наявність трьох компонентів: комплексу технологій та засобів формування матриці атак; методу глибокого навчання з підкріпленням для обробки даних матриці атак; інструментів, платформ та засобів тестування на проникнення.

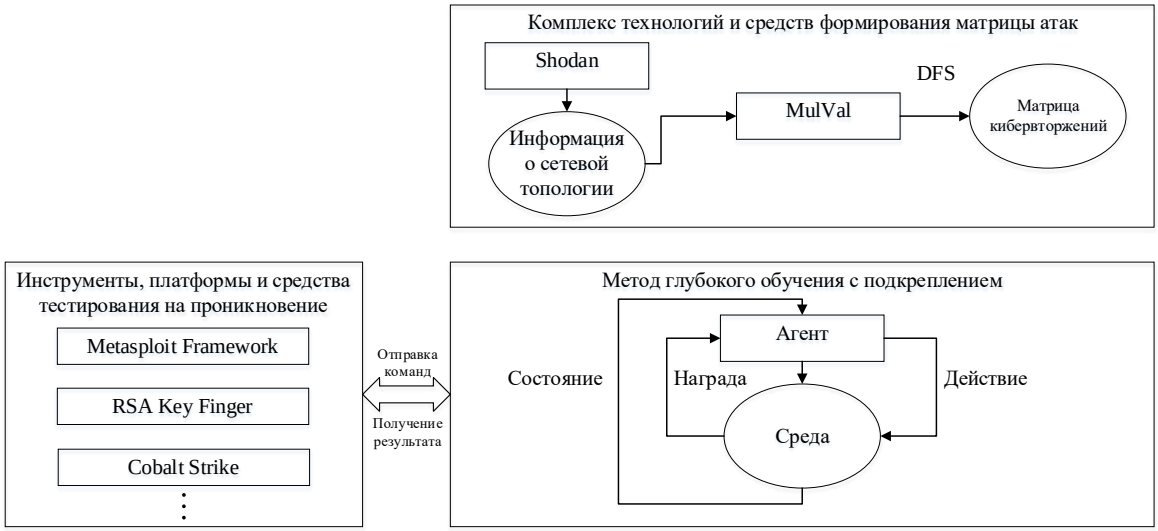

Рисунок 2.6 – Загальна структура методу автоматизованого тестування на проникнення

2.5 Дослідження можливостей платформи Shadov та Mulval

Відомо, що система Shodan призначена для роботи з тіньовими каналами інтернету. Система шукає не веб-ресурси з контентом, а фізичні пристрої, що підключені до інтернету. Такими можуть бути принтери, веб-камери, маршрутизатори, GPS-навігатори та навіть системи технічного обслуговування комерційних підприємств. Основний принцип роботи Shodan полягає у надсиланні запитів на всі публічно доступні IP-адреси та протоколювання їх відгуків. Алгоритм сканування даної системи:

- генерація випадкової IP-адреси;
- вибір випадкового номера порту зі списку доступних у Shodan портів;
- перевірка обраної IP-адреси (порту) та отримання банера;
- повторення кроку 1. Таким чином, система виконує сканування всього адресного простору випадковим чином, щоб забезпечити рівномірне покриття інтернету та запобігти зміщенню даних у будь-який момент часу.

Shodan також підтримує пошук за відомостями про вразливість програмного забезпечення.

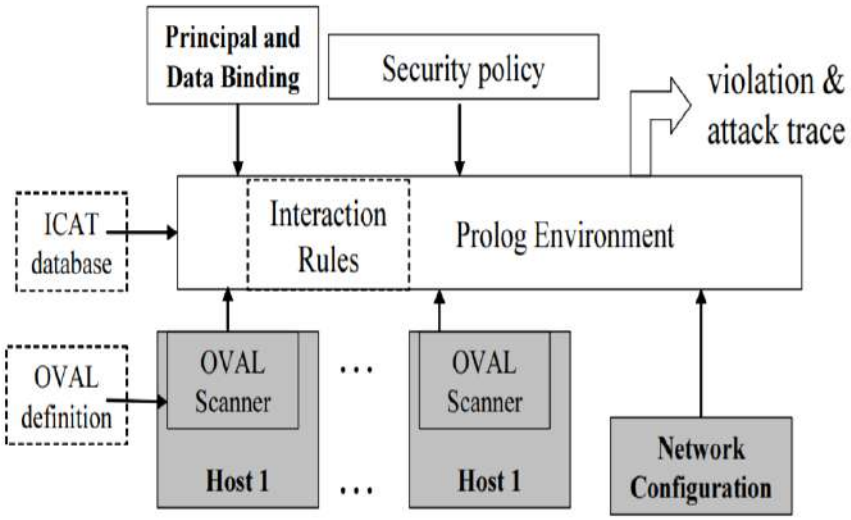

Рисунок 2.7 – Структурна схема фремворку Mulval

Система також дозволяє вибирати кілька критеріїв пошуку та фільтрації даних для моніторингу актуального стану ІВ. До основних фільтрів Shodan відносяться: City/country (фільтрація пристроїв, розташованих у межах заданого міста/країни, наприклад city:minsk); Port (виведення пристроїв із заданим відкритим портом, наприклад port:443); OS (фільтрація пристроїв, які працюють на заданій операційній системі, наприклад, os:linux); Geo (точні вказівки координат розташування пристрою (довгота, широта), наприклад, geo:42.9693,74.1224); Net (пошук пристроїв із заданого діапазону IP-адрес, наприклад net:216.0.0.0/16). Mulval – це широко використовуваний інструмент аналізу мережевої безпеки, який використовує сканер уразливостей для пошуку мережевих уразливостей, а потім генерує графіки атак для аналізу безпеки.

3 МЕТОД АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ НА ПРОНИКНЕННЯ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ DEEP LEARNING

3.1 Розробка методу формування матриці з використанням інструменту Mulval

Отримавши неформальне поняття в 1999 році в роботі Брюса Шнайтера, дерева атак, як методологія опису загроз безпеці, наразі набули широкого поширення. Дерево атак характеризується наступним чином: кореню дерева відповідає мета порушника, вершинам – дії порушника задля досягнення мети. Дії комбінуються за допомогою логічних І та АБО. Вершинам і ребрам можуть призначатися різні числа всіх значень, які характеризують ймовірність успіху, складність, вартість дій порушника. На рис. 3 представлений граф та текстове подання кожного з можливих типів вузлів, І та АБО.

