Модель інформаційної безпеки функціонування програмного забезпечення

Сергій Лєнков[1], Володимир Джулій[2], Олександр Яворський[3], Костянтин Зацепін[4]

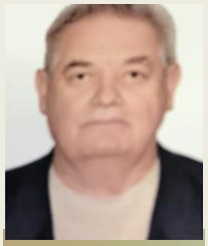
[1] Військовий інститут Київського національного університету імені Тараса Шевченка
Юлії Здановської, 81, Київ, Україна, 03189
[1] lenkov_s@ukr.net, https://orcid.org/0000-0001-7689-239X,
[2,3,4] Хмельницький національний університет
вул. Інститутська, 11, Хмельницький, Україна, 29000
[2,3,4] dzhuliivm@khmnu.edu.ua, https://orcid.org/0000-0003-1878-4301

Received 29.08.2024, accepted 24.10.2024
https://doi.org/10.32347/st.2024.2.1202

Анотація. В роботі виконано систематизацію моделей надійного та безпечного функціонування програмного забезпечення. В результаті проведеного дослідження виділено три типи моделей: аналітичні; статистичні; емпіричні [4, 9].

Розглянуто ряд найчастіше застосовуваних моделей, виділено їх недоліки та переваги з погляду розв'язуваної задачі опису безпечного функціонування програмного продукту, та розпізнавання шкідливого програмного забезпечення. За результатами проведених досліджень розглянуті моделі мають переваги у плані простоти їх практичної реалізації, проте водночас виділено наступні недоліки: деякі з розглянутих моделей при реалізації вимагають великого об’єму обчислювальних ресурсів – для аналізу безпеки та накопичення архівних даних; використання статистичними та імовірнісними моделями припущень про те, що інтенсивність атак/відмов чи кількість помилок у програмному забезпеченню мають заздалегідь відомий розподіл (біноміальний, стандартний або пуасонівський), що не завжди вірно для реальних процесів і систем; немає поділу на відмови програмного забезпечення і вихід з ладу внаслідок кібератак, не враховуються також вразливості нульового дня; не аналізуються звернення досліджуваного програмного забезпечення до пам'яті, що могло б дати важливу інформацію про його легітимність чи наявність шкідливих функцій; жодна з розглянутих моделей не забезпечує комплексного представлення про процес функціонування програмного забезпечення, у тому числі, аналіз зі сторони інформаційної безпеки відсутній [1, 8, 16].

Завдання розпізнавання шкідливого програмного забезпечення з кожним роком стає дедалі актуальнішим і складнішим у зв'язку з цифровізацією галузей діяльності людини та використанням програмного забезпечення для виконання бізнес-логіки та технічних процесів у складних системах. Внаслідок цього, чим більший обсяг програмного забезпечення в системі, тим потенційно більше в ньому помилок, при цьому через підключення сучасних систем до мережі Інтернет, програмне



Сергій Лєнков
д.т.н. професор кафедри, головний науковий співробітник

Володимир Джулій
Доцент кафедри кібербезпеки

Олександр Яворський
Магістр Хмельницького національного університету

Костянтин Зацепін
Магістр Хмельницького національного університету

забезпечення часто поширюється по мережі, що дозволяє зловмисникам створити нові вектори кібератак на системи [2, 3, 5, 7].

Запропонована модель усуває наведені недоліки за рахунок того, що вона враховує характерні особливості прояву шкідливого програмного забезпечення на пристроях, а саме вплив шкідливого програмного забезпечення на обчислювальні ресурси системи та роботу з оперативною пам'яттю. Це дозволяє розробленій моделі враховувати як надійність функціонування програмного забезпечення, так і безпеку [9, 11, 19].

У термінах моделі сформульовані критерії безпечного функціонування програмного забезпечення, зроблено висновок про те, що для найбільш ефективної реалізації такої моделі на практиці має бути використаний гіпервізор [14, 17, 20].

Ключові слова: модель, інформаційна безпека, вразливості, атаки, конфіденційні дані, шкідливе програмне забезпечення, безпечне функціонування.

ВСТУП

Число кібератак на найважливіші галузі інфраструктури у всьому світі неухильно зростає з кожним роком. Так, за даними компанії Trellix, вже в першому кварталі 2023 року в Україні було зафіксовано на 22,5% більше кібератак, ніж у останньому кварталі 2022 року. У другому кварталі кількість кібератак зросла ще на 9%, порівняно з першим кварталом 2023 року [5, 7].

