

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЗАЇКА АННА СЕРГІЇВНА

Допускається до захисту:  
Завідувач кафедри  
інформаційних технологій,  
к.т.н., доцент,  
\_\_\_\_\_ Нескородева Т. В.  
«\_\_» \_\_\_\_\_ 20\_\_ р.

СУЧАСНІ МЕТОДИ ЗАХИЩЕНОСТІ ТА  
БЕЗПЕКИ ПРОГРАМНИХ СИСТЕМ

Спеціальність 125 Кібербезпека  
Кваліфікаційна (бакалаврська) робота

Керівник:

Загоруйко Л. В.,  
к.т.н., доцент кафедри  
інформаційних технологій

\_\_\_\_\_  
(підпис)

Оцінка : \_\_\_\_\_ / \_\_\_\_\_ / \_\_\_\_\_  
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: \_\_\_\_\_  
(підпис)

Вінниця - 2021

## АНОТАЦІЯ

**Загоруйко Л.В.** Сучасні методи захищеності та безпеки програмних систем. Спеціальність 125 “Кібербезпека”, Донецький національний університет імені Василя Стуса, Вінниця, 2021.

У бакалаврській роботі досліджено методи захищеності та безпеки програмних систем. Показано принципи безпечного користування програмних систем. Встановлено розвиток у напрямку захисту від основних вразливостей програмного забезпечення.

Ключові слова : Методи захищеності, кібернапади, віруси, формальні методи захищеності, *DDoS* атаки.

Табл. 3. Рис. 12. Бібліограф.: 47 найм.

**Zagoruyko L.V.** Modern methods of security and protection of software systems. Specialty 125 “Cybersecurity”. Vasyl` Stus Donetsk National University, Vinnytsia 2021.

In the bachelor's work the methods of security and safety of software systems are investigated. The principles of safe use of software systems are shown. Development in the direction of protection against the main vulnerabilities of the software is established.

Keywords: Security methods, cyber attacks, viruses, formal security methods, *DDoS* attacks.

Tabl. 3. Fig. 12. Bibliography.: 47 items.

## Зміст

АНОТАЦІЯ.....	1
Вступ.....	3
Основні терміни та визначення .....	5
Розділ 1. ОСНОВНІ ОСОБЛИВОСТІ МЕТОДІВ ЗАХИЩЕНОСТІ ПРОГРАМНИХ СИСТЕМ.....	6
1.1 Основні поняття про кібернапади та їх класифікація .....	6
1.2 Типи мережевої архітектури .....	11
1.3 Засоби захисту мереж та середовищ даних .....	13
Розділ 2. МЕТОДИ ЗАХИЩЕНОСТІ МЕРЕЖ І СЕРЕДОВИЩ ТА ЇХ КЛАСИФІКАЦІЯ.....	16
2.1 Основні методи захищеності мереж та їх властивості.....	16
2.2 Формальні методи, які застосовуються для визначення атак .....	20
2.3 Систематизація та побудова формальних методів .....	25
2.4 Інсерційний підхід до методу захищеності та його особливості.....	28
Розділ 3. ФУНКЦІОНАЛЬНА ЧАСТИНА МЕТОДУ ЗАХИЩЕНОСТІ ТА БЕЗПЕКИ ПРОГРАМНИХ СИСТЕМ.....	31
3.1 Класифікація та визначення <i>DDoS</i> атак.....	31
3.2 Методи захищеності <i>DDoS</i> атак та їх реалізація .....	37
ВИСНОВОК .....	43
Список літератури.....	44

## Вступ

Проблема безпеки та захищеності програмних ресурсів є однією з найбільш актуальних проблем, що активно вивчаються, також поява мережових програм-зидників (*ransomware*) що, свідчить про високий рівень досконалості технологій атак. Зловмисники дедалі більш вправно уникають виявлення та ефективно використовують прогалини в системі безпеки. Важливим буде навести огляд основних понять [1] та нагальних проблем у галузі захищеності та безпеки, а також приділити увагу засобам кіберзахисту (*cybersecurity*), побудованим з використанням формальних методів на основі сучасних алгебраїчних теорій.

Безпека даних стала серйозною проблемою в 1980-х роках, коли почали формуватися комп'ютерні клуби, а також шкідливе програмне забезпечення. Найперші віруси були помилками, помилкою в алгоритмі, нездатним до самовідтворення. Інтерес до вірусів, особливо шкідливим, продовжував зростати, а вже у 1985 році німецький комп'ютерний інженер по імені Ральф Бергер виступив з програмною промовою для комп'ютерного клубу Хаосу (в даний час найбільшого хакерського клубу Європи), закликавши інших досліджувати цей новий аспект комп'ютерного програмування.

Перший навмисно шкідливий комп'ютерний вірус, іменований як мозок, була розроблена в 1986 році і призначалася для дискет, аспект вірусу був розроблений двома братами, Амджад і Басита Фарук Альва, заявивши, що вони стурбовані, що їх програмне забезпечення копіювалося. Мозок працює на комп'ютерах *IBM PC*, змінюючи дискет, та її завантажувальний сектор вірусом, а сам вірус уповільнює роботу диска.

Віруси і хакери створювали хаос із загрозливою швидкістю в 1990-х роках, були зроблені спроби заблокувати несанкціоноване проникнення в комп'ютерні системи, а персоналу комп'ютерів були випущені попередження і пам'ятки про способи виявлення вірусів. Ці зусилля включали створення ізольованих резервних копій, тому дані, якщо вони були пошкоджені на



комп'ютері, все ще були доступні в окремому місці. Програмне забезпечення швидко стало популярним методом зберігання даних резервних копій, паролі та шифрування стали популярними з точки зору блокування хакерів.

Для ранніх комп'ютерних програм не було необхідності в захисті авторських прав. Комп'ютерів було небагато, і велика частина програмного забезпечення була розроблена спеціально для внутрішніх додатків. Лише на початку 1960-х комп'ютерні програми стали активно продавати не тільки виробники комп'ютерів, але і виробники програмного забезпечення. До того, як програмне забезпечення стало широко продаватися, було легко захиститись за допомогою контракту або ліцензійної угоди на будь-яку комп'ютерну програму. Закон про авторське право 1976 р вступив в чинності 1 січня 1978 року, ясно дав зрозуміти, що Конгрес має намір охороняти програмне забезпечення авторським правом.

У поточному десятилітті, масштаби витоків даних і кібератаки виросли, а тактика і стратегії доступу змінюються.

**Метою роботи** огляд і дослідження сучасних методів захищеності та безпеки програмних систем.

**Завданнями** досліджень є

- проаналізувати особливості методу захищеності програмних систем ,
- розглянути принципи безпечного користування програмними системами,
- запропонувати розвиток у напрямку захисту від основних вразливостей програмного забезпечення.

## Основні терміни та визначення

*Flood* – атака, що пов'язана з великою кількістю запитів, що є неправильно сформовані або не мають суті. Призводить до відмови відмови у роботі.

*Exploit* - це атака, що експлуатує деяку вразливість. Атака починається з проникнення у віддалений комп'ютер, який може бути як настільним, так і складовою частиною мережі або хмарного середовища.

*ПК* – персональний комп'ютер, “машина”.

*P2P* - тип мережевої архітектури є однорангова мережа.

Модель *OSI* – стандартна модель для взаємодії відкритих систем, що має сім рівнів.

*DDoS (Distributed Denial of Service)* – зловмисна спроба порушити трафік мережі, шляхом затримання об'єкта завдяки потоком Інтернет.

*IPv6* – (*Internet Protocol version 6*) – нова версія IP протоколу, версії шостої.

*IPv4 (Internet Protocol version 4)* – четверта версія інтернет – протоколу.

*Snyfing (Spoofing attack)* - це момент коли програма маскується під схожу на себе, за допомогою підробки даних та отримує перевагу незаконну.

## **Розділ 1. ОСНОВНІ ОСОБЛИВОСТІ МЕТОДІВ ЗАХИЩЕНОСТІ ПРОГРАМНИХ СИСТЕМ**

### **1.1 Основні поняття про кібернапади та їх класифікація**

Організації все частіше зберігають, обробляють і передають найбільш важливу інформацію, використовуючи програмно-інтенсивні системи, безпосередньо підключені до Інтернету. Фінансові операції приватних осіб розкриваються через Інтернет за допомогою програмного забезпечення, використовуваного для здійснення покупок, банківських операцій, сплати податків, покупки страховки, інвестування, реєстрації дітей в школі і приєднання до різних організацій і соціальних мереж. Підвищена вразливість, пов'язана з глобальним зв'язком, зробила конфіденційну інформацію і програмні системи, які її обробляють, більш уразливими для ненавмисного і несанкціонованого використання. На даний момент йде ера інформаційної війни, кібертероризму, комп'ютерних злочинів та кібернападу. Терористи, організована злочинність та інші злочинці націлені на весь спектр систем з інтенсивним програмним забезпеченням і завдяки невдалій людській винахідливості, успішно пробираються до цих систем. Більшість таких систем недостатньо стійкі до атак або атак, щоб протистояти їм.

Кібернапад або хакерська атака — це сукупність дій зловмисників (хакерів) з метою втручання в роботу програмної системи для захоплення даних, отримання контролю або виведення з ладу ресурсів комп'ютерного середовища. Атака використовує вразливість системи, спричинену невдалою архітектурою або помилкою в коді.

Програмні системи, які мають помилки, є вразливими для атак. Наприклад, база даних операційної системи Linux має понад 90000 відкритих помилок. Проблема, пов'язана з можливим використанням цих помилок як



вразливостей для атак, є більш гострою, ніж питання виправлення помилки. Термін «*exploit*» за визначенням — це атака, що експлуатує деяку вразливість. Атака починається з проникнення у віддалений комп'ютер, який може бути як настільним, так і складовою частиною мережі або хмарного середовища. Відповідно атаки класифікують за об'єктами нападу (настільний ПК, веб-атаки, хмарне середовище, блокчейн). Атаки можуть використовувати зловмисні програми, які активізуються під час проникнення у комп'ютер користувача. Основні типи зловмисних програм відповідно до способів їхнього проникнення показані у табл. 1.1 [2].

Таблиця 1.1  
Тип та опис програм кібернападу

Тип програми кібернападу	Опис програми
<b>Вірус</b>	Програма, що може способом приєднання себе до інших програм, виконувати використання її в середовищі, яке являється вже “інфікованим”.
<b>Хробак</b>	Ця програма ( <i>Worm</i> ) може працювати та розповсюджувати свої копії на інших комп'ютерних ресурсах, на відмінну від вірусу, вона виконується незалежно.
<b>Троян</b>	Напад програми, що зовні має вигляд корисного додатку, а зловмисну дію виконує наприкінці свого життєвого циклу.
<b>Шпигун</b>	Ця програма ( <i>Spyware</i> ) відслідковує дії користувача, а саме з метою отримання важливої інформації. Такої як, пароль, номер рахунків, особисті дані та інше.
<b>Вимагач</b>	Програма ( <i>Ransomware</i> ), що таємно



	встановлюється на цифровий носій користувача, з метою шифрування даних , а в подальшому вимагання викупу за дешифрування.
<b>Підробка</b>	Саме ця програма ( <i>Scareware</i> ) призначена для того, щоб ошукати користувача, який має намір щось купити. Таким може бути наприклад потенційно небезпечний антивірусний захист.
<b>Імітатор</b>	Програма ( <i>Bot</i> ), що імітує дії користувача за допомогою відповідних інтерфейсів. Це відбувається за допомогою поширення непотрібної реклами або перевантаження інтернет джерела. Прикладом такої програми є <i>DDoS</i> - атака.
<b>Схованки</b>	Набір програмних засобів ( <i>Rootkits</i> ), які дають змогу неавторизованому користувачеві, отримати контроль над системою без виявлення, з метою викрадення даних.

Дефекти програмного забезпечення, включно з помилками кодування, такі як переповнення буфера, недоліки конструкції, несумісна обробка помилок, зустрічаються усюди. Зловмисники, а також шкідливий код, який вони використовують для отримання несанкціонованого доступу і запуску атак, можуть поставити під загрозу систему, скориставшись дефектами програмного забезпечення[3].

Атаки на веб-сайти або веб-сервіси, розподілені та локальні мережі, хмарне середовище враховують та використовують особливості, пов'язані з транспортуванням даних до мережеских вузлів. Проблеми безпеки в цій галузі зумовлені різними типами атак, які можуть застосовуватися для порушення нормального функціонування мережеских систем (активні атаки) або викрадення інформації, переданої мережею (пасивні атаки). Існують такі типи

мережевих атак [4], які у той чи інший спосіб порушують нормальне функціонування мережевих систем (уповільнення та відмови, втрати і спотворення інформації, перехоплення інформації). Відомі типи веб-атак наведено у табл. 1.2.

Таблиця 1.2  
Види атак та їх опис

Назва атаки	Опис атаки
<b>Підробка</b>	Стороння програма, яка використовує фальшиву ідентифікацію, змушуючи систему, яку атакують.
<b>Модифікація</b>	Зловмисна система, що змінює маршрутизацію в мережі.
<b>Тунель атака</b>	Система, що атакує, отримуючи пакет даних, а потім спрямовує його в інше місце, імітуючи для системи, що атакують, найкоротший шлях в мережі.
<b>Фабрикація</b>	Генерація, яка зловмисною системою фальшивого повідомлення для мережевої маршрутизації, що порушує роботу пристроїв-маршрутизаторів.
<b>Відмова в обслуговуванні</b>	Уповільнення, порушення роботи системи, яку атакують, за рахунок перезавантаження мережевого трафіку, надмірної кількості запитів, що надходять у систему.
<b>Вигрібна яма</b>	Створення перешкод, що не дають змогу системі, яку атакують, отримувати повну і коректну інформацію, метою є

	вибіркова модифікація та витрата інформація.
<b>Сивілла</b>	Атака, що використовує безліч зловмисних вузлів мережі, під час якої один такий вузол передає секретні ключі іншим.
<b>Аналіз трафіку</b>	Здійснення аналізу обсягу даних, переданих між вузлами, що атакуються.
<b>Підслуховування</b>	Атаки, що часто здійснюються у мережах мобільного зв'язку, з метою викрадення даних, важливої інформації, зокрема секретних ключів.
<b>Моніторинг</b>	Перехоплення важливої інформації в мережі, та без її модифікації.
<b>Чорна діра</b>	Система, що атакує, використовує протоколи маршрутизації, для того, щоб представити себе найкращим вузлом для маршрутизації даних для системи, яку атакують.
<b>Поспих</b>	Система, що атакує, здійснює підміну пакетів, які передаються між вузлами, що атакуються, а потім дублює ці пакети знову, створюючи враження високонавантаженої системи.
<b>Повтор</b>	Система, що атакує, може повторювати мережеві пакети, та затримувати їх, а в процесі оброблення, викрадати конфіденційну інформацію.
<b>Візантійська атака</b>	Вузли системи, що атакує, вбудовуються між вузлами, що атакуються,



	використовуючи неоптимальні маршрути, створюючи цикли в маршрутах, або вибірково “втрачаючи” пакети.
<b>Розкриття розшифровування</b>	Збір інформації про вузли, що атакуються, і маршрути, моніторинг трафіка з подальшим застосуванням інших видів атак.

## 1.2 Типи мережевої архітектури

Архітектура комп'ютерної мережі визначається як фізичний і логічний дизайн програмного забезпечення, обладнання, протоколів та засобів передачі даних. Можна виділити два основних типи мережевої архітектури. Такі як:

- Мережа клієнт – сервер
- Однорангова мережа

Архітектура клієнт-сервер (клієнт / сервер) - це мережева архітектура, в якій кожен комп'ютер або процес в мережі є клієнтом, або сервером.

Сервери – це комп'ютерна програма або пристрій, що надає послуги іншій комп'ютерній програмі та її користувачу, так званому “клієнт”. Клієнти це комп'ютери або робочі станції, на яких саме користувачі запускають додатки [5]. Архітектура клієнт – сервер, являється однією з найрозповсюдженішої архітектури. Метою її є коли один або кілька комп'ютерів діють як сервер, які в свою чергу пропонують послуги іншій частині мережі, так званим “клієнтам”, як показано на рис. 2.1.

Перевагами клієнт – серверної мережі є такі аспекти:

- 1) Ресурси та їх безпека даних контролюється через сервер.
- 2) Не обмежується малою кількістю комп'ютерів.
- 3) Доступ до самого сервера, можна отримати де завгодно, а також на декількох платформах.

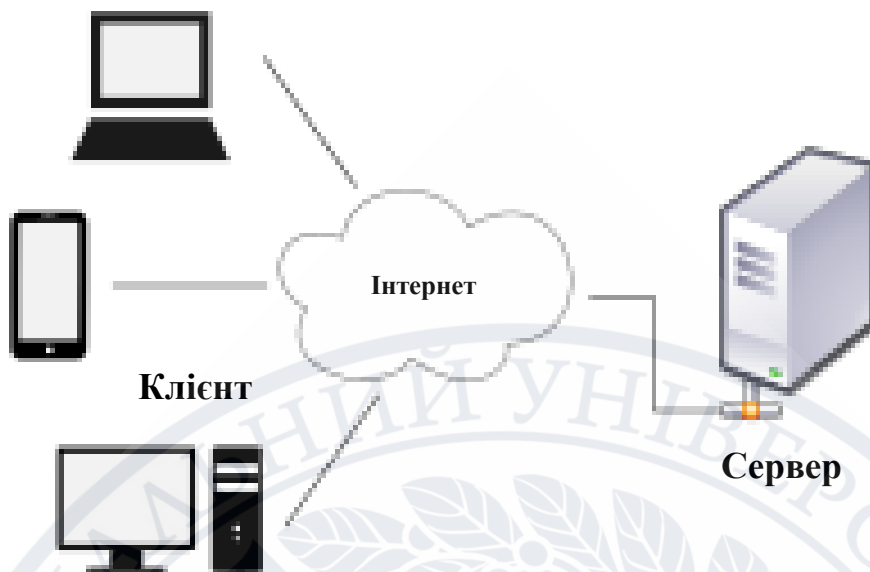


Рис.2.1. Архітектура мережа клієнт - сервер

Недоліком даної мережі є наступні пункти:

- 1) Може бути дорогим в обслуговуванні та комплектації, через необхідність в сервері, а також в мережевих пристроях, таких як маршрутизатори та комутатори.
- 2) Коли сервер вийде з ладу, це вплине на усю мережу.
- 3) Технічний персонал, який необхідний для ефективного обслуговування та забезпечення роботи мережі.

Інший тип мережевої архітектури є однорангова мережа. Скорочена назва мережі *P2P* [6], та принцип дії полягає у тому, що кожний вузол має еквівалентні обов'язки та можливості як показано на рис. 2.2. Однорангова мережа корисна тим, що зручно використовувати для невеликих середовищ, зазвичай до десяти комп'ютерів. Також можна виділи факт, що в даній мережі немає виділеного сервера, а кожному комп'ютеру назначаються спеціальні дозволи для співпраці з ресурсами. Але проблемою є, якщо комп'ютер з ресурсами не працює.

Перевагами однорангової мережі є такі аспекти:

- 1) Цей вид мережі менш затратно, так як не має окремого сервера.
- 2) Якщо один комп'ютер перестане працювати, інші комп'ютери не відмовлять в роботі.
- 3) Дану мережу легко настроїти і обслуговувати, адже кожен комп'ютер управляє сам собою.

Недоліками даної мережі є наступні пункти:

- 1) Однорангова мережа не має центральної системи, в наслідок чого вона не може виконати резервне копіювання даних через те, що дані в різних місцях відрізняються.
- 2) Проблема безпеки, оскільки прилад має контроль тільки сам над собою.

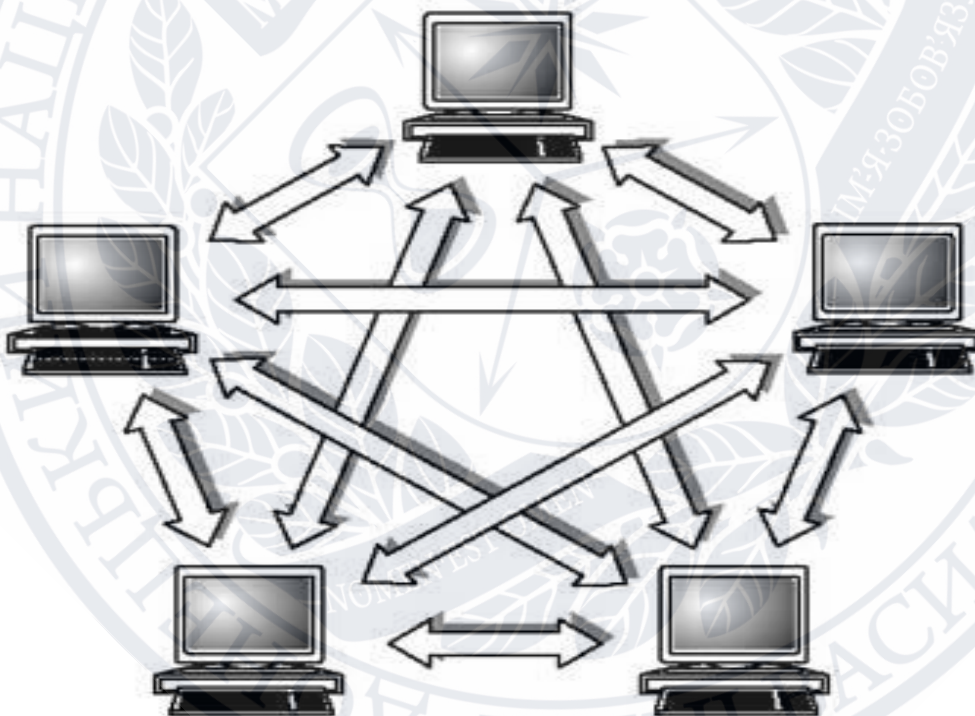


Рис 2.2. Архітектура однорангової мережі

### 1.3 Засоби захисту мереж та середовищ даних

Існують такі різновиди сучасних традиційних систем захисту.

- Системи виявлення зломисників (*IDS, Intruder Detection Systems*),



програми, що спостерігають за поведінкою систем та фіксують відхилення її внутрішніх та зовнішніх проявів від встановленої політики безпеки. Зазвичай, вони знаходяться на вузлах мережі, аналізуючи трафік та збираючи інформацію з підсистем [7].

- Пісочниці, системи, які мають можливість запускати підозрілі програми в ізольованому середовищі на відповідній віртуальній машині. Під час виконання програми в середовищі емуляції остання не зможе зашкодити системі користувача, а атака буде виявлена емулятором.

- Проактивні системи (*HIPS, Host-Based Intrusion Prevention System*), системи, оснащені відкритою таблицею правил. На підставі цієї таблиці система дозволяє або забороняє певні дії. Система орієнтована на діалог з користувачем і висуває високі вимоги до його компетентності.

У багатьох системах використовується технологія евристичного аналізу, яка дає змогу виявити ділянки підозрілого коду, що породжує шкідливу активність. Основні компанії, що працюють над захистом від кібератак є, *Check Point, CISCO, McAfee*.

Компанія *Checkpoint* працює в цій галузі з 1993 року, і розробляє комплексні продукти із захисту програмних систем як для хмарного середовища, так і для мобільних пристроїв. Одним з останніх є продукт *Checkpoint Software Blade Architecture* [8], що забезпечує комплексне конфігурування необхідних компонент кіберзахисту.

Компанія *CISCO* працює на ринку понад 20 років. Протягом цього часу відбулося значне підсилення арсеналу засобів захисту сучасними технологіями, величезною кількістю пристроїв та сенсорів. Компанія є знайомим розробником як апаратних засобів, так і програмного забезпечення, починаючи з роутерів та дата-центрів і закінчуючи поведінковими аналізаторами.

Компанія *McAfee* здебільшого спеціалізується на домашніх комп'ютерах, відома своїми антивірусами, захистом планшетів та смартфонів.

Незважаючи на використання розвинених технологій у програмній та апаратній сфері, сучасні системи захисту не є ідеальним захистом програмного середовища і мають низку недоліків. Насамперед, це досить велика кількість помилкових виявлень, особливо при підвищенні чутливості аналізатора.

Несвоєчасність виявлення атаки зумовлена значною тривалістю процесу виявлення. Це може статися, коли нападник використовує такі засоби маскування, як обфускація (маскування поведінки шляхом введення «мертвого коду»), поліморфізм (зловмисний файл, який змінює себе під час розповсюдження, запуску) та пакування файлів. Такі дії призводять до того, що під час перевірки файлу відома атака може бути взагалі не виявлена [9]. Ізольоване виконання програми також не позбавлене недоліків, воно потребує надто багато часу і ресурсів комп'ютера користувача, що негативно позначається на швидкодії під час виконання повсякденних операцій. До того ж, сучасні шкідливі програми здатні виявляти виконання в імітованому середовищі і припиняти своє виконання в ньому.

Протягом останніх п'яти років відбувся значний розвиток формальних методів, які є основою програм кіберзахисту, у зв'язку з розвитком таких інструментів, як розв'язувані задач та машини автоматичного доведення теорем [10].

Для розв'язання проблеми безпеки, та захищеності або проблеми кіберзахисту, існує таких два аспекти.

1. Виявлення вразливостей у наявній програмі, поданій у вигляді бінарного або вихідного коду програми.
2. Виявлення атаки в режимі функціонування програмних ресурсів у певному комп'ютерному середовищі, такому як окремий комп'ютер, комп'ютерна мережа або хмарне середовище.

У 2016 році змагались 40 команд [11] за виявлення максимальної кількості вразливостей, впродовж обмеженого проміжку часу. Основними переможцями стали такі команди:

- Система *Mayhem* [12] команди з Пітсбургу, що виграла конкурс.

системі було застосовано символічне виконання програм із використанням розв'язної системи *Microsoft Z3* та символну модель пам'яті.

- Система *Xandra* [13] команди з Нью-Йорку, у якій було використано власний символний пошук вразливостей.
- Система *Mechanicle Phish* [14], у якій було використано метод розпорошення ("*Fuzzing*") у поєднанні з символним виконанням.

Метод розпорошення є основним засобом знаходження невідомих уразливостей, він полягає у тому, що під час реалізації різних сценаріїв виконання програми, генеруються критичні величини змінних у програмі, які у свою чергу можуть призвести до вразливих дій. Однією з найвідоміших *Fuzzing*-машин є *ATL*, створена компанією *Google*.

## **Розділ 2. МЕТОДИ ЗАХИЩЕНОСТІ МЕРЕЖ І СЕРЕДОВИЩ ТА ЇХ КЛАСИФІКАЦІЯ**

### **2.1 Основні методи захищеності мереж та їх властивості**

Існує багато методів та засобів за допомогою яких відбувається захист мереж та середовищ, серед них можна виділити основні напрями, такі як

- Аналіз послідовності системних викликів
- Системи виявлення атак за допомогою символних обчислень
- Виявлення вразливостей у повторно використаному коді
- Системи виявлення атак на основі конкретних та символних обчислень

Аналіз послідовності системних викликів. Системний виклик, це звернення програми до ядра операційної системи задля виконання операції. Наприклад такі як, функції, що зчитують або записують файл, а також робота з мережею. Для спілкування з мережею або збереження налаштувань користувача, використовують системні виклики [15]. Ця система працює наступним чином, вона перехоплює та формує послідовності системних викликів для конкретного процесу. Існують відомі, відкриті масиви з



послідовностями викликів, як в свою чергу відповідають конкретним властивостям. З використанням послідовностей, формуються алгоритми, які визначають, чи є ці послідовності потенційними загрозами для атак.

Задля побудови алгоритмів класифікації [16] вразливостей, використовуються формальні методи. Одним із таких формальних методів, є той, що ґрунтується на алгоритмі з використанням вікон, які рухаються по послідовності системних викликів. Цей алгоритм є відомим і розповсюдженим, адже фіксований розмір вікна, дає змогу регулювати швидкість та використовувати статичні методи. Перевага аналізу послідовності системних викликів, є універсальність, тому що виклики являються уніфікованим інтерфейсом роботи між програмою та операційною системою. Також можна знаходити атаки нульового дня, це такі атаки які ще не були виявлені [17]. А недоліками даного підходу є те, що через малу кількість даних для навчання. Такі системи виявлення атак мають великий відсоток помилок першого роду і можуть впливати на швидкість програми.

Система виявлення атак за допомогою символьних обчислень. Символьне обчислення складається з інтерпретації коду, саме коли під час виконання програми використовуються символи, що є довільними значеннями замість конкретного вводу, такі як рядки або цифри. Символьне значення генерує всі можливі шляхи виконання програми через те, що обчислення здійснюється у конкретному виконанні, але за винятком, того, що значення, оброблені за програмою, можуть бути символьним виразом над вихідними символами. Множина вхідних даних, яка формує шлях виконання, виводиться для кожного шляху. Саме ці дані є корисними для виконання діагностики на низькорівневих збірках та високорівневих властивостях програм.

Коли немає доступу до вихідного коду програми, тоді предметом аналізу стає програма у дизасембльованому виді. Дизасемблювання, це процес, під час якого відбувається трансляція програми з машинних кодів у мову асемблер. За рахунок чого, символьне значення вже виконується на основі асемблерного представлення програми.

Методом для використання результатів символьного обчислення є ідентифікування чутливих місць, де саме знаходяться ненадійні дані [18]. Ненадійні дані це, дані , що надходять програмі на вхід і не проходять потрібного оброблення. Чутливі місця це, місця програми, які можуть бути за певних обставин, бути джерелом некоректної поведінки програми.

Отож, підхід виявлення атак за допомогою символьних обчислень є надійний та досить ефективний для статичного аналізу даних. Він може вдало знаходити вразливості, будувати шляхи ненадійних даних та передбачає можливо потенційну загрозу автоматичного збирання великих масивів.

Виявлення вразливостей у повторно використаному коді. Для розробки програмного забезпечення, програмісти можуть копіювати код для повторного використання, а також копіюючи сам код разом з ним йде вразливості програмного забезпечення, що в свою чергу впливають на безпечність системи. Зловмисник може використати ці вразливості і в подальшому усі місця з подібним вразливим кодом та використати у своїх цілях.

Для реалізації даної системи різні по складності методи. Найпростіші методи такі як використання лексичних аналізаторів, що будують послідовність токенів для коду, який містить слабкі місця та може бути в подальшому скопійований. Наступним кроком будується послідовність тих самих токенів для програми, що перевірятиметься на вразливості, а у послідовності система намагається знайти підпослідовність, яка є схожою до послідовності вразливостей. При вводі параметрів подібностей послідовностей, виражається чи змінював програміст код та адаптував його. Методом заміни параметра можна, використовувати попередні дані про вразливості, знаходити їх у зміненому коді, але при цьому точність методу стає гіршою, адже з'являється велика кількість помилок першого роду [19].

Програми, що є більш ефективні, враховують семантичні особливості вразливостей на відміну від простих. Прикладом є система, що будує граф програми, у якому враховується залежність між даними та потоком керування програми [20]. Саме ці графи генеруються для коду, який має слабкі місця, а

також для програми, що перевіряється. У такому випадку пошук вразливостей починає виконувати пошук ізоморфного значення.

Даний метод має також ряд недоліків, зумовлених фундаментальними обмеженнями швидкості пошуку підграфу, а також фрагменти коду окремої мови програмування аналізуються швидше, тобто для кожної мови необхідно розробити свій аналізатор.

Система виявлення атак на основі конкретних та символічних значень. Поява цих систем зумовлена на символічних обчисленнях та називається *concolic* – обчисленнями. Обчислення об'єднують символічні обчислення та конкретні виконання. Використовуючи їх, можна генерувати різноманітні шляхи виконання програм, а також дані, що надходять до відповідних шляхів. Дані використовуються для тестування та знаходження слабких місць у програмі. У *concolic* – обчислень використовуються конкретні дані, а їх символічне представлення слугує для обходу проблеми символічного виконання [21].

У першу чергу виконується налаштування середовища для виконання даної програми. Одним із способів запуску процесу є по черговість виконання задач. На кожному кроку можна зчитувати значення всіх регістрів та значення у пам'яті. Далі існує кілька технік, однією з яких, побудова шляху даних, що будуть подаватись на вхід [22]. А далі будуються тестові дані, що зібрані за допомогою обмежень, які показують слабкі місця у системі за допомогою символічного обчислення.

Задачею цього методу являється усунення обмежень, які є у системі, що у свою чергу гуртуються на символічних обчисленнях. Наприклад, експоненційне зростання кількості шляхів відносно до кількості інструкцій, являє собою обмеження. Також виконується обробка випадків, коли в програмі є цикли та рекурсивні значення. Але досвід показав, що для кожної програми необхідно розробити своє програмне забезпечення для коректної роботи. Також не мало важливим є те, що розробка початкового набору тестових даних відбувається не автоматизовано, а вручну. Це може свідчити,



що такі системи розробляються в більшій мірі як доповнення до наявних програм.

## 2.2 Формальні методи, які застосовуються для визначення атак

Задля забезпечення мережевих систем, що використовують *Web*-технології, створений комплекс, починаючи з рівня мережевого транспорту до рівня прикладного протоколу системи і кінцевих додатків[23].

На рівні мережевого транспорту, завданням є забезпечення мереж, розв'язують за допомогою маршрутизаторів та брандмауерів. Маршрутизатори виконують роль організування мережевого трафіку, дотримуючись окремих правил, які задаються за допомогою налаштувань. Також маршрутизатор може пропускати тільки певну частину мережевого трафіку, що оберігає мережу, яка знаходиться за ним, саме від небажаних та підозрілих пакетів. Маршрутизатор працює на «мережевому» (третьому) рівні мережевої моделі *OSI*, на відміну від комутаторів і концентраторів, які працюють відповідно на другому і першому рівнях моделі *OSI*.

Брандмауер, це захисний екран між глобальним інтернетом і локальною комп'ютерною мережею організації. Він виконує функцію перевірки і фільтрації даних, що надходять з інтернету. Залежно від налаштувань, брандмауер може пропустити їх або заблокувати. Програма, що пропускає тільки частину мережевого трафіку, дотримуючись окремих правил. Існують не завершені питання, пов'язані з перехопленням ключів безпеки[16], паролів або веб-сайтів. Саме для подібних випадків розробляються формальні методи, що мають за основу верифікацію коректності мережевих протоколів.

Протоколи мережевої безпеки зазвичай базуються на криптографічних примітивах та аналізу, що є одним із найскладніших завдань. Такі поля, як криптосистеми, підпис, схеми, захищені хеш-функції, механізми передачі та захищені багатопартійні методи оцінки функцій, використовуються, коли вони необхідні для підвищення безпеки протоколів. Тому мережеві протоколи

часто піддаються неінтуїтивним атакам. Протокол безпеки повинен бути в змозі досягти своїх цілей перед спробою взлому або захопту зловмисниками, як зазначено у роботі [24]. Також у ній описано формальну модель протоколу безпеки та показано можливість визначення для перехоплення ключа безпеки за допомогою, саме формальних методів. Для цього використовується теорія, що включає опис мови, яка складається з множин символів (теорем), констант, символів ролей та функціональних символів, а також множини формул, що задають еквівалентність термів і множин правил виведення. Протоколи задаються як визначення поведінок, кожна з яких є описом транзиційної системи, яка в свою чергу описує процеси створення, відправлення, отримання та обробки повідомлень. Для точного опису транзиційної системи можна використати діаграму *MSC* (*Message Sequence Chart*), а результати застосування правил представити у вигляді трас *MSC*.

Робота [25] дає змогу зрозуміти які вразливості та проблеми можуть виникнути, а саме специфікації протоколу можуть бути недостатньо точними, щоб гарантувати безпеку. Реальні реалізації можуть не відповідати формальним специфікаціям, використовуваним в доказах. Формальні докази можуть привести до самозаспокоєння, перешкоджаючи майбутнім аудиторам та інспекціям. Також описано набір інструментів, що включають мови специфікацій та інструменти для генерації коду, що дають змогу формально описати і реалізувати необхідний протокол, що може та буде відповідати заданій специфікації[26].

У роботі [27] описується техніка офіційної перевірки безпеки маскованих реалізацій проти атак бічних каналів на основі перетворень елементарних схем. Представлено два взаємодоповнюючі підходи: загальний підхід для офіційної перевірки будь-якої схеми, але для невеликої атаки лише замовлення та спеціалізований підхід для перевірки певних схем, але в будь-якому порядку. Також показується, як автоматично створювати докази безпеки для простих схем. Можна відокремити метод визначення атак, пов'язаних зі змінами топології мереж, що ґрунтується на трансформації топологічних схем.

У цій роботі використовуються замасковані контр заходи, такі як замасковані змінні проти атак, що використовують у свою чергу побічний канал.

Методи, пов'язані з моделюванням мережеских атак з також формальним аналізом моделі, описано чіткіше у роботі [28], а саме метод аналізу мінімізації, який дозволяє аналітикам вирішити, який мінімальний набір заходів безпеки гарантуватиме безпеку системи, а також метод аналізу надійності, який дозволяє аналітикам знайти простий компроміс між витратами і вигодою в залежності від ймовірності атак. Атаки представлено графами атак та використано так звану техніку *model checking* для подальшого аналізу властивостей цих графів. Граф атак - це коротке представлення всіх шляхів через систему, які закінчуються в стані, коли зловмисник успішно досяг своєї мети. А також, що є не мало важливим, автори застосовують імовірнісний аналіз за допомогою розмітки графів атак величинами ймовірностей і техніки марковських процесів.

Існують, так звані готові інструменти, такі як *Pro Verif* та *Tamarin Prover*, що призначені для формальної верифікації протоколів безпеки, які використовують, символічні методи верифікації. Існує інструмент *Legacy Crypto*, який застосовується для аналізу вразливостей протоколів[29], що використовують криптографію. Крипто-політика - це системний компонент, який налаштовує основні криптографічні підсистеми, що охоплюють протоколи *TLS*, *IPSec*, *SSH*, *DNSSec* і *Kerberos*. Він надає певний набір політик, які може вибрати адміністратор.

Так як більшість мережеских систем на цей час ґрунтується саме на веб-технологіях, питання безпеки таких систем пов'язані також з вразливостями прикладних протоколів. У роботі [30] описано та докладено про інструмент для моделювання та аналізу веб-систем, який надає змогу визначити вразливості системи для атак. Безпека браузерів його протоколів, ключові компоненти захищеного Інтернету додатків, що в основному ігнорується, на відміну від аналога, що підтверджує правильність протоколу безпеки, для яких існує кілька інструментів, що передбачають великий ступінь автоматизації та



з використанням будь-якого різного підходу, включаючи формальні методи. Хоча такі інструменти є гарною відправною точкою для аналізу безпеки браузерних протоколів, їх недостатньо, оскільки для порівняння протоколів безпеки, браузерних протоколів передбачають більш складну поведінку і більш складну структуру повідомлення. Далі, інструменти для загального аналізу програми, вимагає специфічного обладнання та методу доказу, які не підходять для перевірки браузерних протоколів, оскільки вони часто вимагають повного дослідження простору для досягнення вихідного рішення. Аналіз безпеки браузерних протоколів є складним, схильним до помилок і важким для автоматизації[31]. Він повинен враховувати проблеми, які виходять за рамки інструментів для протоколу безпеки аналізу. Також, це передбачає враховуючи вплив політики браузера на гарантії інформаційної безпеки, наслідки браузера з використанням фреймів, запущених сценаріїв та введення файлів *cookie*, складні взаємодії, які можуть виникнути у користувача, пов'язаного з браузером, мережа та сервери, на яких запущено протокол, що аналізується, а також поведінка є складний протокол, який часто залишає користувачу неповне знання того, яка інформація виконується, а яка обмінюється.

Існує два види атак, такі як :

- Атака *Cross Site Scripting (XSS)*, один з різновидів атак на веб-системи, яка виконує впровадження шкідливого коду на певну сторінку сайту і взаємодія цього коду з віддаленим сервером зловмисників при відкритті сторінки користувачем. Ця дія дає змогу виконувати транзакції від імені клієнта та викрадати особисті дані клієнта .

- Атака *Cross Site Request Forgery (CSRF)*, підробка міжсайтових запитів, під час якої користувач отримує вільний доступ до сторінки, яка в свою чергу має контроль зловмисником, що дає змогу останньому виконувати транзакції від імені користувача.

Відмінність цих двох видів атак полягає у тому, що *CSRF* сервер повинен знаходитись під пильним контролем зловмисника, а у *XSS* сервер та його

відповідні служби і цілому залишаються безпечними.

Протоколи на основі браузерів діляться на незліченну схожість з іншими протоколами і додатками, однак, у випадку браузерів на основі протоколів, учасники в них не мають повне розуміння того, що таке правильний потік інформації або знати про все дані, що передаються від одного учасника до іншого. Ці відмінності полягають в тому, що потрібна модель, спеціально розроблена для них, а інструмент, що представлений не лише призведе до кращого розуміння браузерного протоколу властивостей, що у свою чергу дозволяють аналізувати потенційну безпеку вразливостей та недоліки зазначених протоколів може мати, але також допомогти у розвитку нової політики та можливості для браузерів та серверів, оскільки для перевірки їхньої безпеки потрібно невелика модифікація інструменту.

Інші види атак у розподілених системах, що використовують веб-технології, пов'язані з:

- Інтерпретацією адрес, це коли браузер посилає на сервер некоректно сформований запис – *URL*, а у свою чергу невірно налаштований сервер може надати зловмисникові повний доступ до інформації, що зберігається або виконати будь-які системні виклики.
- Недостатньою перевіркою наданих даних, такі як клієнт-зловмисник може відправляти інформацію до сервера, що порушує стандартне функціонування останнього. На цьому етапі можуть використовуватись недопустимі символи, невідповідності типів, порушення меж масивів, значення за межами допустимого діапазону.
- Прямою інтерпретацією *SQL*-запитів, а саме часті параметри *SQL*-запитів передаються безпосередньо в даних веб-форм або в рядках *URL*, і в цих випадках звичайна підміна параметрів може надати зловмисникові несанкціонований доступ до даних.
- Проблемами з переповненням буфера, є недостатнім ступенем захисту коду у веб застосунках, що призводить у разі переповнення вхідних буферів до порушення функціонування сервера або навіть надання зловмисникові

можливості виконати на сервері системні команди. Частина *DDoS*-атак також використовують ці вразливості.

- Декомпіляція *Java* коду. Оскільки байт-код може бути найефективніше компільований, злочинець може отримати з нього інформацію, що робить неможливим несанкціонований доступ до сервера, такий як паролі.

- Перехоплення ключів безпеки. Це означає, що ключі *SSL* можуть бути відкриті і підроблені або замінені зловмисником, що дає змогу останньому виконувати транзакції від імені клієнта.

- Імперсоналізація або перехоплення сесій. Мається на увазі те, що протокол *HTTP* не передбачає зображення станів під час взаємодії з користувачем, а для цього створені ідентифікатори сесій, що передаються між користувачем і сервером як приховані поля. Перехоплення такої інформації призводить до отримання інформації про користувача.

### **2.3 Систематизація та побудова формальних методів**

Швидкий розвиток Всесвітньої павутини в останні роки різко позначився, збільшивши обмін інформації між клієнтами та компаніями, а також активізувавши операції електронної комерції. Традиційно довілля де відбуваються електронні транзакції, складається з веб-сервера, який пропонує "послуги" для клієнтів, які використовують *Web*-браузер для відбору інформації чи продуктів, що вони хочуть отримати. У наш час цей погляд змінюється, компанії пропонують зробити користування послугами - автоматично, тобто вони хочуть «жити» у світі, де взаємодія між різними програмами працює на окремих платформах. Технологія, яка з'явилася нещодавно і пропонує подібні операції є *Web*-служби [32]. *Web*-служби реалізують нову архітектуру, орієнтовану на сервіси (*SOA*), яка базується на вільно пов'язаних між собою послугах. У веб-сервісах навколишнього середовища можна знайти такі компоненти [33]:



- Службові брокери: дозволяють клієнтам отримувати доступ до сервісного інтерфейсу та інформацію про впровадження.

- Клієнти послуг: шукають послугу в реєстрі брокерів, а потім під'єднуються до постачальника послуг, щоб використовувати його.

Однією з найважливіших проблем у розробці веб-служб є те, що кожний функціональний блок повинен бути незалежним від платформи або мови програмування, і доступний для всіх. Таким чином, цей самий блок повинен бути описаний за допомогою стандартів. Найважливішими стандартами, що стосуються веб-послуг, є:

- *XML (eXtensible Markup Language)* [34] - це мова розмітки, яка лежить в основі більшості специфікацій, пов'язаних з веб-послугами. *XML* насправді являється унікальною мовою, адже вона описує іншу мову, яка дозволяє розробити власні мови розмітки для необмежено різних типів документів.

- *SOAP (Simple Object Access Protocol)* [35] - це протокол обміну повідомленнями на основі *XML*, який використовується для кодування інформації у запиті та відповіді веб-служб повідомлення перед надсиланням їх через мережу. Повідомлення *SOAP* не залежать від будь-якої операційної системи або протоколу і може транспортуватися за допомогою різноманітності Інтернет-протоколів, включаючи *SMTP*, *MIME* та *HTTP*.

- *WSDL (Мова опису Web-служб)* [36] - це формат *XML* для опису мережевих служб як набору кінцевих точок, що діють на повідомлення, що в свою чергу містять інформацію, орієнтовану на документ або дію.

- *UDDI (універсальний опис, виявлення та інтеграція)* - це клієнтський *API* та реалізація сервера на основі *SOAP*. Він зберігає та отримує інформацію про постачальників послуг та веб-сервісів.

В останні кілька років багато дослідників безпеки пропонували наділити веб-платформу більш суворими, аналітичними засадами. Їх метою є розробка моделей, які дозволяють чітко міркувати з питань веб-безпеки та розробляти ефективні інструменти, задля того щоб зробити *Web* безпечнішим, спростивши роботу веб-розробників. Однак, враховуючи складність мережі,

дослідницькі зусилля в цій області досить узагальнені по багатьох різних темах та проблемах, а через це на сьогоднішній день нелегко зрозуміти суть формальних методів веб-безпеки. Для більш чіткої роботи у цій галузі, виділяють три основні підтеми для вирішення проблеми з безпеки .

Визначення найбільш важливих викликів, з якими повинні стикатися розробники, зацікавлені в дослідженні застосування офіційних методів до веб-безпеки.

Збирання, класифікація та перегляд існуючих пропозицій в області офіційних методів веб-безпеки, що охоплюють багато різних тем: безпека *JavaScript*, безпека браузера, безпека веб-додатків та аналіз веб-протоколів.

Обговорення рекомендацій для розробників, які працюють у галузі формальних методів веб-безпеки.

Існує представлення формальних моделей атак, що пов'язані зі спробами зламу зашифрованих даних користувача [37]. Злам відбувається за допомогою наступних веб-додатках, менеджер паролів, хмарне середовище та системи електронного голосування. Зашифрована програма зберігання, яка використовує *JavaScript* та сеанси на основі файлів *cookie* піддається дії і повинна захищати від ряду веб-атак.

*Доставка коду.* У типових розгортаннях веб-сайтів виконується код *JavaScript*, клієнтське шифрування самостійно завантажується з Інтернету. Якщо зловмисник контролює сервер, на якому розміщений *JavaScript*, він може пошкодити код програми в порядку витоку ключів.

*XSS.* У найпростішій формі зловмисник може скористатися користувачем при вході в програму для введення *JavaScript*, який розміщується на веб-сайті *HTML* і запускається разом із надійним кодом *JavaScript*. Це може дати зловмисникові повний контроль над веб-сторінкою у браузері та всіма криптографічними матеріалами які доступні для цієї сторінки.

*Викрадення сесії.* Якщо зловмисник досягне *cookie*, він може виконувати той самий набір операцій із сервером, що і користувач. Рішенням такої

проблеми, може бути встановлення файлу *cookie* в безпечному режимі, забороняючи браузеру надсилати його через незашифроване з'єднання.

*CSRF*. Наприклад, коли дія може бути ініційована шляхом доступу до певної *URL*-адреси, змінивши цю саму адресу електронної пошти поточного користувача.

*Фішинг*. Можуть бути представлені функції, що стосуються третіх осіб, так звані нові вектори атак. Наприклад, у протоколі автоматичного заповнення форми, ненадійний веб-сайт може спробувати подати розширення фальшивою *URL*-адресою замість рідної *URL*-адреси для входу. Це сприяє фішингу, де веб-сайт може обдурити користувачів, що вони відвідують сторінку правдиву коли насправді вони перебувають на веб-сайті фальшивому.

## **2.4 Інсерційний підхід до методу захищеності та його особливості**

У 2017 році в Інституті кібернетики ім. В.М. Глушкова була розпочата робота з активного використання алгебраїчного підходу, в процесі якого виявлення слабких місць у бінарному коді. За основою взятою із теорії інсерційного моделювання, розробленої О.А. Летичевским та Д. Гільбертом [38], було здійснено формалізацію семантики асемблера мови інструкцій x86. Для цього в свою чергу було використано математичний апарат алгебри поведінок, що є частиною інсерційного моделювання.

Основний принцип інсерційного підходу [39] є взаємодія агентів у даному середовищі, при чому кожен агент володіє інформацією тільки про середовище, але може бути сам середовищем для агентів іншого рівня абстракції. Поведінку кожного агента можна охарактеризувати рівняннями алгебри поведінок. Атомарні дії агентів є об'єктами алгебри поведінок та можна визначити як перед– та після умовою на атрибутах агентного середовища. Можна відокремити два типи машин: машини реального часу або інтерактивні та аналітичні машини [40]. Перші існують у реальному або



віртуальному середовищі, взаємодіючи у реальному або віртуальному часі. Аналітичні машини призначені для аналізу моделей, дослідження його властивості, вирішення проблем тощо. Драйвери для двох типів машин відповідно поділяються також на інтерактивні та аналітичні. Інтерактивний драйвер після нормалізації стану навколишнього середовища повинен вибрати одну альтернативу та виконати дію, зазначену як префікс цієї альтернативи. Машина з інтерактивним драйвером діє як агент, що вставляється у зовнішнє середовище із вставкою функції, що визначає закони функціонування цього середовища. Аналітична машина для вставки[41], на відміну від інтерактивної може розглядати різні варіанти прийняття рішення про виконані дії, повернення до точок вибору (як при логічному програмуванні), а також розгляд різних шляхів у дереві поведінки моделі. Модель системи може включати модель зовнішнього середовища цієї системи та продуктивність водія залежить від цілей вставки машини [42]. У загальному випадку аналітична машина вирішує задачі шляхом пошуку станів, що мають відповідні властивості (цільові стани) або стани в які зазначені властивості безпеки можуть порушуватись. Зовнішнє середовище для вставки машина може бути представлене як користувачем, який взаємодіє з машиною, задає проблеми та контролює активність самої машини. Для цього використовується аналітична машина, збагачена логікою та дедуктивними інструментами генерування слідів символічних моделей систем. Стан символічної моделі представлений за допомогою властивостей значень атрибутів, а не конкретних цінностей. Драйвер моделі високого рівня забезпечує інтерфейс між системою та зовнішнім середовищем, включаючи користувачів системи [43].

Алгебра поведінок є так званою двосортною алгеброю, що визначає відношення та операції над поведінками та діями. Кожна поведінка являється композицією дій та поведінок. Операція префіксинга  $a.B$  визначає, що дія  $a$  передуює поведінці  $B$ . Операція недетермінованого вибору поведінок визначає альтернативу сценарію поведінки. Алгебра як відомо має три термінальних константи: успішне закінчення, тупиковий стан, а також невідому поведінку.

На множині поведінок є визначене відношення апроксимації, яке в свою чергу встановлює частковий порядок на множині поведінок з мінімальним елементом. Алгебра поведінок також може визначати композицію поведінок, а саме послідовну, та паралельну[44]. Для узагальнення, поведінка агента може бути представлена як вираз над поведінкою і рекурсивною дією. Вище зазначені дії, визначаються над деякими середовищами атрибутів, у яких всі агенти взаємодіють один з одним. Кожен агент визначається певним набором атрибутів [45]. Агент змінює свій стан за деяких важливих умов, сформованих значеннями атрибутів. Дії кожного агента визначаються конкретною парою –  $a = \langle P, S \rangle$ , де  $P$  – передумова дії, що має вигляд як формула в деякій базовій логічній мові,  $S$  – постумова. Базова логічна мова представлена у вигляді набору формул логіки першого порядку над поліноміальною арифметикою та деякі спеціалізовані теорії. Прикладом є переліковані типи, побітові операції або байтові вектори. Обумовлюючи, семантика дії [46] означає, що агент міг би змінювати свій стан, за умови, якщо передумова є істинною, а стан зміниться відповідно до постумови, що в свою чергу також є формулою логіки першого порядку [47].

## Розділ 3. ФУНКЦІОНАЛЬНА ЧАСТИНА МЕТОДУ ЗАХИЩЕНОСТІ ТА БЕЗПЕКИ ПРОГРАМНИХ СИСТЕМ

### 3.1 Класифікація та визначення *DDoS* атак

Додатковим типом *DoS* атаки можна рахувати розподілену відмова в обслуговуванні, а саме *DDoS* атака. Ця атака зазвичай відбувається, коли декілька систем організовують синхронізовану *Dos* – атаку на одну ціль. Значна відмінність полягає у тому, що одну ціль атакують одразу декілька, а не одну ціль атакує один. Через розподіл хостів, що визначає *DDoS* атак, зловмисник має значну перевагу :

- Можна використовувати велику кількість техніки для проведення серйозної атаки.
- Місце атаки складно буде визначити, адже існує випадковий розподіл атакуючих систем.
- Вимкнення машин майже не реально через їх велику кількість
- Ідентифікувати атакуючу сторону складно, так як вони замасковані в основному за скомпрометованими системами.

*DDoS*-атаки зазвичай здійснюються в мережах комп'ютерів, підключених до Інтернету. Ці самі мережі складаються з комп'ютерів і інших пристроїв, таких як наприклад пристрій *IoT*, заражених шкідливим програмним забезпеченням, що дозволяє зловмисникові дистанційно керувати ними. Ці окремі пристрої називаються *bot*. Складена група із так званих ботів, називається “ботнет”.

Вже після створення “ботнету” зловмисник може направити атаку, відправивши окремі інструкції кожному *bot*. На момент коли сервер або мережа на яку нападає зловмисник стає метою “ботнету”, кожен *bot* відправляє запити на *IP*-адресу цілі, потенційно викликаючи перевантаження сервера або мережі, що вже може призвести до відмови в обслуговуванні нормального трафіку.

Для визначення того, що здійснюється *DDoS* атака є основні правила. Першим і головним показником є миттєве зменшення швидкості роботи сайту або його



недоступність. Такий не продуктивний трафік може викликати ряд проблем, тому необхідні інструменти для своєчасного вирішення питань. Існують деякі методи для аналізу явних ознак *DDoS* атак :

- Підозрілі обсяги трафіку, які виходять з однієї *IP*-адреси або діапазону *IP*-адрес.
- Потік трафіку від користувачів, які поділяють один спільний профіль, такий як геолокацію, тип пристрою, або версія веб-браузера.
- Незрозумілий сплеск запитів до окремої сторінки або кінцевої точки
- Нестандартні моделі трафіку, такі як сплески в непарні години дня або занадто часткові, прикладом є сплески які відбуваються кожні 5 хвилин.

Існує велика кількість та різні типи *DDoS*-атак , які в свою чергу націлені на різні компоненти мережевого підключення. Мережеве з'єднання в Інтернеті складається з безлічі різних компонентів або з семи «шарів», які показані у табл.3.1. З цих шарів складається модель *OSI*, що являє собою основу, ку використовують для опису мережевих під'єднань на будь-якому із семи рівнів.

Таблиця 3.1

Модель *OSI*

Сьомий рівень – Прикладний	Початок створення пакетів даних
Шостий рівень – Представницький	Трансляція даних від джерела до отримувача
П'ятий рівень – Сеансовий	Управління процесом установки та завершення з'єднання та синхронізація сеансів

Четвертий рівень - Транспортний	Забезпечення безпомилкової передачі інформації
Третій рівень – Мережевий	Маршрутизація та передача інформації між різними мережами
Другий рівень - Канальний	Встановлення та супровід передачі повідомлень на фізичному рівні
Перший рівень – Фізичний	Передача двійкових даних

На рис. 3.2 зображена класифікація та відмінність *DoS* та *DDoS* атак. Спершу атаки поділяються на протоколи *IPv4* та *IPv6*. *IPv4* протокол мережевої взаємодії, що використовується для передачі зашифрованих даних. *IPv6* нова розроблена мережа протоколювання мережевої взаємодії. Також діляться на атаки рівнів чотири на сім. Атака рівня чотири – транспортна, передбачає безпечну передачу інформації між вузлами. Завдана шкода від *DDoS* атаки на четвертому рівні це досягання меж по ширині самого каналу або за кількістю допустимих підключень, порушення роботи мережевого устаткування. Атака сьомого рівня – прикладна, передбачає створення пакетів даних, доступ та приєднання до даних. Завдана шкода від *DDoS* атаки на сьомому рівні це брак ресурсів, надмірне споживання системних ресурсів службами на сервері, що атакується. Також атаки на основі протоколу *IPv6* поділяються на атаки за основою протоколу *Discovery*.

Атаки рівня чотири - включають наступні аспекти :

- Атака *TCP*
- Атака *UDP*

- Атака *ICMP* (v6)
- Атака *Smurf*

Атаки сьомого рівня - включають наступні аспекти :

- Незавершені *HTTP*-запити за допомогою методу *GET*
- Неповні *http*-запити методом *post*
- *HTTP*-запити за допомогою методу *HEAD*

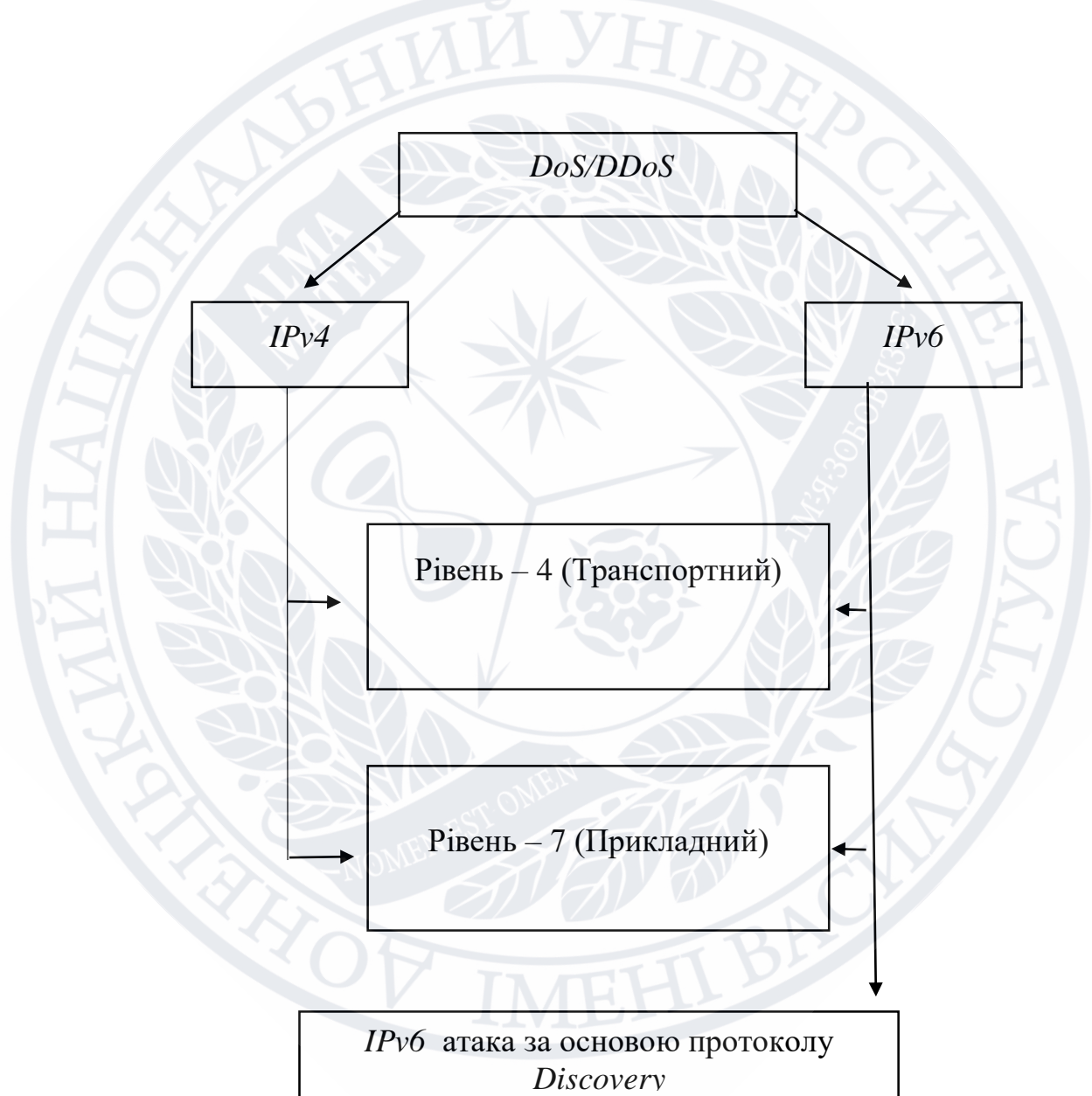


Рис. 3.2. Основна класифікація *DoS/DDoS* атак



Незважаючи на те, що *DDoS* атаки націлені лише на перенавантаження мережі можна виділити серед них основні три напрямки по яким цілять *DDoS* атаки.