Враховуючи наведені факти, слід зауважити, що перший етап - моделювання дерева атак, є дуже важливим для формування навчальних даних. Ці дані, своєю чергою, надзвичайно важливі використання у алгоритмах глибокого навчання. Для формування вхідних даних у роботі пропонується виконання наступних актуальних етапів.

Перший етап. Збір мережевої інформації для моделювання мережевого середовища за допомогою Shodan.

Другий етап Генерація дерева атак, що відповідає цьому мережному середовищу з використанням платформи Mulval.

Третій етап. Попередня обробка даних (формування матриці атак) адаптації до вимог алгоритмів глибокого навчання.

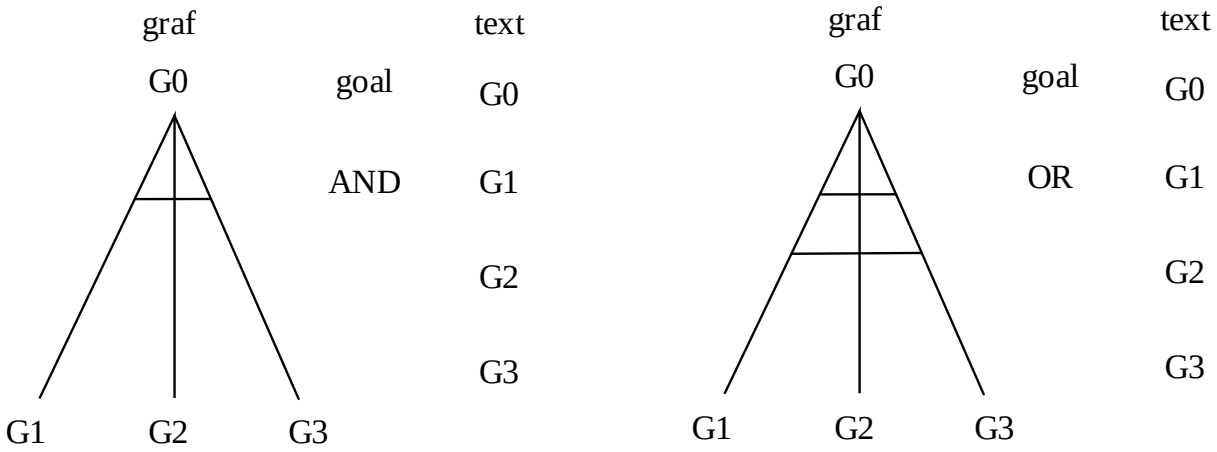

Рисунок 3.1 – Зображення графа та текстового представлення кожного з можливих типів вузлів, І та АБО

На першому етапі формується необхідний практичний запит у системі Shodan, наприклад, запит на інформацію реальних Web-серверах. Приклад даних зібраних через Shodan (за винятком IP-адрес) на рисунку 3.2 На підставі отриманих даних створюється файл-профіль з необхідною інформацією про мережний сайт. Приклад профілю веб-серверів наведено в таблиці 3.2.

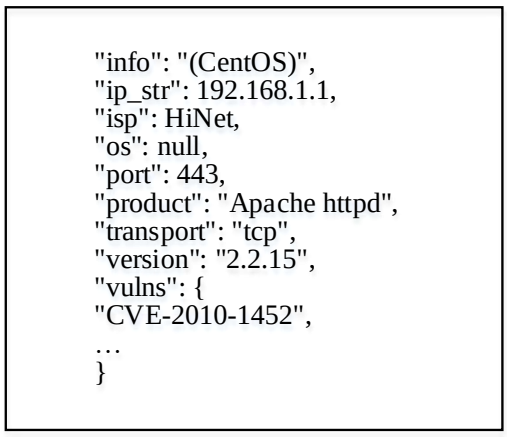

Рисунок 3.2 – Приклад даних зібраних через Shodan

Таблиця 3.1 – Приклад профілю веб-серверів

Product	Port	Protocol	Vulnerability	OS
Apache	80	https	CVE-2010-1452	CentOS
Nginx	8080	https	CVE-2011-0419	Ubuntu
mt-daapd DAAP	3689	Tcp	CVE-2017-9617	FreeBSD

На додаток до представленого в таблиці 3.1 набору даних також створюється файл вразливостей. Дані включають:

- – ідентифікатор номер уразливостей CVE, Microsoft тощо;
- – компонент типу базової оцінки;
- – компонент оцінки придатності exploitablityScore.

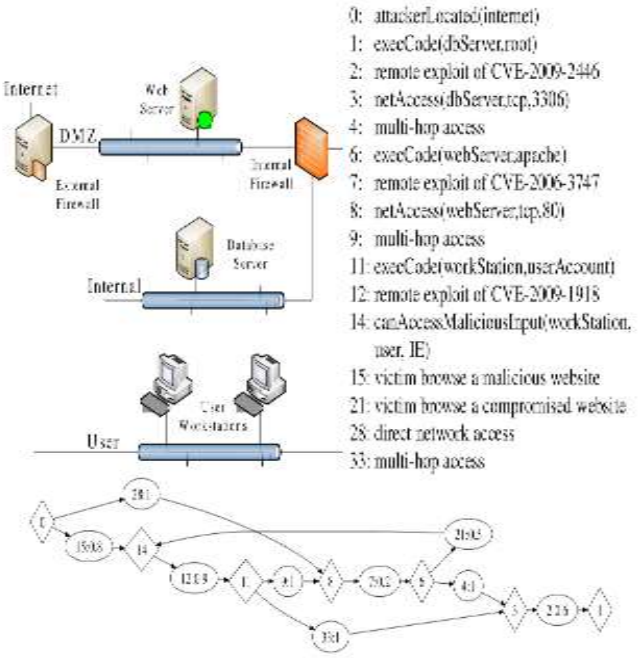

Рисунок 3.3 – Приклад сценарію графа атак, розробленого з використанням інструменту Mulval

На другому етапі генерації дерева атак пропонується використовувати відому платформу Mulval. Модель Mulval ґрунтується на логічному програмуванні та описується за допомогою мови DataLog, яка є діалектом та

підмножиною мови Prolog.

Реалізує цю мову середовище IDE XSB. При генерації на основі Mulval виділяють такі види об'єктів-вершин графа атак – вершини виведення (правило виведення) та вершини фактів (правила – предикати, що визначають значення та вид прототипу дії), приклад генерації дерева атак відображено на рисунку 3.5.

3.2 Алгоритм перетворення дерева атак на матрицю кібервторжень

Наступним складовим елементом методу автоматизованого тестування, що розробляється, на проникнення є алгоритм перетворення дерева атак в матрицю кібервторжень. Для проведення досліджень як прототип був обраний алгоритм перетворення дерева атак у відповідну матрицю, розроблений авторами. Проведене на основі дослідження методу дозволило зробити вибір про обмеження його використання для тестування на проникнення. Це багато в чому пов'язане з нехтуванням авторів такими вхідними даними, як «доступ до файлів», «виконання команд», і концентрація уваги лише на вразливості.

Наведемо нижче опис удосконаленого алгоритму перетворення дерев атак на матрицю кібервторжень.

На першому етапі алгоритму відбувається зіставлення всіх вузлів у дереві атак у матричну форму.

На другому етапі в комплексі зіставляються кількісні оцінки вразливостей безпеки базовій оцінці відповідних вузлів. Розрахунок базової оцінки виконується відповідно до структури CVSS (Common Vulnerability Scoring System). Відповідно до [вікіпедія] оцінки розраховуються за спеціальними формулами на основі метрик і характеризують простоту впровадження експлойту та його впливу на комп'ютерну систему. Як удосконалення пропонується накладати на базову оцінку CVSS деяку заздалегідь визначену оцінку експлуатованості (наприклад, «доступ до

файлів») відповідно до виразу:

$$\text{ovsc} = \text{bsc} \times \frac{\text{expsc}}{5} \tag{3.1}$$

де ovsc – загальна оцінка вразливості безпеки,

bsc – базова оцінка вразливості безпеки,

expsc - оцінка експлуатованості.

Тут же, на цьому кроці, замість впровадження матриці кібервторжень безпосередньо в алгоритм глибокого машинного навчання, передбачається спростити матрицю з використанням модифікованого алгоритму DFS - алгоритму Тар'яна. Цей алгоритм в першу чергу є одним з варіантів пошуку в глибину. Вершини при цьому відвідуються від коріння до листя, а закінчення їх обробки виконується на зворотному шляху. Таке спрощення повної матриці дозволяє вибрати ті вузли, які можна використовувати для досягнення мети атаки.

Після закінчення алгоритму Тар'яна третім кроком формується спрощена матриця кібервторжень, у яку записуються:

- значення оцінки для початкового вузла в першому стовпці;
- значення загальних балів, що характеризують проміжні етапи в проміжних стовпцях;
- значення оцінки для кінцевого вузла в останньому стовпці.

Ці дані будуть використані як вхідні з метою оцінки винагороди в алгоритмі глибокого машинного навчання.

3.3 Deep Learning для визначення оптимальної траєкторії атаки

У методі автоматичного тестування на проникнення метод глибокого машинного навчання з підкріпленням використовується для визначення оптимальної траєкторії атаки шляхом безперервного навчання.

Вхідні дані для методу формуються на основі спрощеної матриці кіберворжень, а прототипом функції активації послужить soft max функція – логістична функція для багатовимірного випадку (3.2):

$$\nu(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{3.2}$$

У процесі навчання агент моделі глибокого машинного навчання з підкріпленням (DQL) виступає в режимі етичного хакера, при цьому система, що таргетується, представляється у вигляді спрощеної матриці кіберворжень. Агент переміщається від вузла до вузла в розробленій матриці до моменту, поки не досягне бажаного результату - «сервера жертви». Бонусні коефіцієнти, що використовуються в процесі моделювання DQL, відповідають значенням оцінки вразливостей виходячи з даних загальної системи оцінки вразливостей (CVSS)

У CVSS базова оцінка інтерпретує ступінь негативних наслідків окремих видів уразливостей, тоді як оцінка експлуатованості фіксує можливість використання цієї конкретної вразливості. Для отримання найбільш адекватного з практичного погляду результату доцільно провести математичне зважування цих оцінок шляхом множення відповідні коефіцієнти. Значення коефіцієнтів визначається емпіричним шляхом з урахуванням думки експертів у сфері кібербезпеки.

3.4 Експериментальні дослідження методу автоматизованого тестування на проникнення

Для оцінки працездатності та ефективності розробленого методу автоматизованого тестування на проникнення програмного забезпечення було проведено серію експериментів. При цьому система, що таргетується,

була представлена у вигляді топологічної структури комп'ютерної мережі.