Одним із найпоширеніших методів реалізації кібератак є використання шкідливого програмного забезпечення. Так, на початку 2023 року кількість кібератак, реалізованих цим методом, незначно поступилася лише кібератакам методами соціальної інженерії. Частка кібератак з використанням шкідливого програмного забезпечення, за даними Trellix, склала 71% для фізичних осіб та 60% для юридичних осіб [5, 7, 22].

У зв'язку з цим, актуальність завдання розпізнавання шкідливого програмного забезпечення щорічно зростає. Слід також враховувати, що зловмисники при реалізації кібератак комбінують різні типи шкідливого програмного забезпечення – використовують багатофункціональні трояни або завантажують на скомпрометовані об'єкти інфраструктури безліч різних за функціоналом шкідливих програм. Це значно ускладнює процес аналізу безпеки програмного забезпечення. Особливої гостроти проблема набуває останнім часом у зв'язку з об'єктивною необхідністю організації віддаленої роботи та доступу до інформаційних ресурсів компаній, систем управління та моніторингу технологічних процесів – що потребує вирішення завдання захисту «територіально розподіленого периметра». Зростає небезпека атак, що використовують прийоми як явно вираженої, так і прихованої соціальної інженерії, коли для впровадження шкідливих програм застосовуються довірені джерела [4, 6, 7, 20].

Одним із каналів доставки шкідливих програм можуть виступати комплекти та компоненти легітимного програмного забезпечення. У разі відкритого коду користувачам надається можливість ознайомитися з вихідними текстами, та за необхідності виконати оцінку та аналіз як ефективності продукту, так і його безпеки. У разі програмного забезпечення із закритим кодом вихідні тексти або не надаються взагалі, або надаються частково або повністю державним сертифікаційним органам для винесення рішення про допуск до використання таких продуктів у певних мережах та системах. Більшість ліцензійних угод на використання програмного забезпечення містять повідомлення користувача про обмеженість гарантії та відмову від відповідальності при використанні продукту. Таким чином, програмний продукт без вихідного коду є «чорною» скринькою для користувача, і навіть у разі авторства відомого та сумлінного розробника та постачальника може містити: недокументовані можливості; шкідливі компоненти, впроваджені засоби розробки внаслідок атак на сховища; шкідливе програмне забезпечення і «чорні ходи», впроваджені на етапі розробки, компіляції програмного коду або поширення; помилки та недоробки, не

усунені у процесі тестування; шкідливе програмне забезпечення та «чорні ходи», навмисно впроваджені третіми особами у процесі доставки [9, 10, 15].

Таким чином, завдання оцінки безпеки програмного забезпечення не вирішується довірою до програмного продукту, який розроблений відомою компанією, сертифікований та отриманий із джерел, які видаються довіреними.

АНАЛІЗ ОСТАННІХ ДОСЛІДЖЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ

Виявити потенційні загрози можна на підставі аналізу вихідних текстів та динамічного аналізу поведінки у контрольованому середовищі. Однак в умовах відсутності вихідних текстів потрібне використання ще більш технічно складних процедур, у тому числі декомпіляції, реверсі інжинірингу і т.д. На практиці фахівцю із захисту інформації потрібно швидко і ефективно визначити можливість використання програмного забезпечення в інфраструктурі, що захищається, без порушення інформаційної безпеки. При цьому час та обчислювальні ресурси обмежені, а повна автоматизація процесу аналізу безпеки неможлива у зв'язку з високою складністю предметної галузі [6, 17, 18].

На даний час не існує універсального та оптимального вирішення задачі виявлення шкідливого програмного забезпечення. Використання всього спектра наявних засобів дозволяє максимально наблизитися до вирішення завдання виявлення шкідливого коду, визначення його параметрів та поведінки, і навіть ідентифікації методів, а можливо навіть особистості зловмисника, проте вимагає докладання значних зусиль, тимчасових витрат і кваліфікації [5, 7].

Для підвищення ефективності вирішення цього завдання в умовах обмежених часових та обчислювальних ресурсів, а також відсутності відкритих вихідних текстів програмного забезпечення, пропонується підхід, що полягає в автоматизації оцінки інформаційної безпеки програмного забезпечення без вихідних текстів та базується на моделі безпечного функціонування ПЗ та методики оцінки інформаційної безпеки [14, 16].

Безпека ПЗ означає, що в результаті його включення в систему не відбувається порушення стану безпеки, тобто повною мірою зберігаються функціональні характеристики системи, не змінюються регламентовані взаємодії суб'єктів та об'єктів системи. Безпека також означає відсутність дефектів недокументованих можливостей, які можуть спричинити порушення безпеки як власне аналізованого ПЗ, так і системи, в яку воно встановлюється, у тому числі при взаємодії з іншими компонентами системи. Безпека ПЗ також передбачає відсутність у його складі шкідливого коду [4, 10, 20].

Характеристики шкідливого програмного забезпечення наведені в таблиці 1.

Наведена характеристика порівняно умовна, оскільки шкідливе програмне забезпечення, що використовується для здійснення кібератак може мати комбінований функціонал. Найбільш небезпечними є засоби, націлені на приховану присутність у системі з метою викрадення інформації та перехоплення управління [5, 7, 8].

Регламентовані стандартом заходи та процедури також можуть бути використані для побудови системи оцінки безпеки програмного забезпечення, що пропонується сторонніми розробниками. Однак, безпосереднє застосування методик оцінки, побудованих на аналізі вихідного коду, не може бути ефективно здійснено у разі його відсутності. Більша частина програмного забезпечення для систем Windows, що пропонується користувачам, не постачається з вихідними текстами [2, 9, 13].

Таблиця 1. Характеристики шкідливого програмного забезпечення

Table 1. Characteristics of malware

Класи шкідливого програмного забезпечення, середовища оперування, область дії та поширення		Шпигунське	Рекламне	Куки файли	"Чорний хід"	Троянець	Сніфери	Спам	Мережеві боти	Логічні бомби	Черв'яки	Віруси
Технології створення ШПЗ	Патерни	+	+	+	+	+	+	+	+	+	+	+
	Обфузування	+	+	+	+	+	+	+	+	+	+	+
	Поліморфізм	+	+	+	+	+	+	+	+	+	+	+
	Спеціальні інструменти	+	+	+	+	+	+	+	+	+	+	+
Середовище виконання	Мережа	+	+	+	+	-	+	+	+	+	+	-
	Віддалене виконання	+	+	+	+	+	+	+	+	-	-	-
	Робоча станція	-	-	-	-	-	-	-	-	+	+	+
Засоби поширення, середовище доставки	Мережа	+	+	+	+	+	+	+	+	+	+	+
	Змінні диски	+	+	+	+	+	+	+	+	+	+	+
	Завантажувальні файли	+	+	+	+	+	+	+	+	+	+	+
Руйнівний вплив	Порушення конфіденційності	+	-	+	-	+	+	-	-	-	-	-
	Відволікання користувачів	-	+	-	-	-	-	+	-	-	-	-
	Відмова в обслуговуванні	-	-	-	+	-	-	+	+	+	+	+

На етапі впровадження програмного забезпечення, завдання оцінки безпеки стоїть дуже гостро, і має вирішуватися експлуатаційним персоналом достатньо ефективно. В даний час немає єдиного методичного документа, який дозволяв би при його застосуванні персоналом без спеціальних знань приймати рішення про можливість застосування ПЗ у корпоративних мережах. Натомість існує великий інструментарій, безліч методів та підходів для дослідження поведінки ПЗ, шкідливий характер якого відомий, і завданням дослідника є визначення класу шкідливого ПЗ, характерних ознак та поведінки для подальшого використання в системах захисту [8, 20].

Об'єктивні причини появи вразливостей у ПЗ полягають у надзвичайно високій структурній складності програмного коду, динамічності розвитку версій, що випускаються розробником, та легкості самодифікації програм при віддаленому оновленні. До цього можна додати проблему достовірної ідентифікації навмисно створених програмних закладок, недосконаліть нормативно-методичної та відставання інструментальної бази сертифікаційних випробувань [6, 16, 20].

Вразливі сигнатури у вихідних текстах ПЗ шукаються розробниками лише на етапах внутрішнього тестування, основною метою якого є виявлення власних помилок у розробленому програмному забезпеченню. При цьому про оцінку безпеки ПЗ найчастіше не замислюються [6, 12, 22].

Статичний аналіз вихідних текстів здійснюється без реального виконання досліджуваних програм. Залежно від інструмента, що використовується, глибина аналізу може змінюватись від визначення поведінки окремих операторів до дослідження, що включає весь наявний вихідний текст.

Динамічний аналіз вихідних текстів виконується за допомогою запуску програм на реальному чи віртуальному процесорі. Утиліти динамічного аналізу можуть вимагати завантаження спеціальних бібліотек, перекомпіляцію програмного коду. Для більшої ефективності динамічного аналізу потрібно подання тестованій програмі достатньої кількості вхідних даних, щоб отримати повніше покриття коду.

Статичний та динамічний аналізи дозволяють виконати наступні види контролю:

1. Контроль повноти та відсутності надмірності вихідних текстів. Він здійснюється експертом за допомогою програмних засобів, які виявляють можливо надлишкові функціональні об'єкти, що не використовуються.

2. Контроль відповідності вихідних текстів його об'єктному коду, полягає в компіляції вихідних текстів програм і порівнянні отриманих об'єктних (завантажувальних) кодів з аналогічними, наданими з дистрибутивом ПЗ, що перевіряється. В якості програмних засобів для здійснення цієї перевірки використовуються компілятор та програма підрахунку контрольних сум та порівняння файлів.

3. Контролює зв'язки функціональних об'єктів з управління. Ця перевірка полягає у побудові графа взаємодії функціональних об'єктів з управління (граф виклику функцій). Перевірка може бути виконана експертом без залучення додаткових програмних засобів, проте обсяг вихідних текстів та кількість функціональних об'єктів у сучасному програмному забезпечення досить велика.

4. Контроль зв'язків функціональних об'єктів за інформацією побудові графа взаємодії функціональних об'єктів за інформацією (граф передачі змінних між функціями та використання глобальних змінних усередині функцій).

5. Контроль інформаційних об'єктів передбачає побудову переліку інформаційних об'єктів (усі змінні, крім локальних, що не передаються інші функціональні об'єкти).

6. Контроль наявності заданих конструкцій у вихідних текстах полягає у пошуку певних конструкцій у вихідних текстах ПЗ. Пошук здійснюється за базою вразливостей, що містить сигнатури чи відмітні ознаки вразливості.

При контролі виконання функціональних об'єктів відбувається зіставлення фактичних маршрутів їх виконання та маршрутів, побудованих у процесі проведення статичного аналізу [5, 7, 15].

Лексичний аналіз передбачає пошук розпізнавання та класифікацію різних лексем об'єкта дослідження (програмного забезпечення), представленого у виконуваних кодах. У цьому лексемами є сигнатури. В даному випадку здійснюється пошук сигнатур наступних класів: сигнатури вірусів; сигнатури елементів; сигнатури (лексеми) «підозрілих функцій»; сигнатури штатних процедур використання системних ресурсів та зовнішніх пристроїв. Пошук лексем (сигнатур) реалізується за допомогою спеціальних інструментів – сканерів [10, 14, 16, 22].

Синтаксичний верифікаційний аналіз передбачає пошук, розпізнавання та класифікацію синтаксичних структур, а також побудова структурно-алгоритмічної моделі самого продукту [11]. Семантичний аналіз передбачає дослідження функцій (процедур) продукту аспекті операційної середовища інформаційної системи. На відміну від попередніх видів аналізу, заснованих на статичному дослідженні, семантичний аналіз націлений на вивчення динаміки продукту - його взаємодії з довкіллям. Процес дослідження здійснюється у віртуальному операційному середовищі з повним контролем його дій та відстеженням алгоритму його роботи з структурно-алгоритмічної моделі.

Семантичний аналіз є найбільш ефективним видом аналізу, але при цьому найтрудомісткішим. Узагальнена класифікація методів аналізу програмного забезпечення представлена на рис. 1.

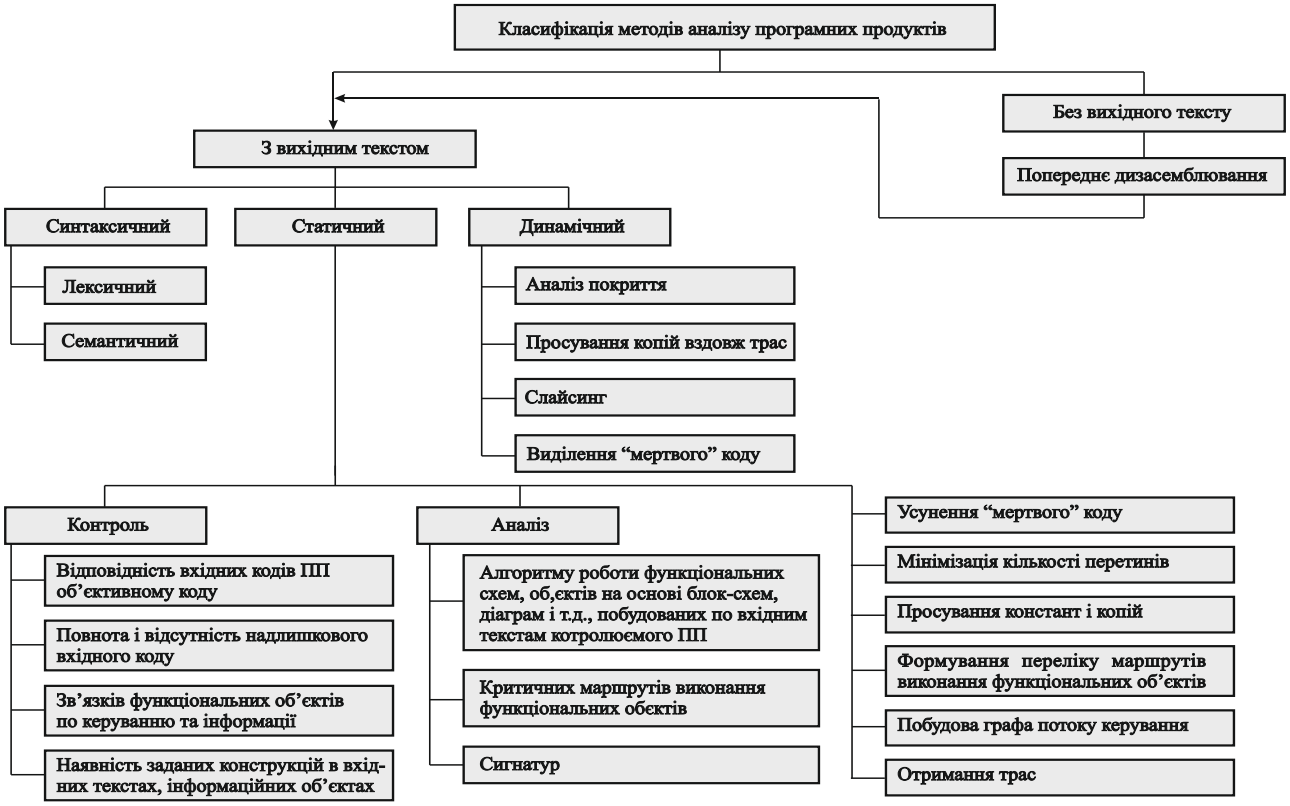


Рис. 1. Узагальнена класифікація методів аналізу програмного забезпечення

Fig. 1. Generalized classification of software analysis methods

Зі зростанням складності ПЗ динамічний аналіз стає нерозв'язним завданням і перетворюється на формальну процедуру. Багато програмних комплексів, які використовуються експертами у повсякденній роботі, мають списки потенційно небезпечних конструкцій, що робить неефективними методи сигнатурного аналізу. Не згадується сигнатурний аналіз у вимогах до випробувань ПЗ нижче другого рівня контролю (тобто, програм, що обробляють секретну та конфіденційну інформацію) [13]. Немає механізмів виявлення помилок кодування, пов'язаних із переповненням буфера, викликом функцій із чужого адресного простору, відсутнє очищення пам'яті тощо. Також відсутні вимоги щодо побудови переліку маршрутів під час виконання функціональних гілок програми, немає механізмів визначення повноти покриття коду та його достатності при динамічному аналізі.

Дослідження вітчизняних та зарубіжних джерел дозволило виділити такі ключові методи, способи та підходи до оцінки інформаційної безпеки ПЗ без вихідних текстів:

1. Перевагою статичного аналізу є повне покриття коду, на відміну методів і способів динамічного аналізу. Виконуючи бінарний аналіз програмного забезпечення, що розробляється, є штатним для середовища функціонування, можна виконати перевірку сторонніх бібліотек, які були використані при розробці. Також можна виконати контрольну перевірку фінальної версії програмного забезпечення шляхом порівняння результатів аналізу вихідного коду та виконуваного коду. Також статичний аналіз може бути успішно використаний при дослідженні стороннього ПЗ, що отримується від постачальників або завантажується з Інтернету.

2. Динамічний аналіз із використанням «пісочниці». Такий вид аналізу являє собою підхід до виявлення підозрілої поведінки у віртуальному ізольованому середовищі. Важливо, що налаштування віртуальних середовищ може бути виконане відповідно до одного з двох підходів.

Оптимальний тут підхід, при якому запуск зразка програмного забезпечення виконується в такому віртуальному середовищі, яке повністю відповідає цільовій системі, якій призначений даний зразок. Однак у такому разі «пісочниця» повинна «на льоту» визначати цільове середовище вузла мережі та запускати потрібну віртуальну машину.

3. Динамічний аналіз із використанням відладчика. Відладники реалізують покрокове виконання програм та встановлення в них точок зупинки. Це може бути реалізовано лише на рівні центрального процесора, мікроконтролера, операційної системи. Ключові функції відладчика – запуск програми, встановлення точок її зупинки, відображення даних, що використовуються програмою, та внесення змін до програми, щоб перевірити, чи можливо таким чином усунути виявлені помилки.

4. Аналіз на імітаційному стенді із використанням форензики. Інтерактивне тестування безпеки додатків є методикою аналізу безпеки ПