ПРОГРАМНО-ТЕХНІЧНІ КОМПЛЕКСИ

О.О. ЛЕТИЧЕВСЬКИЙ, В.С. ПЕСЧАНЕНКО, Я.В. ГРИНЮК, В.Ю. РАДЧЕНКО, В.М. ЯКОВЛЕВ

УДК 004.05, 004.4[2+9], 004.94, 519.7

ОГЛЯД СУЧАСНИХ МЕТОДІВ ЗАХИЩЕНОСТІ ТА БЕЗПЕКИ ПРОГРАМНИХ СИСТЕМ

Анотація. Показано, що безпека та захищеність програмних ресурсів нині є однією з найбільш актуальних проблем в IT-галузі, оскільки дії зловмисників стають дедалі більш витонченими, а збитки від кібератак збільшуються. Традиційні методи боротьби з кібератаками втрачають свою ефективність, тому розроблення нових методів і засобів захисту програмних ресурсів є нагальною потребою. Особливо цікавими і перспективними є розробки, які ґрунтуються на формальних методах з використанням сучасних алгебраїчних теорій.

Ключові слова: алгебраїчне моделювання, алгебра поведінок, кібербезпека, інсерційне програмування, формальні методи, символьні методи, символьне моделювання, пошук вразливостей.

Проблема безпеки та захищеності програмних ресурсів є однією з найбільш актуальних проблем, що активно вивчаються та обговорюються на міжнародних конференціях. Поява мережевих програм-здирників (ransomware), які вимагають викуп, свідчить про безпрецедентний рівень досконалості технологій атак. Зловмисники дедалі більш вправно уникають виявлення та ефективно використовують прогалини в системі безпеки. Важливість розв'язання цієї проблеми підкреслюється тим, що згідно з [1] обсяг збитків, завданих кібернападами, сягнув трьох трильйонів доларів США у 2015 р., а в 2021 р., за оцінками, становитиме шість трильйонів доларів США.

У цій статті наведено огляд основних понять та нагальних проблем у галузі захищеності та безпеки. Особливу увагу приділено засобам кіберзахисту (cybersecurity), побудованим з використанням формальних методів на основі сучасних алгебраїчних теорій.

ОСНОВНІ ПОНЯТТЯ ТА КЛАСИФІКАЦІЯ КІБЕРНАПАДІВ

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, хмар-

Таблиця 1

Тип зловмисної програми	Опис зловмисної програми
Вірус	Програма, що може приєднувати себе до іншої програми та виконуватися, використовуючи її в середовищі, яке є «інфікованим»
Хробак (Worm)	На відміну від вірусу ця програма може виконуватися незалежно та розмножувати свої копії на інших комп'ютерних ресурсах
Троян	Програма, що зовні має вигляд корисної, а зловмисну дію виконує в кінці свого життєвого циклу
Шпигун (Spyware)	Тип зловмисної програми, яка відслідковує дії користувача з метою отримання важливої інформації, наприклад паролю, номерів рахунків та ін.
Вимагач (Ransomware)	Програма, яка таємно інсталюється на комп'ютері користувача та зашифровує дані з метою вимагання викупу за коректне дешифрування
Підробка (Scareware)	Шкідливі комп'ютерні програми, призначені для того, щоб змусити користувача купувати та завантажувати непотрібне та потенційно небезпечне програмне забезпечення, наприклад підроблений антивірусний захист
Імітатор (Bot)	Програма, яка імітує дії користувача за допомогою відповідних інтерфейсів згідно з встановленим розкладом або алгоритмом з метою мережевих атак, шахрайства, поширення непотрібної реклами та іншого перевантаження інтернет-каналу. Такою програмою є, наприклад, DDoS-атака
Схованки (Rootkits)	Набір програмних засобів, які дають змогу неавторизованому користувачеві отримати контроль над комп'ютерною системою без виявлення з метою викрадення даних

не середовище, блокчейн). Атаки можуть використовувати зловмисні програми, які активізуються під час проникнення у комп'ютер користувача. У табл. 1 визначено основні типи зловмисних програм відповідно до способів їхнього проникнення [2].

Атаки на веб-сайти або веб-сервіси, розподілені та локальні мережі, хмарне середовище враховують та використовують особливості, пов'язані з транспортуванням даних до мережевих вузлів. Проблеми безпеки в цій галузі зумовлені різними типами атак, які можуть застосовуватися для порушення нормального функціонування мережевих систем (активні атаки) або викрадення інформації, переданої мережею (пасивні атаки). Згідно з [3] є такі типи мережевих атак, 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. 2.

ТРАДИЦІЙНІ ЗАСОБИ ЗАХИСТУ МЕРЕЖ ТА СЕРЕДОВИЩ

Існують такі різновиди сучасних традиційних систем захисту.

• Системи виявлення зловмисників (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.

• 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 з боку застосунків. 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.

Основними гравцями на ринку традиційного захисту від кібератак є компанії Check Point, CISCO, McAffee.