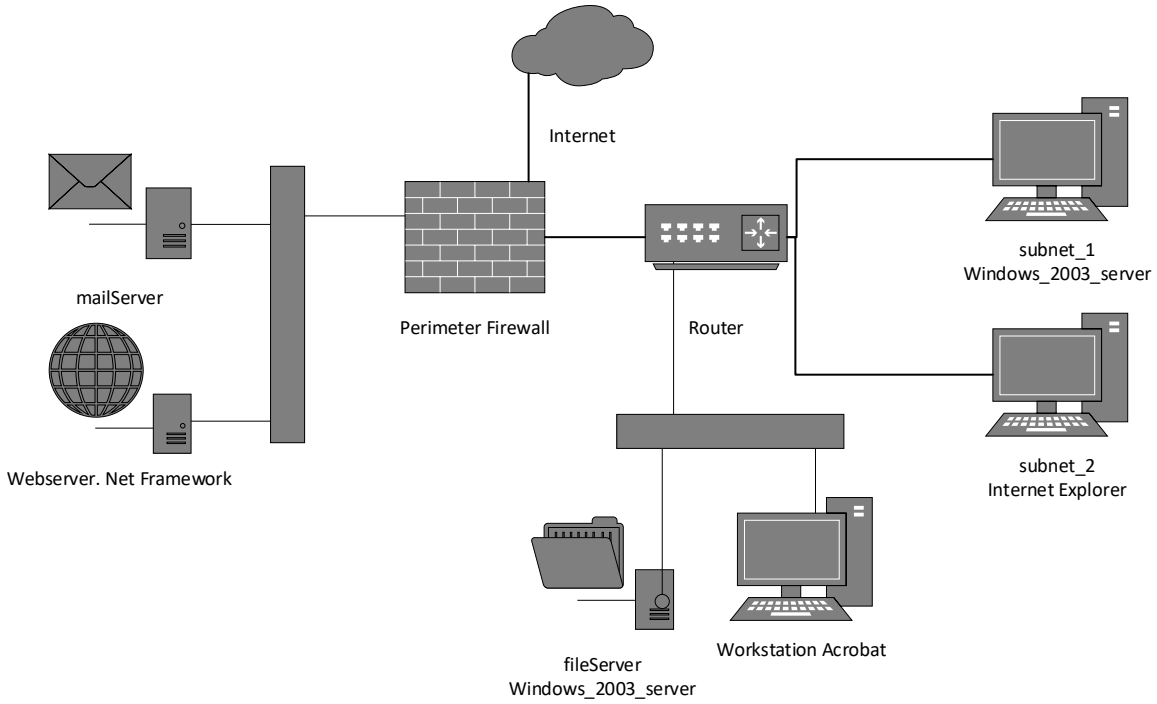

Рисунок 3.4 – Топологічна структура досліджуваної комп'ютерної мережі

З представленої схеми видно, що система, що таргетується, являє собою сукупність трьох різних сервісів, включаючи веб-сервер і поштовий сервер в одній підмережі, а також файловий сервер в іншій підмережі. Клієнт у першій підмережі керується системою Windows 2000, а клієнт другої підмережі – системою Internet Explorer.

У поданій схемі також до однієї підмережі з файловим сервером підключено робочу станцію з програмним забезпеченням Acrobat.

Як засіб захисту комп'ютерної мережі від кібервторжень у схемі використовують міжмережевий екран.

Правила підключення до цього засобу є наступними:

- агент перебувати у зовнішній мережі і має можливість доступу до веб-серверу через протокол HTTP і відповідний HTTP-порт;
- між веб-сервером та робочими станціями мережі є дуальний зв'язок;
- веб-сервер та файловий сервер з'єднується за протоколом NFS;

- вихід файлового сервера першої та другої підмереж у зовнішню мережу здійснюється за протоколом HTTP;
- файловий сервер та робоча станція з'єднуються за протоколом NFS.

Проведені дослідження показують, що з даної топології характерно кілька вразливостей ПЗ. Наприклад, на думку експертів, одна з найбільш небезпечних уразливостей Веб-сервера (CVE-2020-1198) є вразливістю переповнення буфера в модулі mod_uwsgi. Атакуючий може викликати відмову в обслуговуванні та виконати довільний код.

Таблиця 3.2 – Уразливості досліджуваної системи

№	Апаратні засоби	Список вразливостей
1	Веб-сервер	CVE-2020-1198
2	Перша підмережа	CVE -2016-0189
3	Друга підмережа	CVE-2020-1380
4	Файл-сервер	CVE- 2010-0492

Клієнти першої підмережі можуть бути схильні до атак зловмисника з причин наявності вразливості CVE -2016-0189 у відомому компоненті Vbscript.dll. Ця вразливість дозволяє хакеру виявити довільний код. Клієнти другої підмережі схильні до атак віддаленого виявлення коду в скриптовому движку, що поставляється у складі браузера Internet Explorer (уразливість CVE-2020-1380). Файл-сервер містить уразливість CVE-2010-0492, яка пов'язана з ОС Windows 2003 SP 2. При цьому зловмисник має можливість реалізації атаки в обхід обмежень вихідної IPV4 адреси.