1) Атаки на рівні додатків.

Це атака на сьомому рівні – транспортному. Метою та цілю такої атаки є вичерпати ресурси, які стали б метою в відмові на обслуговування. Атаки націлені на рівень, на якому веб-сторінки, що створюються на сервері і доставляються у відповідь на *HTTP*-запити. Один *HTTP*-запит є швидким для обчислення у виконанні на стороні клієнта, але для цільового сервера відповідь на нього може бути значно довшим, адже сервер часто завантажує кілька файлів і виконує запити до бази даних, щоб створити веб-сторінку.

Від атак рівня сьомого – транспортного, складно бути захищеною, так як складно відрізнити зловмисний трафік від легітимного.

Прикладом атаки на сьомому рівні є атака під назвою *HTTP flood*.

Дана атака має схожість з процесом натискання кнопки для оновлення *web* – сторінки, тільки це відбувається з декількох комп'ютерів та одночасно. Велика кількість *HTTP* – запитів заповнює сервер виконання, що призводить у свою чергу до відмови в обслуговуванні рис 3.3.

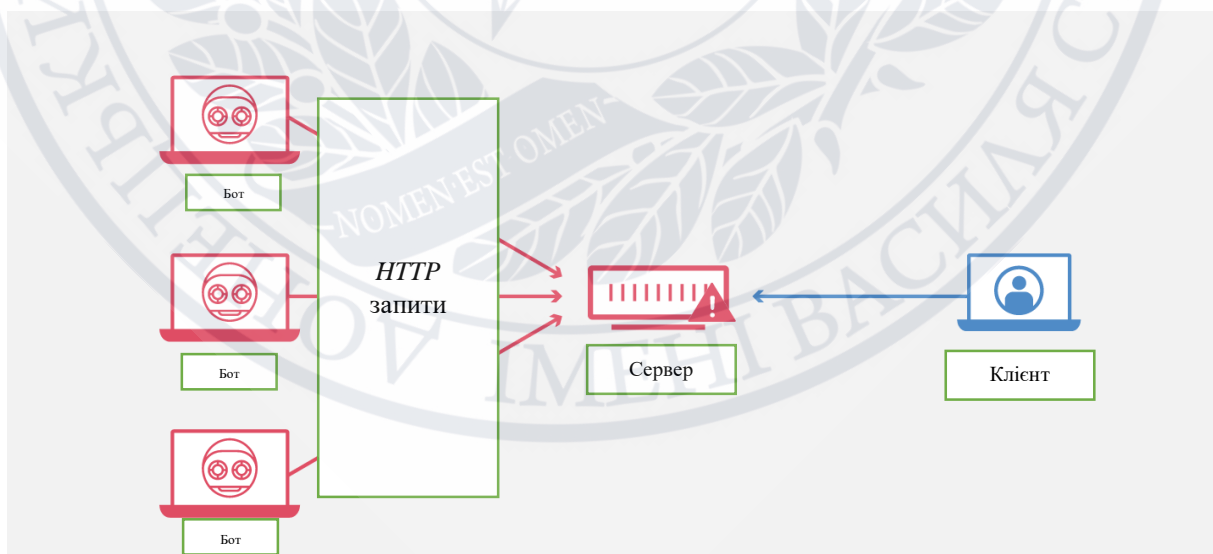


Рис. 3.3. Атака *HTTP flood*

2) Протокол атаки.

Суть даної атаки полягає у тому, що виконується надмірне споживання ресурсів серверу або мережевого оснащення, такого як наприклад брєндмауєри. Виконання цих дій провокує виснаження стану.

Ці атаки призначені на рівні три та чотири – мережевий, транспортний відповідно, використовують слабкі місця протоколів.

Прикладом атаки рівнів три та чотири є атака *SYN flood*.

*SYN* потік полягає у тому, що запит йде за пакетом даних та очікує певного підтвердження, перш ніж доставити його. Наступним кроком отримується велика кількість запитів пакетів без їх підтвердження, до того часу, поки вони не перестануть відправляти паки даних, що не будуть перенавантажені і запити не будуть без відповідей.

Ця атака використовує так зване “рукостискання” *TCP* - послідовність обміну даними, за допомогою якої два комп'ютера ініціюють мережеве з'єднання - шляхом відправки цільового об'єкту, великої кількості *SYN*-пакетів *TCP* з підробленими *IP*-адресами джерела. Йде відповідь на кожен присланий запит до підключення, а потім йде очікування останнього кроку рукостискання, якого ніколи не відбувається, виснажуючи ресурси цільового об'єкта в процесі рис. 3.4.

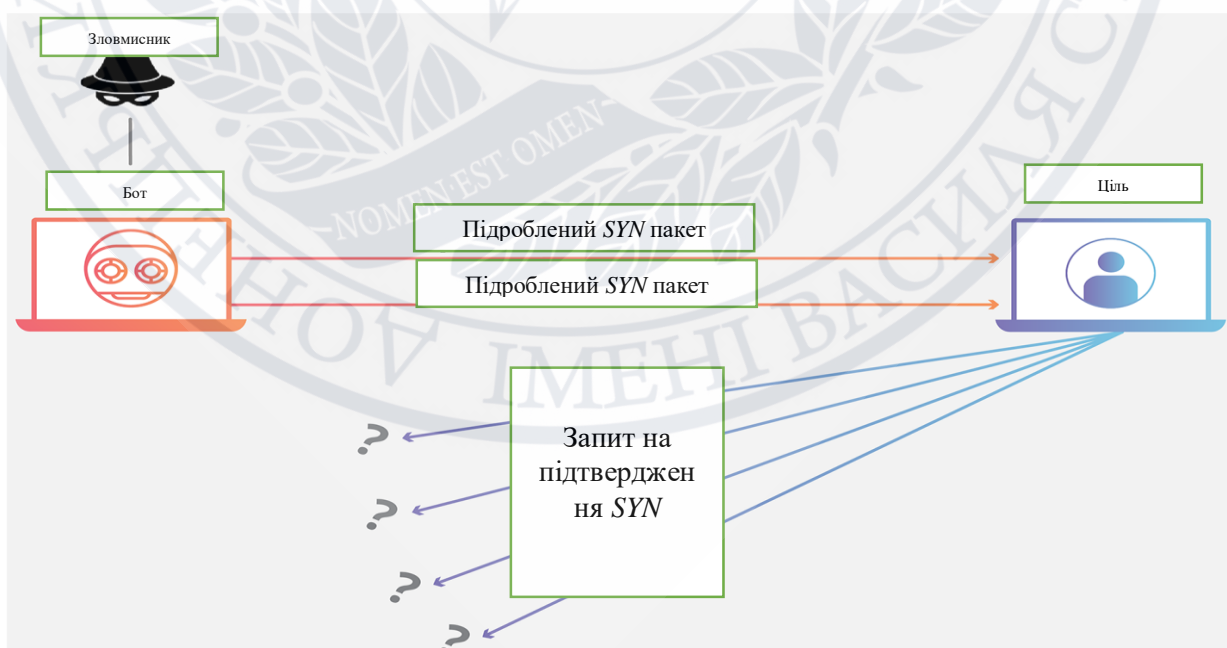


Рис. 3.4. Атака *SYN flood*

### 3) Об'ємні атаки.

Даний тип *DDoS* атаки створює перевантаження, використовуючи всю доступну смугу пропускання між цілю та Інтернетом. Великі обсяги даних відправляються цілі з використанням форми посилення або інших засобів створення масового трафіку, таких як запити від “ботнету” рис 3.5. Створюється довга відповідь, що надсилається жертві. Виконуючи запит до відкритого *DNS*- сервера з підробленим *IP* - адресою – жертви, цільовий *IP*-адрес потім отримує відповідь від сервера.

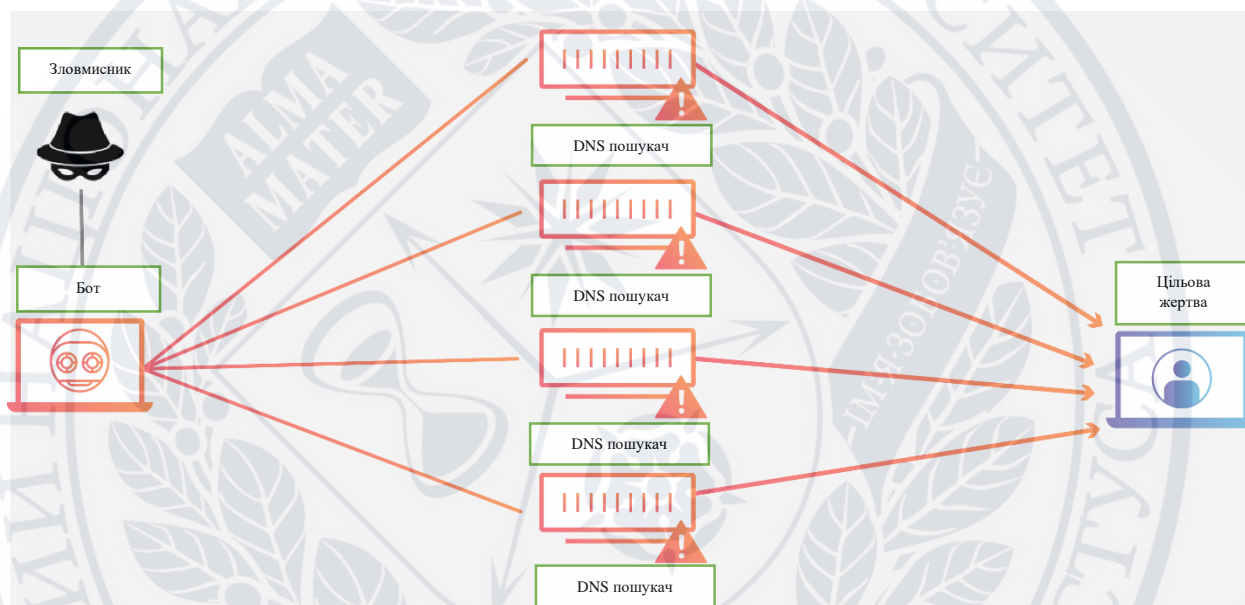


Рис. 3.5. Атака *DNS*

## 3.2 Методи захищеності *DDoS* атак та їх реалізація

### 1. Захист від *DDoS* атак рівня четвертого.

Захист від *DDoS* атак на четвертому рівні (транспортному) засновані цілком на підробці *IP* адреса, а тому захиститись від цього не складе проблем. Зазвичай використовується *IPSec*, задля запобігання спуфінгу *IP*. Однак якщо зловмисник використовує велику кількість вузлів, цей спосіб може не спрацювати. Адже зловмисники можуть розсіюватись по різних часових поясах, що в свою чергу не дасть змогу для відслідковування *IP*. Інші типи



захисту включають використання *IDS* на основі програмних засобів *Snort*, який є доступним кодом системи виявлення вторгнень та порушень у мережі (*NIDS*).

Також існує ідея, що виявляє атаку за допомогою шаблону введення потоку та механізму обробки за допомогою брандмауера. Доведено, що використання шаблонів записів потоку, може надати безпеку більш точного виявлення вихідної *IP*-адреси злоумисника.

## 2. Захист від *DDoS* атак рівня сьомого.

Дані атаки є більш небезпечними, адже не обмежуються використанням спуфінгу *IP*. Атаки такого роду можна запускати, маючи мінімальну пропускну здатність. Можливий спосіб – *web* – сервер не приймає запит, поки він не являється повним запитом *HTTP*. Результатом є часткові запити які мають лише фрагмент *HTTP* заголовку, що відхилені сервером.

Існує інше рішення, це можливість задіяти математичне обчислення на стороні клієнта перед установкою з'єднання. Це в свою чергу призведе до численних витрат на обрахування зі сторони клієнта.

В таблиці 3.6 представлено узагальнення усіх методів захисту.

Таблиця 3.6.  
Методи захисту

Атаки	Методи захисту
На четвертому рівні	<ul style="list-style-type: none"><li>• Підміна <i>IP</i> адресу</li><li>• <i>IPSec</i></li><li>• Використання <i>IDS</i> та <i>NIDS</i></li><li>• Аналіз структури виходу потоку</li></ul>
На сьомому рівні	<ul style="list-style-type: none"><li>• Повний прийом <i>HTTP</i> запитів</li></ul>

- Математичне обчислення на стороні клієнта до процесу з'єднання

Задля реалізації захисту від *DDoS* атак, було використано мову програмування *Python 3.7*, а також бібліотеку, що допомагає зберігати стан мережі *pickle* та бібліотеку, що допомагає у побудові нейронної мережі *PyBrain3*. Необхідна в роботі бібліотека для роботи з даними *numpy* та бібліотека *user\_agent*, яка отримує інформацію по самому *User-Agent*. На рис. 3.7 зображені усі перелічені бібліотеки для обов'язкового підключення.

Важливим є факт створення методу, що в свою чергу переводить інформацію про *log* в представлення у вигляді програми, що слугує у подальшому для його переведу в інформаційний потік для нейронної мережі. Даний метод необхідний для створення уже моделі, що буде навчатись. На рис. 3.8 представлено переведення у вигляді програми, *log* у модель. А для подальшого запуску програми, що виконує парсинг логів, вводиться команда "*python3.py main -m start*", як зображено на рис. 3.9.

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  from collections import namedtuple
4  from re import compile as reg_match, IGNORECASE
5
6  import numpy as np
7  from pickle import dump, load, Unpickler
8  from optparse import OptionParser
9  from user_agents import parse as user_agent_parser
10 from logging import basicConfig, DEBUG, INFO, getLogger
11 from pybrain3.datasets import ClassificationDataSet
12 from pybrain3.structure.modules import SoftmaxLayer, SigmoidLayer
13 from pybrain3.tools.shortcuts import buildNetwork
14 from pybrain3.supervised.trainers import BackpropTrainer
15 from pybrain3.utilities import percentError
16 from pybrain3.tools.xml.networkwriter import NetworkWriter
17 from pybrain3.tools.xml.networkreader import NetworkReader
18
19
20 class MetaSingleton(type):
21     """
22     Metaclass: Singleton
23     """
24     _instances = {}
25
26     def __call__(cls, *args, **kwargs):
27         if cls not in cls._instances:
28             cls._instances[cls] = super(MetaSingleton, cls).__call__(*args, **kwargs)

```

Рис. 3.7. Обов'язкові бібліотеки для створення програми

Наступний крок передбачає процес навчання, після того як виконалось зчитування *log*. На рис. 3.10 показано, що сам процес навчання розподілений на 10 пунктів. Для того щоб дізнатись де *bot*, а де безпечні користувачі, програмне забезпечення розпочинає процес зчитування всіх *log* рис. 3.11.

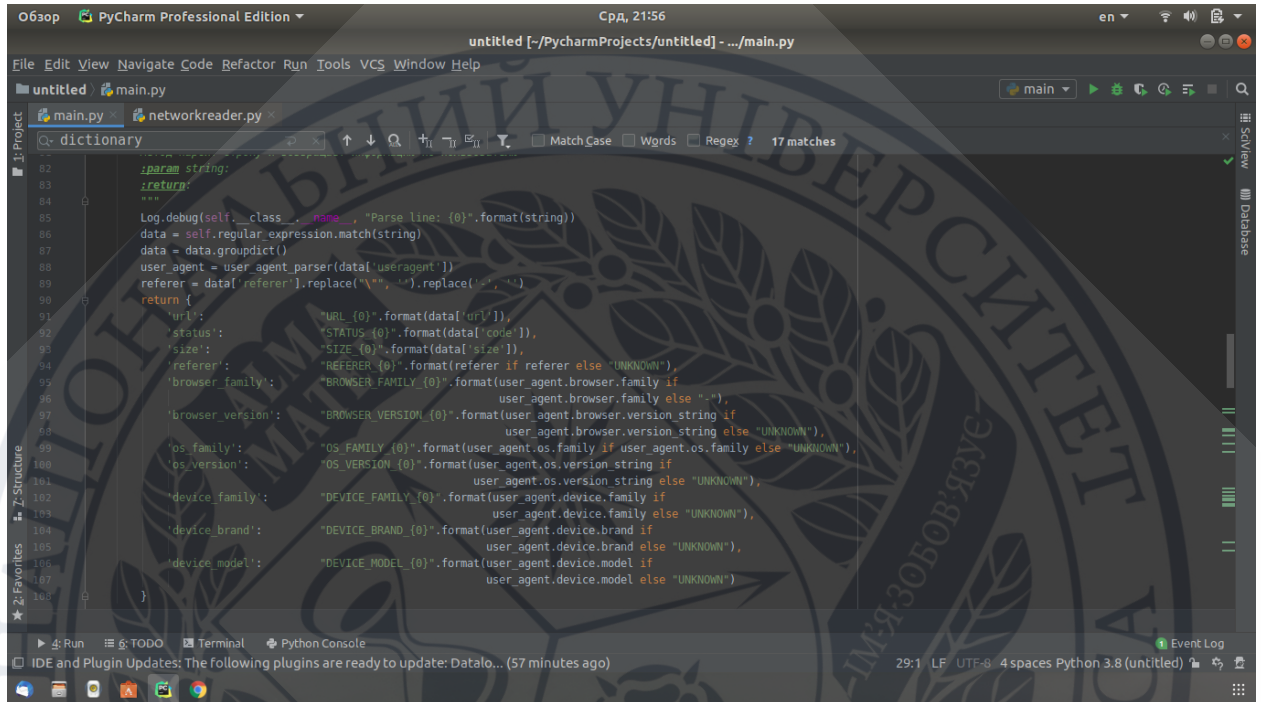


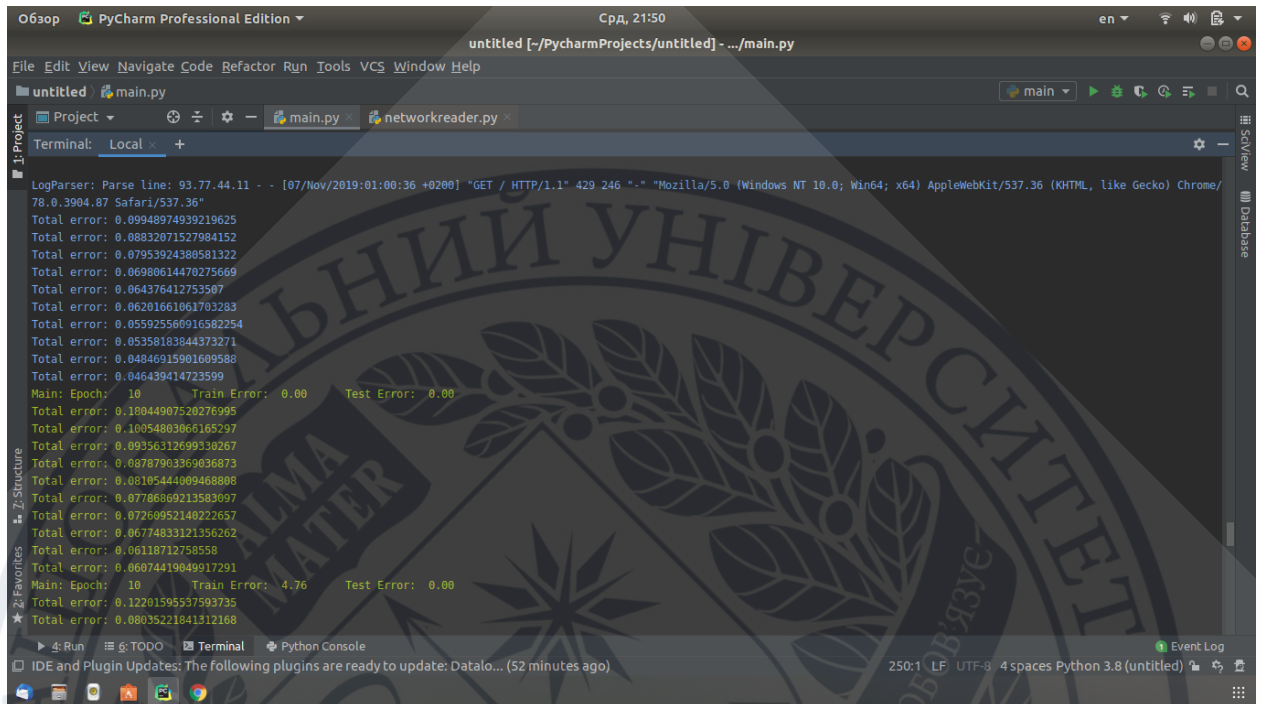
Рис. 3.8. *Log* переводиться в модель



Рис. 3.9. Подальший запуск програми та зчитування *log*



Коли процес зчитування *log* завершився, наступним кроком нейрона мережа може вивести усіх *bot*, що знайшла рис. 3.12.



```
LogParser: Parse line: 93.77.44.11 - - [07/Nov/2019:01:00:36 +0200] "GET / HTTP/1.1" 429 246 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
Total error: 0.09948974939219625
Total error: 0.08832071527984152
Total error: 0.07953924380581322
Total error: 0.06980614470275669
Total error: 0.064376412753507
Total error: 0.06201661061703283
Total error: 0.055925560916582254
Total error: 0.05358183844373271
Total error: 0.04846915901609588
Total error: 0.046439414723599
Main: Epoch: 10 Train Error: 0.00 Test Error: 0.00
Total error: 0.10844907520276995
Total error: 0.10854803066165297
Total error: 0.09356312699130267
Total error: 0.09787003360936873
Total error: 0.08105444009468888
Total error: 0.07786809213583097
Total error: 0.07269952140222657
Total error: 0.06774833121356262
Total error: 0.06118712758558
Total error: 0.06074419049917291
Main: Epoch: 10 Train Error: 4.76 Test Error: 0.00
Total error: 0.12201595537593735
Total error: 0.08035221841312168
```

Рис. 3.10. Процес навчання



```
LogParser: Parse line: 93.77.44.116 - - [07/Nov/2019:01:00:29 +0200] "GET / HTTP/1.0" 200 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.116 - - [07/Nov/2019:01:00:29 +0200] "GET / HTTP/1.1" 200 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.116 - - [07/Nov/2019:01:00:29 +0200] "GET / HTTP/1.0" 200 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.116 - - [07/Nov/2019:01:00:29 +0200] "GET / HTTP/1.1" 200 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.116 - - [07/Nov/2019:01:00:30 +0200] "GET / HTTP/1.0" 200 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.116 - - [07/Nov/2019:01:00:30 +0200] "GET / HTTP/1.1" 200 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.15 - - [07/Nov/2019:01:00:35 +0200] "GET / HTTP/1.1" 499 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.15 - - [07/Nov/2019:01:00:35 +0200] "GET / HTTP/1.1" 499 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
LogParser: Parse line: 93.77.44.15 - - [07/Nov/2019:01:00:35 +0200] "GET / HTTP/1.1" 499 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.87 Safari/537.36"
```

Рис. 3.11. Процес зчитування *log*

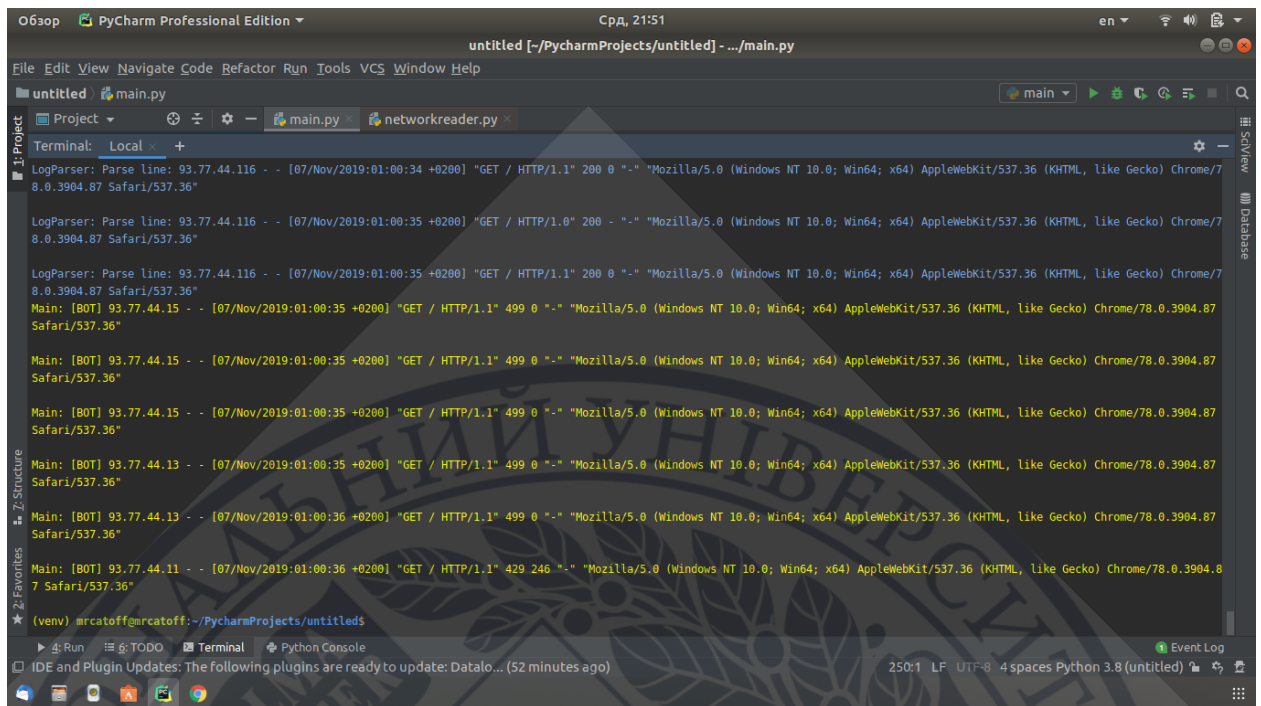


Рис. 3.12. Процес виводу *bot*

Дана програма не потребує щоразу перезавантаження, адже може зберігати свої дані про навчання для подальшого використання та запуску уже навченої системи, що у разі спрощує використання і час виконання.

## ВИСНОВОК

У роботі розглянуто розвиток досить перспективних методів захисту та безпеки програмних систем. Наводиться короткий аналіз сучасності цієї тематики на основі стану проблематики у відсутності безпеки. При аналізі усіх загроз та загальної складності мережевих систем, виділяється різка еволюція застосованих технологій для вирішення питань, пов'язаних з безпекою.

Захист від *DDoS* атак є актуальною та важливою темою на даний момент для подальшого вивчення та дослідження. Надано класифікацію *DDoS* атак, які можливі в *IPv4* і *IPv6* та на окремих рівнях моделі *OSI*. Уточнено та проаналізовано методи захисту від різних типів атак програмних систем. Наводиться опис розробленого методу захисту від *DDoS* атак на основі нейронної мережі, яка навчається та має здатність фільтрувати користувачів. В цілому, можна зробити висновок про велику та стрімко наростаючу кількість атак, через низьку ефективність захищеності мережі. Дану проблему можна вирішити за рахунок ретельного вивчення незахищених сторін.



## Список літератури

- [1] Cybercrime Magazine. URL: <https://cybersecurityventures.com/>.
- [2] Гнеушев В.А., Кравец А.Г., Козунова С.С., Бабенко А.А. Моделирование сетевых атрибутов злоумышленников в корпоративной информационной системе, Промышленные АСУ и контроллеры. 2017, № 6.
- [3] Nwokedi Idika and Aditya P. Mathur, A Survey of Malware Detection Techniques, Department of Computer Science, Purdue University, West Lafayette, IN 47907 (2007).
- [4] Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation. Bruce Dang; 2014 рік.
- [5] Основы кибербезопасности. Стандарты, концепции, методы и средства обеспечения. Белоус А.И., Солодуха В.А.
- [6] Mohan V. Pawar and Anuradha J., “Network security and types of attacks in network,” ICC-2015. URL: [https://ac.els-cdn.com/S1877050915006353/1-s2.0-S1877050915006353-main.pdf?tid=21866302-2e58-4f1e-88d0-7d3b9825e011&acdnat=1543497106\\_74f03131d7fc65a2469e9708a18cc54c](https://ac.els-cdn.com/S1877050915006353/1-s2.0-S1877050915006353-main.pdf?tid=21866302-2e58-4f1e-88d0-7d3b9825e011&acdnat=1543497106_74f03131d7fc65a2469e9708a18cc54c).
- [7] Check Point Software Technologies Ltd., Software Blade Architecture. URL: <https://www.checkpoint.com/downloads/product-related/brochure/Software-Blades-Architecture.pdf>.
- [8] Кибербезопасность. Томас Паренти и Джек Домет
- [9] DARPA, ‘Cyber Grand Challenge’.
- [10] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, “Unleashing Mayhem on binary code,” Proc. IEEE Symp. on Security and Privacy (2012), pp. 380–394.
- [11] Оладько В.С. Программный комплекс для определения закономерностей распределения атак злоумышленников, Вопросы кибербезопасности. 2015, № 1 (9).

[12] A. Nguyen-Tuong, D. Melski, J. W. Davidson, M. Co, W. Hawkins, J. D. Hiser, D. Morris, D. Nguen, and E. Rizzi, "Xandra: An autonomous cyber battle system for the cyber grand challenge," IEEE Security & Privacy, Vol. 16, No. 2, 42–53 (2008).

[13] Mechaphish Github Repository.  
URL: <https://github.com/mechaphish/mecha-docs>.

[14] The Practice of Network Security Monitoring: Understanding Incident Detection and Response. Richard Bejtlich; 2013 рік.

[15] 6. Косенко М.Ю., Мельников А.В. Вопросы обеспечения защищенности информационных систем от ботнет атак, Вопросы кибербезопасности. 2016, № 4 (17).

[16] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequence," AI 2016: Advances in Artificial Intelligence, Proc. 29th Australasian Joint Conference, Hobart, TAS, Australia, December 5–8 (2016), pp. 137–149.

[17] M. Cova, V. Felmetzger, and G. Banks, "Static detection of vulnerabilities in x86 executables," 22nd Annual Computer Security Applications Conference (ACSAC'06) (2006). <https://doi.org/10.1109/ACSAC.2006.50>.

[18] Кибербезопасность: стратегии атак и обороны», Юрий Диогенес, Эрдаль Озкайя.

[19] Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu, "Vulpecker: An automated vulnerability detection system based on code similarity analysis," Proc. 32nd Annual Conf. on Computer Security Applications, ACSAC'16 (2016), pp. 201–213.

[20] H. Flake, "Structural comparison of executable objects," Proc. IEEE Conf. on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) (2004), pp. 161–173.

[21] Threat Modeling: Designing for Security. Adam Shostack; 2014 рік.

[22] M. Mouzarani, B. Sadeghiyan, and M. Zolfaghari, "Detecting injection vulnerabilities in executable codes with concolic execution," Proc. 8th IEEE Intern.

Conf. on Software Engineering and Service Science (ICSESS) (2017). <https://doi.org/10.1109/ICSESS.2017.8342862>.

[23] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, “Unleashing MAYHEM on binary code,” SP’12 Proc. IEEE Symp. on Security and Privacy (2012). <https://doi.org/10.1109/SP.2012.31>.

[24] Информационная безопасность. Защита и нападение», Андрей Бирюков.

[25] G. Lee, “How to formally model features of network security protocols,” Intern. J. of Security and Its Applications, Vol. 8, No. 1, 423–432 (2014). URL: [formal.hknu.ac.kr/Publi/ijasia.pdf](http://formal.hknu.ac.kr/Publi/ijasia.pdf).

[26] 4. Android Hacker's Handbook. Joshua J. Drake; 2014 рік.

[27] J. Dodds, “Formal methods and the KRACK vulnerability,” Galois Inc. (2017). URL: <https://galois.com/blog/2017/10/formal-methods-krack-vulnerability/>.

[28] J.-S. Coron, “Formal verification of side-channel countermeasures via elementary circuit transformations,” Proc. 16th Intern. Conf., ACNS 2018, Leuven, Belgium, July 2–4 (2018), pp. 65–82. URL: <https://eprint.iacr.org/2017/879.pdf/>.

[29] Cybersecurity Attacks — Red Team Strategies. Іоґанн Реберґер.

[30] S. Jha, O. Sheyner, and J. Wing, “Two formal analyses of attack graphs,” Proc. 15th IEEE Computer Security Foundations Workshop (2002). URL: <https://ieeexplore.ieee.org/document/1021806>.

[31] The Art of Computer Virus Research and Defense. Peter Szor; 2005 рік.

[32] V. Ferman, D. Hutter, and R. Monroy, “A model checker for the verification of browser based protocols,” Comp. y Sist. Vol. 21, No. 1 (2017). URL: <http://www.scielo.org.mx/pdf/cys/v21n1/1405-5546-cys-21-01-00101.pdf>.

[33] W3C. “Web Services Architecture”. <http://www.w3c.org/TR/2004/NOTE-ws-arch-20040211>, February 2004.

[34] Кибербезопасность: стратегии атак и обороны» — Юрий Диогенес, Эрдадь Озкайя.



[35] L. Tobarra, D. Cazorla, F. Cuartero, and G. Diaz, “Application of formal methods to the analysis of web services security,” URL: <https://www.semanticscholar.org/paper/Application-of-formal-methods-to-the-analysis-of-tobarra-cazorla/544d181da33da5439efcf49f31d50116355410d9>.

[36] Social Engineering: The Science of Human Hacking 2nd Edition. Christopher Hadnagy.

[37] Agile Application Security: Enabling Security in a Continuous Delivery Pipeline — Рич Смит, Лора Белл, Майкл Брантон-Сполл, Джим Бёрд.

[38] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffei, “Keys to the cloud: Formal analysis and concrete attacks on encrypted web storage,” URL: <http://antoine.delignat-lavaud.fr/doc/post13.pdf>. URL: <https://hal.inria.fr/hal-00863375/file/keys-to-the-cloud-post13.pdf/>.

[39] The Tangled Web: A Guide to Securing Modern Web Applications — Михал Залевский.

[40] Анализ вредоносных программ» — К. Монаппа.

[41] Обнаружение нарушений безопасности в сетях» — Стивен Норткат, Джуди Новак.

[42] M. Bugliesi, S. Calzavara, and R. Focardi, “Formal methods for web security,” Università Ca’ Foscari Venezia. URL: [https://www.researchgate.net/publication/308004472\\_Formal\\_methods\\_for\\_Web\\_security](https://www.researchgate.net/publication/308004472_Formal_methods_for_Web_security).

[43] The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. Mark Dowd, Justin Schuh.

[44] Tribe of Hackers: Cybersecurity Advice from the Best Hackers in the World. Marcus J Carey.

[45] D. Gilbert and A. Letichevsky, “Model for interaction of agents and environments,” in: Bert D., Choppy C. (Eds.). Recent Trends in Algebraic Development Technique, Wadt 1999. LNCS, Vol. 1827, Springer-Verlag, Berlin–Heidelberg (2000), pp. 311–328.

[46] Reversing: Secrets of Reverse Engineering 1st Edition. Eldad Eilam.

[47] A. Letichevsky, O. Letychevskiy, and V. Peschanenko, "Insertion modeling and its applications," *Computer Sci. J. of Moldova*, Vol. 24, Issue 3, 357–370 (2016).