Таблиця 2

Назва атаки	Опис атаки
Підробка (Spoofing)	Стороння програма, яка використовує фальшиву ідентифікацію, змушуючи систему, яку атакують, змінювати топологію мережі
Модифікація (Modification)	Зловмисна система змінює маршрутизацію в мережі
Тунельна атака (Wormhole)	Система, що атакує, отримуючи пакет даних, переспрямовує його в інше місце, імітуючи для системи, яку атакують, найкоротший шлях в мережі
Фабрикація (Fabrication)	Генерація зловмисною системою фальшивого повідомлення для мережевої маршрутизації, яке порушує роботу пристроїв-маршрутизаторів
Відмова в обслуговуванні (Denial-of-Service, DDoS)	Уповільнення або порушення роботи системи, яку атакують, за рахунок перевантаження мережевого трафіку і надмірної кількості запитів, що одночасно надходять у систему
Вигрібна яма (Sinkhole)	Створення перешкод, що не дають змоги системі, яку атакують, отримувати повну і коректну інформацію. Метою є вибіркова модифікація, перенаправлення і втрата інформації
Сивілла (Sybil)	Атака з використанням безлічі зловмисних вузлів мережі, під час якої один такий вузол передає секретні ключі іншим
Аналіз трафіку (Traffic analysis)	Здійснення аналізу обсягу даних, переданих між вузлами, що атакуються
Підслуховування (Eavesdropping)	Атаки, які часто здійснюються у мережах мобільного зв'язку з метою викрадення важливої інформації, зокрема секретних ключів
Моніторинг (Monitoring)	Перехоплення важливої інформації в мережі без її модифікації
Чорна діра (Black hole)	Система, що атакує, використовує протоколи маршрутизації для того, щоб представити себе найкращим вузлом для маршрутизації даних для системи, яку атакують
Поспіх (Rushing)	Система, що атакує, здійснює підміну пакетів, які передаються між вузлами, що атакуються, потім дублює ці пакети знову і знову, створюючи враження високонавантаженої системи
Повтор (Replay)	Система, що атакує, може повторювати мережеві пакети та/або затримувати їх, а в процесі оброблення викрадати конфіденційну інформацію, наприклад паролі
Візантійська атака (Byzantine)	Вузли системи, що атакує, вбудовуються між вузлами, що атакуються, використовуючи неоптимальні маршрути, створюючи цикли в маршрутах, або вибірково «втрачаючи» пакети
Розкриття розташування (Location disclosure)	Збирання інформації про вузли, що атакуються, і маршрути, моніторинг трафіка з подальшим застосуванням інших видів атак

Компанія Check Point працює в цій галузі з 1993 р. і розробляє комплексні продукти із захисту програмних систем як для хмарного середовища, так і для мобільних пристроїв. Одним з останніх є продукт Check Point Software Blade Architecture [4] (рис. 1), 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 знаним 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 сере-

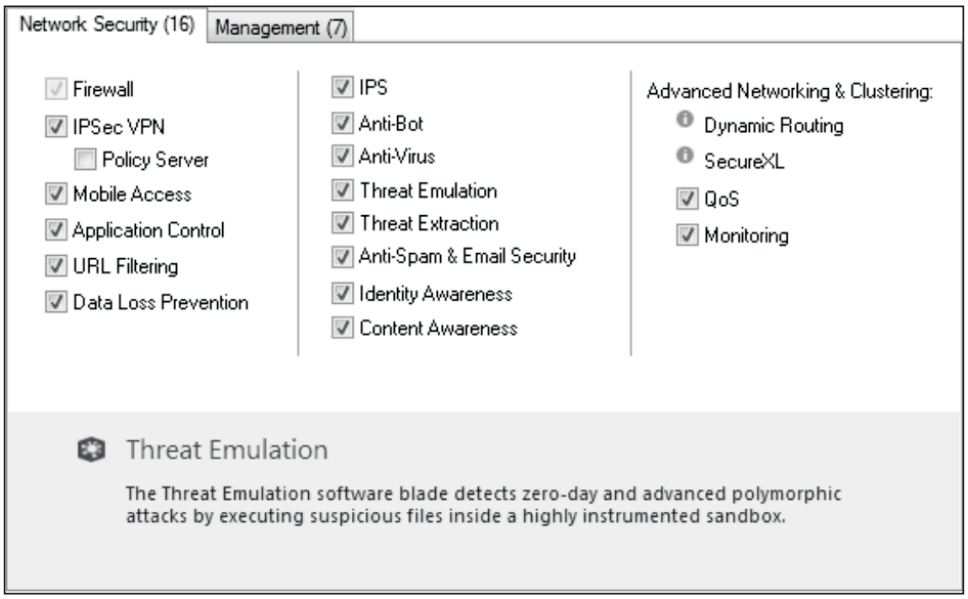

Рис. 1. Вікно налаштування системи Check Point Software Blade Architecture

довища і мають низку недоліків. Насамперед, це досить велика кількість помилкових виявлень, особливо при підвищенні чутливості аналізатора. Ця риса найбільшою мірою притаманна евристичному методу. Несвоєчасність виявлення атаки зумовлена значною тривалістю процесу виявлення. Це може статися, коли нападник використовує такі засоби маскування, як обфускація (маскування поведінки шляхом уведення «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. 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'язувачі задач та машини автоматичного доведення теорем.

ФОРМАЛЬНІ МЕТОДИ, ЗАСТОСОВАНІ В МЕТОДАХ КІБЕРЗАХИСТУ

Для розв'язання проблеми безпеки та захищеності або проблеми кіберзахисту (cybersecurity) розглядають такі 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 р. компанією Darpa було проведено так званий Cyber Grand Challenge [5] — змагання низки програмних систем для виявлення вразливостей у запропонованих програмах. 40 команд зі своїми системами змагалися у спроможності виявити максимальну кількість вразливостей у програмах впродовж обмеженого проміжку часу. Основними переможцями стали:

- система Mayhem [6] команди ForAllSecure з Пітсбургу, що виграла конкурс. У системі було застосовано символьне виконання програм із використанням розв'язної системи Microsoft Z3 та символьну модель пам'яті;

• система Xandra [7] команди з Нью-Йорку, у якій було використано власний символьний шукач вразливостей;

• система Mechanical Phish [8], у якій було використано метод розпорошення («фаззінг», Fuzzing) у поєднанні з символьним виконанням.

Особливістю цього конкурсу було те, що кожний переможець застосовував алгебраїчні методи та дедуктивні засоби (розв'язувачі задач та автоматичне доведення теорем).

Компанія Amazon є однією з провідних, яка використовує формальні методи в захисті хмарного середовища від зовнішніх атак. Вона має понад 1500 сервісів, отже рівень небезпеки втручання є досить високим. Для точного визначення атак компанія дуже активно застосовує формальні методи, зокрема, більше десяти дедуктивних інструментів.

Метод розпорошення є основним засобом знаходження невідомих уразливостей (уразливості нульового дня). Він полягає в тому, що під час реалізації різних сценаріїв виконання програми генеруються критичні величини змінних у програмі, які можуть призвести до вразливих дій. Однією з найвідоміших Fuzzing-машин є AFL (American Fuzzy Lop), створена компанією Google [9].

Загалом можна виділити такі напрями використання формальних методів.

Аналіз послідовності системних викликів. Системний виклик — це програмний спосіб запиту сервісів у операційної системи. Прикладами системних викликів можуть слугувати функції для зчитування-запису файлів або робота з мережею. Більшість програм використовують системні виклики для спілкування мережею, збереження налаштувань користувача та інших подібних дій. Система виявлення атак перехоплює та формує послідовності системних викликів для конкретного процесу. Також є відкриті масиви з послідовностями системних викликів, які відповідають конкретним вразливостям. З використанням відомих послідовностей формуються алгоритми, які визначають, чи є ці послідовності потенційними атаками.

Для побудови алгоритмів класифікації [10] вразливостей використовуються різні формальні методи. Один з них ґрунтується на алгоритмі з використанням так званих вікон, які рухають по послідовності системних викликів. Це досить популярний алгоритм, адже фіксований розмір вікна дає змогу регулювати швидкодію та використовувати різні статистичні методи. Можна застосовувати приховані марковські моделі, рекурентні нейронні мережі, але водночас вибір параметрів для цих методів є достатньо складним і компромісним, оскільки потрібно уникнути перенавчання та зменшити використовувані ресурси. Мовний підхід передбачає формування простих n-грамів подібно до того, як це роблять в обробленні текстів, але навіть на невеликих n-грамах розмір потенційного простору векторів стає дуже великим.

Перевагами аналізу послідовності системних викликів є універсальність, адже системні виклики є уніфікованим інтерфейсом роботи між програмою та операційною системою. Також є можливість знаходити атаки нульового дня, тобто атаки, які ще не були виявлені. Цей підхід також має недоліки через малу кількість даних для навчання, а такі системи виявлення атак мають великий відсоток помилок першого роду і впливають на швидкодію програм в цілому, адже існують деякі накладні витрати під час зчитування системних викликів.

Системи виявлення атак на основі символьних обчислень. Символьне виконання складається з інтерпретації коду, коли під час виконання програми використовуються символи, що представляють довільні значення замість конкретних введень, наприклад рядки чи цифри. При цьому обчислювання здійснюється, як у конкретному виконанні, за винятком того, що значення, оброблені за програмою, можуть бути символьним виразом над вхідними символами. Завдяки цьому символьне виконання генерує всі можливі шляхи виконання програми. Для кожного шляху виводиться множина вхідних даних, які формують такий шлях виконання. Ці дані є корисними для діагностики як низькорівневих збоїв, так і високорівневих властивостей програми.

Часто немає доступу до вихідного коду виконуваної програми, тому предметом аналізу виступає програма у дизасембльованому вигляді. Дизасемблювання — процес трансляції програми з машинних кодів у мову асемблер (інструкції процесора). Таким чином, символьне обчислення виконується на основі асемблерного представлення програми.

Одним із способів використання результатів символьного обчислення є ідентифікування чутливих місць, де використовуються ненадійні дані [11]. Ненадійні дані — це дані, які подаються програмі на вхід і не проходять належного оброблення; оброблення може бути різним залежно від характеру даних та місць їхнього використання. Чутливі місця — це місця програми, які можуть за деяких обставин бути джерелом аномальної поведінки програми, привілейованого доступу тощо. Наприклад, виклик функції system запускає програму, яка описується вхідним аргументом. У тому разі, якщо значення, яке передається у system, обробляється некоректно, зловмисник може запускати будь-які програми, а це є серйозною вразливістю. Такий підхід довів свою практичність у виявленні вразливостей, пов'язаних із переповненням буферу, переповненням цілочисельних змінних, віддаленим виконанням коду.

Однак у випадку використання символьних обчислень кількість можливих шляхів зростає експоненційно до кількості інструкцій навіть у невеликих програмах. Якщо не брати до уваги найпростіші випадки, повний аналіз може потребувати великого обсягу часу та обчислювальних потужностей. Для подолання цього застосовуються різні техніки, які оптимізують виконання, але при цьому зменшують точність методу. До того ж внаслідок використання символьних обчислень можна генерувати дані, що підтверджують вразливість, розв'язуючи умови символьного шляху.

Отже, цей підхід є достатньо ефективним для статичного аналізу виконуваних файлів. Він дає змогу будувати шляхи ненадійних даних та знаходити вразливості, передбачає потенційну можливість автоматичного збирання великих масивів тестових даних для постійного тестування програмної системи під час її розвитку.

Системи виявлення атак на основі конкретних та символьних обчислень. Ці системи з'явилися у відповідь на обмеження систем, що ґрунтуються лише на символьних обчисленнях та називаються concolic-обчисленнями. Ці обчислення є поєднанням символьних обчислень та конкретного виконання. Використовуючи символьні обчислення, можна генерувати всі можливі шляхи виконання програми, а також вхідні дані, які відповідають таким шляхам. Ці дані використовуються для тестування та знаходження вразливостей у програмі. Проте у такого підходу є недоліки. Наприклад, якщо у програмі є цикли та рекурсивні виклики, для виконання символьних обчислень може знадобитися надто великий обсяг часу та ресурсів. У concolic-обчисленнях частково використовуються конкретні дані, а не їхнє символьне представлення для того, щоб обійти проблеми символьного виконання [12].

Процес concolic-обчислень починається з налаштування середовища для виконання програми. У сучасних операційних системах існує можливість запускати програму у спеціальному режимі так, що програма виконується покроково, інструкція за інструкцією, і на кожному кроці можна зчитувати значення усіх регістрів та значення у пам'яті. Надалі використовуються різні техніки. Однією з можливих технік є побудова шляху даних, що подаються на вхід, наприклад нове обмеження додається кожного разу, коли є інструкція переходу jump, що залежить від вхідних даних [13]. Далі на основі зібраних обмежень будуються тестові дані, які можуть показати можливі вразливості у системах на основі символьного обчислення. Цими даними також можна скористатися для того, щоб знайти більше можливих шляхів виконання програми під час конкретного виконання.

Отже, метою систем на базі concolic-обчисленнь є усунення тих обмежень, які є у системах, що ґрунтуються на чисто символьних обчисленнях. Прикладом таких обмежень є експоненційне зростання кількості шляхів відносно кількості

інструкцій. Ці системи також дають змогу обробляти випадки, коли в програмі є цикли та рекурсивні виклики. Приклади реалізації таких систем довели їхню ефективність під час знаходження таких вразливостей, як переповнення буфера, різноманітних вразливостей, пов'язаних із цілочисельною арифметикою, діленням на нуль, звертанням до нульового вказівника. Водночас розробити і підтримувати таку систему більш складно, ніж систему, що ґрунтується лише на символьних обчисленнях, адже потрібно побудувати інфраструктуру для запуску програм та їхнього виконання з конкретними даними. Для цього потрібно вибрати програмне забезпечення, яке надасть необхідне середовище для тестування. Сама специфіка програми може вимагати якогось особливого програмного забезпечення. Це означає, що для кожної програми систему виявлення вразливостей потрібно адаптувати. До того ж розроблення початкового набору тестових даних здійснюється вручну та не може бути автоматизоване. Це означає, що такі системи розробляються, як доповнення до наявних програм, які треба перевіряти, та не можуть бути системами загального призначення.

Виявлення вразливостей у повторно використаному коді. Під час розроблення програмного забезпечення програмісти копіюють код для повторного використання. Водночас, копіюючи код, програміст також копіює його вразливості, що відповідно впливає на безпечність програмної системи в цілому [10]. Наприклад, чверть коду ядра відкритої операційної системи Linux повторюється; існує навіть деяка кількість сегментів, що повторюються до восьми разів. Також було виявлено 145 місць із вразливістю у скопійованому коді, які досі не виправлені. Використовуючи інформацію про відомі вразливості, зловмисник може знайти усі місця з подібним вразливим кодом та використати їх у своїх цілях. Як результат, були запропоновані системи виявлення вразливостей, які спеціалізуються саме на скопійованому коді.

Для реалізації такої системи застосовують різні методи. Найпростіші з них, використовуючи лексичні аналізатори, будують послідовність токенів для коду, який містить вразливість та може бути потенційно скопійований. Далі будується послідовність токенів для програми, що перевіряється на вразливості. У послідовності токенів програми, що перевіряється, система намагається знайти підпослідовність, яка є подібною до послідовності токенів вразливості як такої. Вводячи параметр подібності послідовностей, можна виразити те, що під час копіювання програміст змінив код та адаптував його до конкретної ситуації. Шляхом змінювання цього параметра можна, використовуючи ті самі дані про вразливості, знаходити їх у зміненому коді, але при цьому погіршується точність методу, адже з'являється більше помилок першого роду [14]. Такий спосіб є достатньо примітивним, оскільки не враховуються семантичні властивості програми.

У більш прогресивних способах враховано семантичні особливості вразливостей. Наприклад, одна із систем, що ґрунтується на такому підході, будує граф програми, у якому враховуються залежності між даними та потік керування програми [15]. Такі графи генеруються як для коду, який має вразливості, так і для програми, що перевіряється. У цьому випадку пошук вразливостей зводиться до пошуку ізоморфного підграфу. Водночас пошук ізоморфного підграфу є NP-повною задачею, тому дослідники намагаються під час генерування графу вилучати непотрібні ребра або додавати більше семантичної інформації. Такий підхід має недоліки, зумовлені фундаментальним обмеженням швидкості пошуку підграфу, до того ж найчастіше аналізуються фрагменти коду конкретної мови програмування, тобто для кожної мови потрібно розробляти новий аналізатор.

ФОРМАЛЬНІ МЕТОДИ, ЗАСТОСОВАНІ ДЛЯ ВИЗНАЧЕННЯ ВЕБ-АТАК

Завдання убезпечення мережевих систем та систем, що використовують веб-технології, розв'язуються в комплексі, починаючи з рівня мережевого транспорту до рівня прикладного протоколу системи і кінцевих додатків.

На рівні мережевого транспорту завдання убезпечення мереж здебільшого розв'язують за допомогою маршрутизаторів (router) та брандмауерів (firewall). Маршрутизатори організовують мережевий трафік, дотримуючись певних правил, які задаються налаштуваннями. При цьому маршрутизатор може пропускати тільки певну частину мережевого трафіку, оберігаючи мережу, яка знаходиться за ним, від небажаних або підозрілих пакетів. Брандмауер є програмою, що пропускає тільки частину мережевого трафіку, дотримуючись заданих правил. Водночас відкритими залишаються питання, пов'язані з перехопленням ключів безпеки, паролів, веб-сесій тощо. Для таких випадків розробляють формальні методи, що мають на меті верифікацію коректності мережевих протоколів.

Зокрема, в роботі [16] описано формальну модель протоколу безпеки і показано можливість визначати перехоплення ключа безпеки за допомогою формальних методів. Для цього використовується теорія, що включає опис мови, яка складається з множин символів (термів), констант, символів ролей і функціональних символів, множини формул, що задають еквівалентність термів і множини правил виведення. Протоколи задаються як визначення поведінок, кожна з яких описується як транзиційна система, яка, своєю чергою, описує процеси створення, відправлення, отримання та оброблення повідомлень. Для опису транзиційних систем можна використовувати діаграми MSC (Message Sequence Chart), а результати застосування правил представляти у вигляді трас MSC.

У роботі [17] описано набір інструментів, що включають мови специфікацій та інструменти для генерації коду, які дають змогу формально описати і реалізувати необхідний протокол, що відповідає заданій специфікації.

У роботі [18] представлено метод формального визначення атак, пов'язаних зі змінами топології мереж, що ґрунтується на трансформації топологічних схем. У цій роботі використовуються так звані замасковані контрзаходи, наприклад замасковані змінні проти атак, що використовують побічний канал.

Методи, пов'язані з моделюванням мережевих атак з наступним формальним аналізом моделі, описано, зокрема, у роботі [19]. Атаки представлено так званими графами атак та використано техніку model checking для аналізу властивостей цих графів. Крім того, автори застосовують імовірнісний аналіз за допомогою розмітки графів атак величинами ймовірностей і техніки марковських процесів.

Існують також готові інструменти, такі як ProVerif (http://prosecco.gforge.inria.fr/personal/bblanche/proverif/) і Tamarin Prover (https://tamarin-prover.github.io/), призначені для формальної верифікації протоколів безпеки, які використовують, поміж іншим, символьні методи верифікації. У [20] представлено інструмент Legacy Crypto, який застосовується для аналізу вразливостей протоколів, що використовують криптографію.

Оскільки більшість мережевих систем на сьогодні ґрунтується на веб-технологіях, питання безпеки таких систем пов'язані також з уразливостями прикладних протоколів. У роботі [21] описано інструмент для моделювання та аналізу веб-систем, який дає змогу визначити вразливості системи для двох видів атак:

- атака Cross Site Scripting (XSS), під час якої до веб-сторінки, отримуваної з «безпечного» сервера, залучається зловмисний сценарій, що дає змогу виконувати транзакції від імені клієнта та/або викрадати приватні дані клієнта;
- атака Cross Site Request Forgery (CSRF), під час якої користувач отримує доступ до сторінки, яка контролюється зловмисником, що дає змогу останньому виконувати транзакції від імені користувача.

Відмінність цих видів атак полягає в тому, що у випадку CSRF сервер має перебувати під контролем зловмисника, тоді як у випадку XSS сервер і відповідні служби в цілому залишаються безпечними.

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

- інтерпретацією адрес, коли браузер посилає серверу некоректно сформований URL-запит: невірно налаштований сервер у цьому випадку може надати

зловмисникові доступ до інформації, що зберігається на ньому, або виконати будь-які системні команди;

• недостатньою перевіркою наданих даних: клієнт-зловмисник може надсилати серверу інформацію, що порушує нормальне функціонування останнього; тут можуть використовуватися недопустимі символи, невідповідності типів, порушення меж масивів, значення поза допустимими діапазонами тощо.

• прямою інтерпретацією SQL-запитів: часто параметри SQL-запитів передаються безпосередньо в даних веб-форм або в рядках URL, і в цих випадках проста підміна параметрів може надати зловмисникові несанкціонований доступ до даних;

• проблемами з переповненням буфера: недостатній ступінь захисту коду у веб-застосунках 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;

• декомпіляцією JAVA-коду: оскільки байт-код може бути досить ефективно декомпільований, зловмисник може отримати з нього інформацію, що уможливлює несанкціонований доступ до сервера, наприклад паролі (якщо цей код її зберігає);

• перехопленням ключів безпеки: ключі SSL можуть бути розкриті і підмінені зловмисником, що дає змогу останньому виконувати транзакції від імені клієнта; у [20] описано декілька потенційних вразливостей протоколів, що використовують криптозахист;

• імперсоналізацією або перехопленням сесій: оскільки протокол HTTP не передбачає збереження станів під час взаємодії з користувачем, для цього використовуються ідентифікатори сесій, що передаються між клієнтом і сервером як приховані поля; перехоплення такої інформації призводить до отримання інформації про користувача.

Загалом, зважаючи на складність розподілених систем, дослідження у галузі їхньої безпеки зазвичай стосуються окремих аспектів. У роботі [22] представлено огляд, систематизацію та класифікацію формальних методів, використовуваних для розв'язання проблем, пов'язаних з безпекою розподілених систем. При цьому окремо розглянуто кілька важливих розділів: безпека сценаріїв JavaScript, безпека браузерів, безпека веб-додатків і аналіз веб-протоколів.

У [23] запропоновано методологію побудови формальних методів аналізу безпеки систем, що ґрунтуються на веб-технологіях.

Формальні описи атак, пов'язаних із залученням (injection), до яких належать XSS/CSRF, пряма інтерпретація запитів SQL і спроби виконання системних команд на сервері представлено в [24].

У статті [25] описано формальні методи визначення деяких видів веб-атак та їхню реалізацію у вигляді доповнень браузера (тобто тільки на боці клієнта).

У роботі [26] представлено формальні моделі атак, пов'язаних зі спробами зламу шифрованих даних користувача у таких типах веб-застосунків, як менеджер паролів, хмарне сховище даних і системи електронного голосування. Для моделювання власне додатків використано інструмент ProVerif.

Таким чином, у зв'язку із загальною складністю і багатоаспектністю мережевих систем, різноманітністю та еволюцією застосовуваних технологій питання, пов'язані з інтегральною безпекою цих систем (у всіх аспектах функціонування), залишаються актуальними. Досі, навіть в окремих аспектах, завдання убезпечення таких систем розв'язано лише частково.

МАШИННЕ НАВЧАННЯ

Останнім часом ведеться робота щодо використання машинного навчання для аналізу та виявлення вразливостей. Стратегія пошуку вразливостей є стандартною для всіх алгоритмів, що грунтуються на машинному навчанні. Спочатку формують навчальну вибірку, на основі якої створюють модель у вигляді нейтронної мережі, що розпізнає вразливості у коді. Далі готову модель тестують

на контрольній вибірці відомих прикладів вразливостей. Для виявлення вразливостей аналізують такі вибірки, як послідовності системних викликів, притаманні відомим виявленим вразливостям, шаблони доступу до регістрів або пакети, що посилаються мережею. Системи, що ґрунтуються на машинному навчанні та штучному інтелекті, дають змогу знаходити і виводити якісно нові типи вразливостей [10–13].

ІНСЕРЦІЙНИЙ ПІДХІД

У 2017 р. в Інституті кібернетики ім. В.М. Глушкова було розпочато роботи з використання алгебраїчного підходу в процесі виявлення вразливостей у бінарному коді. На основі теорії інсерційного моделювання, розробленої О.А. Летичевським та Д. Гільбертом [27], було здійснено формалізацію семантики асемблера мови інструкцій x86. Для цього було використано математичний апарат алгебри поведінок, що є частиною методу інсерційного моделювання.

Основний принцип інсерційного підходу [28] — це взаємодія агентів у деякому середовищі, при цьому кожен агент знає тільки інформацію про середовище, але може бути сам середовищем для агентів іншого рівня абстракції. Поведінка кожного агента визначається рівняннями алгебри поведінок. Атомарні дії агентів є об'єктами алгебри поведінок та визначаються перед- та післяумовою на атрибутах агентного середовища.

Алгебра поведінок є двосортною алгеброю, яка визначає відношення та операції над поведінками та діями. Кожна поведінка є композицією дій та поведінок. Операція префіксінга a.B визначає, що дія a передує поведінці B. Операція недетермінованого вибору поведінок $u+v$ визначає альтернативу сценарію поведінки. Алгебра має три термінальних константи: успішне закінчення Δ, тупиковий стан 0 та невідому поведінку $\perp$. На множині поведінок також визначено відношення апроксимації $\sqsubseteq$, яке встановлює частковий порядок на множині поведінок з мінімальним елементом $\perp$. Алгебра поведінок також визначає композицію поведінок: послідовну (;) та паралельну (||). Іншими словами, поведінка агента може бути представлена як вираз над поведінкою і рекурсивною дією. Дії визначаються над деякими середовищами атрибутів, у яких всі агенти взаємодіють один з одним. Кожен агент визначається набором атрибутів. Агент змінює свій стан за деяких умов, сформованих значеннями атрибутів. Дії кожного агента визначаються парою a =< P, S >, де P — передумова дії, представленої як формула в деякій базовій логічній мові, S — постумова. Як базову логічну мову ми розглядаємо набір формул логіки першого порядку над поліноміальною арифметикою і деякі спеціалізовані теорії, наприклад переліковні типи, побітові операції, байтові вектори. Загалом семантика дії означає, що агент міг би змінювати свій стан, якщо передумова є істинною, а стан зміниться відповідно до постумови, що також є формулою логіки першого порядку.

Початковий стан агентів може бути представлений початковою формулою. Починаючи з початкової формули, можна застосувати дії, що відповідають вираженню алгебри поведінки. Дія застосовується, якщо її передумова є задовільною і узгоджується з поточним станом. Починаючи з формули початкового стану S0 і з початкової поведінки B0, обираємо дію і переходимо до наступної поведінки. На першому етапі ми перевіряємо виконуваність кон'юнкції $S0 \vee Pa1$, якщо B0 = a1. Bl, і Pa1 є передумовою a1. Наступний стан середовища буде отримано за допомогою предикатного трансформатора, тобто функції PT над поточним станом агента, передумовою і постумовою: PT (Si, Pai, Qai) = Si + 1.

Застосовуючи функцію предикатного трансформатора до різних станів агента, можна отримати послідовності S0, Sl, … формул, які виражають зміну станів агента від початкового стану. Ми отримуємо сценарій поведінки як послідовності дій a1, a2, … Кожен стан агента покриває певну множину конкретних значень агента, а процес генерування таких сценаріїв представляється як символічне моделювання.

Набір інструкцій бінарного коду визначає взаємодію агентської програми з її середовищем: процесор, операційна система і зовнішні пристрої.

Формально він представляє набір таких атрибутів: регістри реєстрів загального та спеціального призначення, визначені як імена регістрів; фізична пам'ять, яка може розглядатися як функція пам'яті; інші специфічні атрибути. Семантика кожної інструкції визначає умови виконання інструкцій, зміну середовища і наступну інструкцію.

Послідовність виконання команди визначає потік керування програмою, який визначається після перекладу бінарного коду на набір виразів алгебри поведінки. Наприклад, потік керування для фрагмента коду

```
0000000000400390  <backtrace_and_maps>:
  400390:  ff cf                  dec     edi
  400392:  0f 8e 3a 01 00 00      jle     4004d2
<backtrace_and_maps+0x142>
  400398:  40 84 f6               test    sil,sil
  40039b:  0f 84 31 01 00 00      je      4004d2
<backtrace_and_maps+0x142>
  4003a1:  55                     push    rbp
  4003a2:  53                     push    rbx
```

відповідає таким специфікаціям алгебри поведінок:

```
B400390  =  dec(37,edi).B400392,
B400392  =  jle(38,4195538).B4004d2 + !jle(38).B400398,
B400398  =  test(39,sil,sil).B40039b,
B40039b  =  je(40,4195538).B4004d2 + !jle(40).B4003a1,
B4003a1  =  push(41,rbp).B4003a2,
B4003a2  =  push(42,rbx).B4003a3,
```

Семантика інструкцій транслюється відповідно у перед- та постумови. Наприклад, інструкція

$cjne\ A\ B\ z$,

транслюється у специфікації поведінки

$$Bx1 = cjne.Bz + !\,cjne.Bx2,\ Bx2 = \ldots$$

та специфікації дій

$$cjne(n, A, B) = !\,(A == B) \rightarrow rip = rip + z + 3;\ FLAG\_C = (B > A);$$

$$!\,cjne(n, A, B) = (A == B) \rightarrow rip = rip + 3;$$

Шаблон вразливості показує поведінку програми, яка призводить до стану вразливості за деяких умов (обмеження над атрибутами середовища). Схема вразливості складається з виразів поведінки та відповідних дій. Загальна форма шаблону вразливості — це вираз поведінки

VulnerabilityPattern = IntruderInput; ProgramBehavior; VulnerabilityPoint.

Існують три основні форми поведінки в узагальненому вигляді. IntruderInput — це поведінка програми, яка представляє інструкції, що являють собою вхідні дії із зовнішнього середовища. Вона може включати командний рядок, читання файлів, отримання інформації з сокета або введення з клавіатури. Така поведінка переважно реалізована як системний виклик основного коду операційної системи. ProgramBehavior — це довільна поведінка програми, яка з'єднує вхідну точку з точкою вразливості і може містити інші деталізовані поведінки. VulnerabilityPoint представляє дії, які містять уразливість.

Далі процес знаходження вразливості полягає в тому, що шаблони вразливостей порівнюються із сценаріями у трансльованому бінарному коді з метою знаходження вразливостей у випадку збігу. Для цього слугує ефективний алгоритм дворівневого пошуку.

Перший рівень пошуку полягає в тому, щоб знайти дерево поведінки, що відповідає алгебраїчнім шаблонам вразливості. Порівнювання поведінки відрізняється від традиційного, що використовується в антивірусних програмах. Традиційні моделі таких програм можуть містити конкретні значення, які перевіряються, тоді як в алгебраїчному підході передбачається розв'язання рівнянь поведінки.

Другий рівень пошуку виконується шляхом символьного моделювання заданої поведінки, отриманого при відповідності поведінки. Під час символьного моделювання ми застосовуємо дії, для яких виявляємо досяжність виразу Env && Prec, де Env є символьним середовищем моделі бінарного коду, а Prec є передумовою відповідної дії. Якщо вона виконується, то ми виконуємо операції постумови в середовищі шаблонів і в середовищі моделі бінарного коду. Якщо ми досягли точки вразливості у шаблоні, то у нас є сценарій, який веде від точки введення. Символічне моделювання вимагає використання специфічних розв'язувачів, які підтримують операції побітових або байтових векторів.

Формальні методи значно підвищують ефективність пошуку вразливостей, водночас з тим ще залишається низка проблем.

Однією з проблем формальних методів та алгебраїчного підходу є те, що проблема досяжності при символічному виконанні є нерозв'язаною в загальному випадку. Може виникнути експоненційний вибух простору станів та інші типові проблеми підходу model checking. Ці проблеми можна розв'язати, використовуючи такі альтернативні символічні методи, як генерація інваріантів, апроксимація або зворотне символічне моделювання. Різні параметри пошуку можуть зменшити простір стану; наприклад, ми можемо надати деякий критерій покриття коду. Однак це скорочення може призвести до того, що ми пропустимо вразливості.

Ще одна задача — узагальнення алгебраїчних моделей. Вона ґрунтується на кількості прикладів та аналізі можливих сценаріїв. Проте немає універсальної техніки і гарантії того, що ця модель охоплює всі можливі поведінки вразливості даного типу.

Іншими проблемами моделювання бінарного коду є такі: моделювання середовища, що містить ядро ОС, бібліотеки, потоки, непряму адресацію та інші специфічні для двійкового коду аспекти. Проте вони є не критичними, а трудомісткими і можуть бути розв'язані в межах цього підходу.

У 2019–2020 рр. очікується значне підвищення активності в галузі кібербезпеки, зокрема впровадження формальних методів у процедуру виявлення вразливостей та шкідливих дій.

Як наслідок, відбувається інтеграція методів перевірки моделей (model checking), зокрема проблеми досяжності та методів автоматизованого доведення теорем для досягнення максимальної ефективності. Основний наголос буде зроблено на розв'язанні проблеми вразливостей «нульового дня» за допомогою методів розпорошування та машинного навчання, які будуть розвиватися з використанням новітніх формальних методів з урахуванням спеціалізацій щодо предметних областей.

СПИСОК ЛІТЕРАТУРИ

1. Cybercrime magazine. URL: https://cybersecurityventures.com/.
2. Nwokedi Idika, Aditya P. Mathur. A survey of malware detection techniques. 2007. Department of Computer Science, Purdue University, West Lafayette, IN 47907. 48 p.
3. Mohan V. Pawar, Anuradha J. Network security and types of attacks in network. ICCC-2015. URL: https://ac.els-cdn.com/S1877050915006353/1-s2.0-S1877050915006353-main.pdf?_tid=21866302-2e58-4f1e-88d0-7d3b9825e011&acdnat=1543497106_74f03131d7fc65a2469e9708a18cc54c.
4. Check Point Software Technologies Ltd. Software Blade Architecture. URL: https://ww.checkpoint.com/downloads/product-related/brochure/Software-Blades-Architecture.pdf.
5. DARPA, "Cyber Grand Challenge." URL: https://www.cybergrandchallenge.com/.

6. Cha S.K., Avgerinos T., Rebert A., Brumley D. Unleashing Mayhem on binary code. *Proc. IEEE Symposium on Security and Privacy.* 2012. P. 380–394.
7. Nguyen-Tuong A., Melski D., Davidson J.W., Co M., Hawkins W., Hiser J. D., Morris D., Nguen D., Rizzi E. Xandra: An autonomous cyber battle system for the cyber grand challenge. *IEEE Security & Privacy*. 2008. Vol. 16, N 2. P. 42–53.
8. Mechaphish Github repository. URL: https://github.com/mechaphish/mecha-docs.
9. American Fuzzy Lop. URL: http://lcamtuf.coredump.cx/afl/.
10. Kolosnjaji B., Zarras A., Webster G., Eckert C. Deep learning for classification of malware system call sequences. *AI 2016: Advances in Artificial Intelligence. Proc. 29th Australasian Joint Conference*, Hobart, TAS, Australia. December 5–8, 2016. P. 137–149.
11. Cova M., Felmetsger V., Banks G. Static detection of vulnerabilities in x86 executables. *22nd Annual Computer Security Applications Conference (ACSAC'06).* 2006. https://doi.org/10.1109/ACSAC.2006.50.
12. Mouzarani M., Sadeghiyan B., Zolfaghari M. Detecting injection vulnerabilities in executable codes with concolic execution. *Proc. 8th IEEE International Conference on Software Engineering and Service Science (ICSESS).* 2017. https://doi.org/10.1109/ICSESS.2017.8342862.
13. Cha S.K., Avgerinos T., Rebert A., Brumley D. Unleashing MAYHEM on binary code. *SP '12 Proc. IEEE Symposium on Security and Privacy.* 2012. https://doi.org/10.1109/SP.2012.31.
14. Li Z., Zou D., Xu S., Jin H., Qi H., Hu J. VulPecker: An automated vulnerability detection system based on code similarity analysis. *Proc. 32nd Annual Conference on Computer Security Applications. ACSAC'16*. 2016. P. 201–213.
15. Flake H. Structural comparison of executable objects. *Proc. IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. 2004. P. 161–173.
16. Lee G. How to formally model features of network security protocols. *International Journal of Security and Its Applications*. 2014. Vol. 8, N 1. P. 423–432. URL: formal.hknu.ac.kr/Publi/ijsia.pdf.
17. Dodds J. Formal methods and the KRACK vulnerability. Galois Inc., 2017. URL: https://galois.com/blog/2017/10/formal-methods-krack-vulnerability/.
18. Coron J.-S. Formal verification of side-channel countermeasures via elementary circuit transformations. *Proc. 16th International Conference, ACNS 2018*. Leuven, Belgium, July 2–4, 2018. P. 65–82. URL: https://eprint.iacr.org/2017/879.pdf/.
19. Jha S., Sheyner O., Wing J. Two formal analyses of attack graphs. *Proc. 15th IEEE Computer Security Foundations Workshop.* 2002. URL: https://ieeexplore.ieee.org/document/1021806.
20. Bhargavan K. et al. Formal methods for analyzing crypto protocols: using legacy crypto: from attacks to proofs. URL: https://cyber.biu.ac.il/wp-content/uploads/2018/02/BIU-Bhargavan-Part1-Slides.pdf.
21. Ferman V., Hutter D., Monroy R. A model checker for the verification of browser based protocols. *Comp. y Sist.* 2017. Vol. 21, N 1. URL: http://www.scielo.org.mx/pdf/cys/v21n1/1405-5546-cys-21-01-00101.pdf.
22. Bugliesi M., Calzavara S., Focardi R. Formal methods for web security. Università Ca' Foscari Venezia. URL: https://www.researchgate.net/publication/308004472_Formal_methods_for_Web_security.
23. Tobarra L., Cazorla D., Cuartero F., Díaz G. 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.
24. Ray D., Ligatti J.. Defining injection attacks. Technical Report #CSE-TR-081114, University of South Florida, Department of Computer Science and Engineering. URL: http://www.cse.usf.edu/~ligatti/papers/broniesTR.pdf.
25. Calzavara S. Formal methods for web session security. Università Ca' Foscari Venezia, Dipartimento di Scienze Ambientali, Informatica e Statistica. URL: http://sysma.imtlucca.it/cina/lib/exe/ fetch.php?media=calzavara.pdf.
26. Bansal C., Bhargavan K., Delignat-Lavaud A., Maffeis S. 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/.

27. Gilbert D., Letichevsky A. Model for interaction of agents and environments. In: *Recent Trends in Algebraic Development Technique. WaDT 1999. LNCS.* Bert D., Choppy C. (Eds.). Berlin; Heidelberg: Springer-Verlag, 2000. Vol. 1827. P. 311–328.

28. Letichevsky A., Letychevskyi O., Peschanenko V. Insertion modeling and its applications. *Computer Science Journal of Moldova.* 2016. Vol. 24, Iss. 3. P. 357–370.

Надійшла до редакції 20.05.2019

А.А. Летичевский, В.С. Песчаненко, Я.В. Гринюк, В.Ю. Радченко, В.М. Яковлев

ОБЗОР СОВРЕМЕННЫХ МЕТОДОВ ЗАЩИЩЕННОСТИ И БЕЗОПАСНОСТИ ПРОГРАММНЫХ СИСТЕМ

Аннотация. Показано, что в настоящее время безопасность и защищенность программных ресурсов является одной из наиболее актуальных проблем в ИТ-отрасли, поскольку действия злоумышленников становятся все более изощренными, а убытки от кибератак растут. Традиционные методы борьбы с кибератаками теряют свою эффективность, поэтому разработка новых методов и средств защиты программных ресурсов является насущной потребностью. Особенно интересными и перспективными являются разработки, базирующиеся на формальных методах с использованием современных алгебраических теорий.

Ключевые слова: алгебраическое моделирование, алгебра поведений, кибербезопасность, инсерционное программирование, формальные методы, символьные методы, символьное моделирование, поиск уязвимостей.

O.A. Letychevskii, V.S. Peschanenko, Y.V. Hryniuk, V.Yu. Radchenko, V.M. Yakovlev

OVERVIEW OF THE MODERN METHODS OF PROTECTION AND SECURITY OF SOFTWARE SYSTEMS

Abstract. Security and protection of software resources are one of the most important problems in the IT industry since attackers' actions become increasingly sophisticated and losses caused by cyberattacks are growing. Traditional methods of cyberattack prevention become inefficient; therefore, development of new methods and tools to secure software resources becomes of essential need. The studies that are based on formal methods with the use of modern algebraic theories are especially interesting and promising.

Keywords: algebraic modeling, behavior algebra, cybersecurity, formal methods, insertion programming, symbolic methods, symbolic modeling, vulnerability detection.

Летичевський Олександр Олександрович,
доктор фіз.-мат. наук, провідний науковий співробітник Інституту кібернетики ім. В.М. Глушкова НАН України, Київ, e-mail: oleksandr.letychevskyi@garuda.ai.

Песчаненко Володимир Сергійович,
доктор фіз.-мат. наук, професор кафедри Херсонського державного університету, e-mail: volodymyr.peschanenko@garuda.ai.

Гринюк Ярослав Васильович,
аспірант Інституту кібернетики ім. В.М. Глушкова НАН України, Київ, e-mail: yaroslav.hryniuk@garuda.ai.

Радченко Віктор Юрійович,
технічний керівник компанії Garuda AI, Київ, e-mail: viktor.radchenko@garuda.ai.

Яковлев Віктор Михайлович,
провідний математик Інституту кібернетики ім. В.М. Глушкова НАН України, Київ, e-mail: victor.yakovlev@garuda.ai, victor.yakovlev@ukr.net.