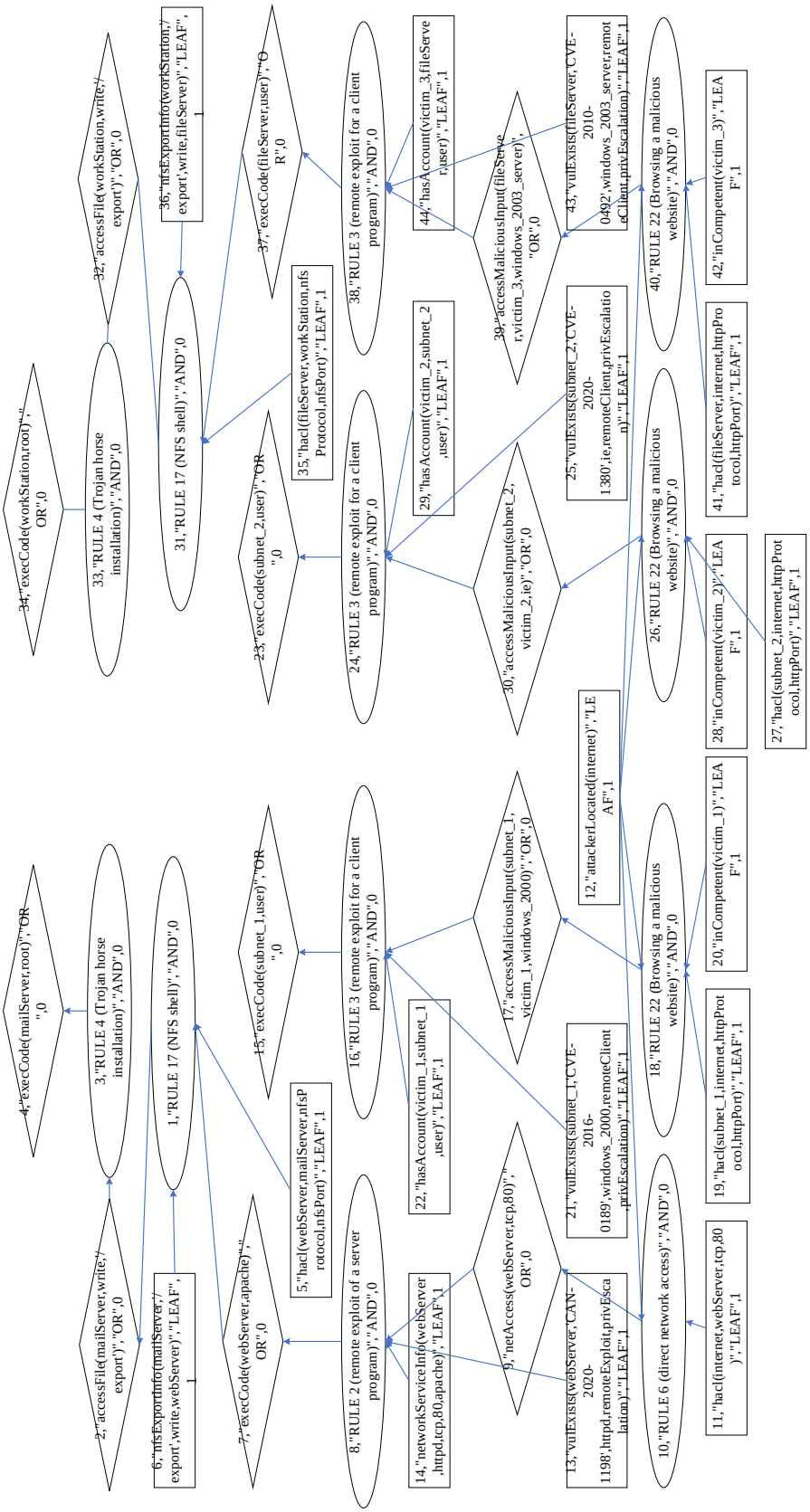

Рисунок 3.5 – Граф атак на досліджувану комп'ютерну систему

За допомогою Mulval сформовано графову схему з трьома різними видами вершин. Вершини, зображені як прямокутників, характеризують конфігурацію системи. Вершини у вигляді ромба представляють потенційні привілеї або доступ, які можуть отримати зловмисники в системі. Еліптичні вузли пов'язують попередні умови з умовами.

Наступним кроком дослідження стала процедура спрощення графа атаки. На рисунку 3.6 представлений уточнений граф атак із використанням алгоритмів розроблених авторами досліджень [5].

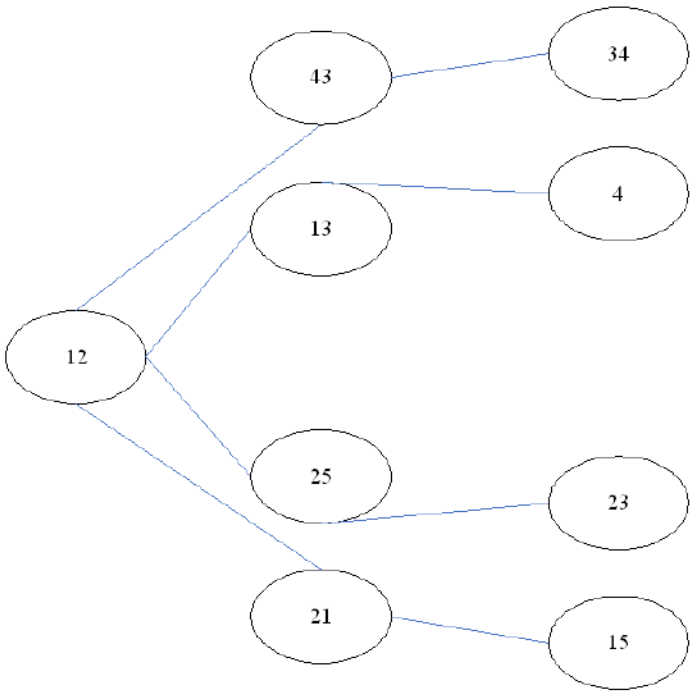

Рисунок 3.6 – Уточнений граф атак

Представлений на рисунку 3.6 граф можна взяти основою моделювання з допомогою методу глибокого машинного навчання з підкріпленням. На цьому малюнку місце розташування агента, вразливості та мети агента-зловмисника представлені у вигляді вершин, а кожне ребро графа ілюструє можливості агента та його дію. Передбачається, що атакуючий може переміщатися між вузлами в будь-якому напрямку графа, доки не досягне

одного з цільових станів.

Слід зазначити, що у розробленому методі автоматизованого тестування на проникнення загальна система оцінки вразливостей (CVSS) також використовується визначення винагород відповідно до виразом 1.

Якщо агент-зловмисник скористається вразливістю, загальний бал, пов'язаний з цією вразливістю, буде розглядатися як величина винагороди. Якщо зрештою агент досягає мети, то нагорода вважатиметься максимальною. Додаткові кільцеві ребра додаються до вершин мети з нагородою 100, оскільки агент, який досяг зловмисної мети, залишається там назавжди. Модель DQL з винагородою представлена на рисунку 3.7.

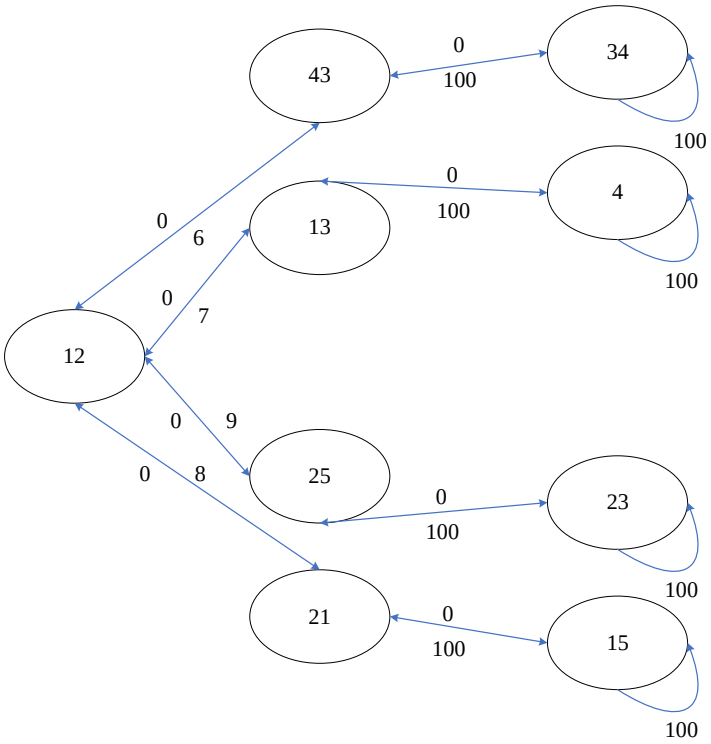

Рисунок 3.7 – Модель DQL з винагородою

Для прикладу проведемо моделювання ситуації, коли агент-зловмисник розташовується в одній із точок комп'ютерної мережі. На рисунку 3.7. дана точка позначається вузлом 12. Даний агент має можливість переходу на інші вузли (13, 21, 25 і 43), при цьому кожна його дія отримує бонус відповідно до метриків уразливостей.

Нехай агент проводить зловмисні дії, починаючи з вузла 43.

У цьому випадку існує два варіанти подій:

- перейти у вузол 34, який є кінцевою метою кібератаки. У цьому агент отримує 100 балів;
- повернутися у вихідне положення та додатково бали не отримати.

Сформуємо узагальнену матрицю станів як таблиці 3.2. У таблиці 3.3 рядки є стан системи стовпці – дії. Наприклад, точка матриці 12, 43 ілюструє перехід агента з вузла 12 в вузол 43, при цьому отримання бонусів у розмірі 6 балів. Алгоритм оцінки поведінки зловмисника представляємо як блок – схеми (рисунок 3.8).

Таблиця 3.3 – Узагальнена матриця станів

	12	13	21	25	43	4	15	23	34
12	-1	7	8	9	6	-1	-1	-1	-1
13	0	-1	-1	-1	-1	100	-1	-1	-1
21	0	-1	-1	-1	-1	-1	100	-1	-1
25	0	-1	-1	-1	-1	-1	-1	100	-1
43	0	-1	-1	-1	-1	-1	-1	-1	100
4	-1	0	-1	-1	-1	100	-1	-1	-1
15	-1	-1	0	-1	-1	-1	100	-1	-1
23	-1	-1	-1	0	-1	-1	-1	100	-1
34	-1	-1	-1	-1	0	-1	-1	-1	100

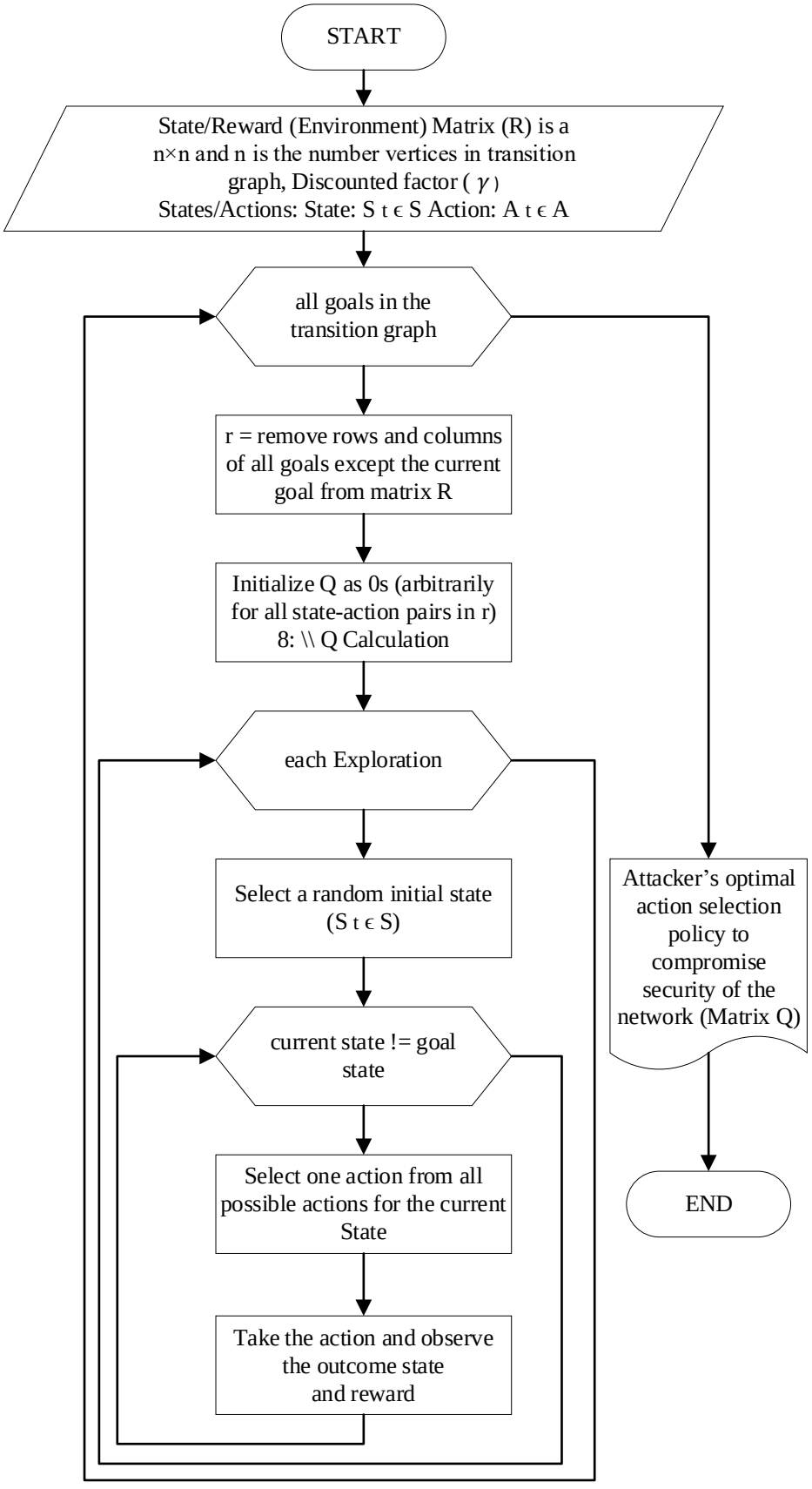

Рисунок 3.8 – Блок-схема алгоритму оцінки поведінки зловмисника

Слід зазначити використання графа переходів матриці станів у цьому алгоритмі. З метою практичної реалізації відповідно до алгоритму на рисунку 3.9 з матриці стану таблиці 3.4. формується підматриця уточнень стану та додаткових бонусів агенту-зловмиснику, яку можна проілюструвати у вигляді таблиці 3.4.

Таблиця 3.4 – Підматриця уточнень стану та додаткових бонусів агенту-зловмиснику

	12	13	21	25	43	4
12	-1	7	8	9	6	-1
13	0	-1	-1	-1	-1	100
21	0	-1	-1	-1	-1	-1
25	0	-1	-1	-1	-1	-1
43	0	-1	-1	-1	-1	-1
4	-1	0	-1	-1	-1	100

Використовуючи цю матрицю, сформуємо діаграму бонусів агента-зловмисника для однієї із заявлених цілей. На рисунку 3.8 проілюстровано граф бонусів агенту при переході в цільовий стан 1. Як видно з рисунка 3.9 перехід зі стану 12 в стан 13 дає максимальну винагороду 408. Перехід зі стану 13 стан, яке характеризує кінцеву мету зловмисника, формалізується максимальним бонусом 500.

Звідси перехід 12, 13, 4 є кращим з погляду бонусів, що заробляються.

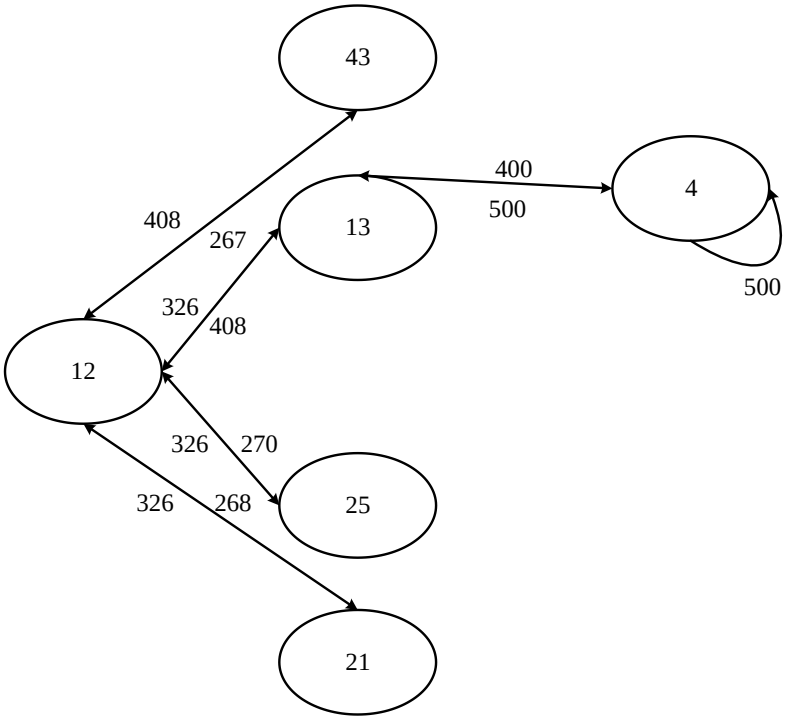

Рисунок 3.9 – Граф бонусів агенту при переході у цельовий стан 1

Ряд інших практичних прикладів результатів використання розробленого методу автоматизованого тестування на проникнення для етичної компрометації цілей у комп'ютерній системі представлені в таблиці 3.5.

Таблиця – 3.5. Практичні приклади результатів використання методу автоматизованого тестування на проникнення

Імітатор вузла зловмисника	Ціль	Максимальна винагорода
12	4	908
12	15	907
12	23	906
12	34	906

Наведена таблиця демонструє можливості тестувальника в обліку максимальної шкоди. Це дозволяє ранжувати вразливості в системі та усувати їх залежно від їхнього пріоритету. Що, своєю чергою, підвищує

ефективність процесу тестування.

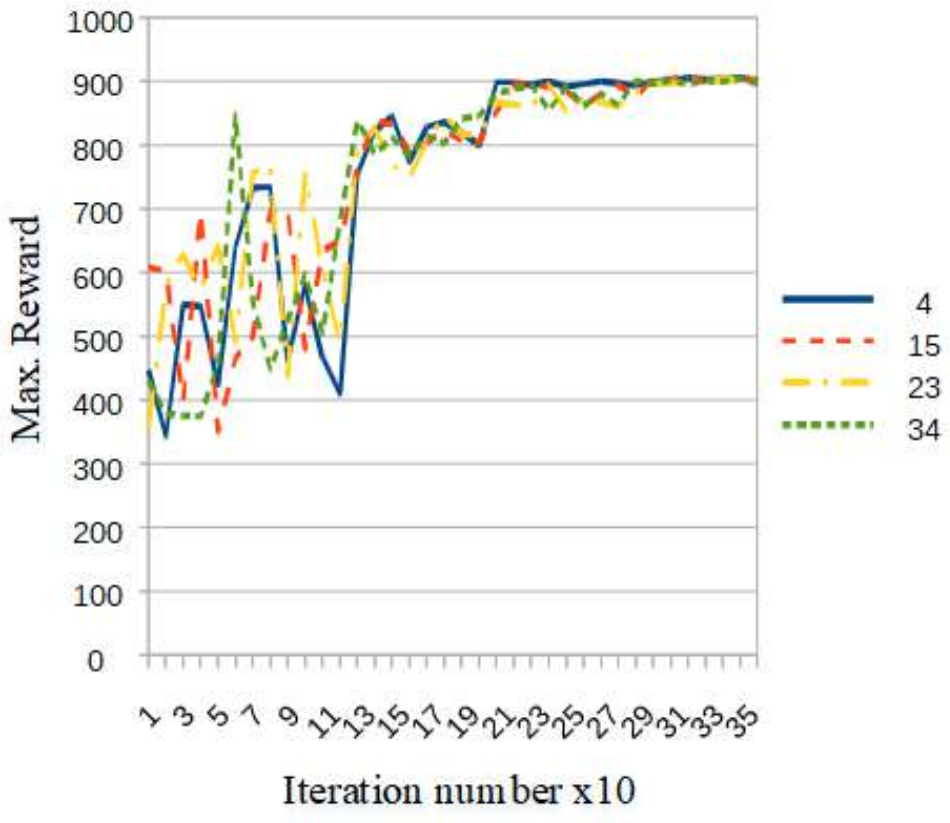

Рисунок 3.10 - Графік залежності винагороди за кожну ітерацію виконаного тесту кількості ітерацій

Для підтвердження практичної можливості використання розробленого методу автоматизованого тестування на проникнення було проведено дослідження збіжності моделі. Цей факт наочно проілюстровано на рисунку 3.10, наведено графік залежності винагороди за кожну ітерацію виконаного тесту від кількості ітерацій. В експерименті розглядалося 1000 ітерацій. Однак, як видно з графіка, для сходження моделі потрібно 350 ітерацій. Максимальна винагорода в цьому прикладі досягає позначки 908.

ВИСНОВКИ

Розроблено метод автоматичного тестування на проникнення. Відмінною особливістю методу є комплексне використання пошукової системи Shodan, платформи аналізу мережевої безпеки MulVal, а також даних про вразливість програмного забезпечення – CVE для отримання вхідних даних та побудови реалістичних сценаріїв атак та перевірки у рамках технології глибокого навчання із підкріпленням. Це дозволило згенерувати дерево атак для різних процедур навчання та провести оптимізацію відповідних сценаріїв автоматичного тестування безпеки програмного забезпечення.

При дослідженні, відповідно до методу глибокого навчання з підкріпленням, були використані оцінки винагороди, що призначаються кожному вузлу відповідно до рейтингу CVSS. Це дозволило зменшити дерева атак та визначити атаку з більшою ймовірністю виникнення.

Для оцінки застосовності методу проведено експеримент та згенеровано дерево атак, також сформовано сценарій тестування та навчання. Підтверджено факт, що навіть за невеликої кількості сценаріїв навчання результати моделювання досягають значення 0.9 щодо найбільш раціонального шляху атаки.

Розроблений метод є ефективним рішенням аналізу програмного забезпечення, оскільки дає можливість тестувальнику вибору обґрунтованої політики етичного хакінгу та дій, спрямованих на пом'якшення негативних факторів можливих кібератак.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кучук Н.Г., Дубовик Т.І., Лисиця Д.О. Автоматизованне тестування застосунків з використанням deep lerning. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : Матеріали дванадцятої міжнародної науково-технічної конференції. – Баку: ВА ЗС АР; Харків : НТУ «ХПІ»; Київ : НАУ; Харків : ДП «ПДПРОНДІАВІАПРОМ»; " Жиліна : університет, 2022. – С. 22.

2. Web Application Performance: 7 Common Problems and How to Solve Them – Режим доступу : https://stackify.com/web-application-problems/ Ian Molyneaux The Art of Application Performance Testing, 2nd Edition / Ian Molyneaux – O'Reilly Media, Inc.

3. Performance Testing : What is, Types, Metrics & Example – Режим Тестування. фундаментальна теорія: https://dou.ua/forums/topic/13389/Dr. Dobbs Journal, chneier B. Attack Trees. December 1999 – С.3

4. Camtepe S., Yener B. A Формальний спосіб для Attack Modeling and Detection. http:// cs.rpi.edu/research/pdf/06- 01.pdf

5. Phillips C., Swiler L.. A Graph-Based System for Network-Vulnerability Analysis // In Proceedings of New Security Paradigms Workshop, Charlottesville, VA, 1998.

6. Tvoroshenko I., and Maksimenko H. To the question of analysis of existing mechanisms of web application testing, Abstracts of I International scientific- practical conference «Problems of modern science and practice» (September 21-24, 2021). Boston, USA, pp. 403-409. 59.

7. Tvoroshenko I., and Maksimenko H. Research of regression and modular testing of web applications, Abstracts of IV International scientificpractical conference «Science, theory and practice» (October 12-15, 2021). Tokyo, Japan, pp. 406-411.

8. Brown A.W. Large-scale Component-Based Development [Text] / A.W.

Brown. - Prentice-Hall, 2010. - 300 p.

9. Cohn M. Agile Estimating and Planning [Text] / Mike Cohn. - Prentice Hall, 2015. - 368 p.

10. Didkovska M. Criteria for integration testing of component-based software [Text] / M.Didkovska // Электроника и связь. - К.: 2004. - №23. - С. 90-94.

11. Myers G.J. The Art Of Software Testing [Text] / G.J. Myers - New York: John Wiley & Sons, Inc., 2004. - 254 p. - ISBN 0-471-46912-2.

12. Offutt J. Generating tests from UML specifications [Electronic resource] / J. Offutt, A. Abdurazik // Second International Conference on the Unified Modeling Language. - Fort Collins, CO, IEEE Computer Society Press, 1999. - С. 416-429

13. Page-Jones M. Fundamentals of Object-Oriented Design in UML / Page-Jones M. - New York: Addison-Wesley, 2000. - 435 p.

14. Patton R. Software Testing [Text] / R. Patton. - 2nd Edn. - Indianapolis: Sams, 2005. - 408p.

15. Кріспін Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд : пер. з англ. / Л. Кріспін., Д. Грегорі – М.: «Вільямс», 2011. – 463 с. – ISBN 0-471-46912-2.

16. Липаев В. В. Тестирование компонентов и комплексов программ: підруч. / Липаев В. В. – М.-Берлін: Діректт-Медіа, 2015. – 528с. – ISBN 978-5-89638-115- 0.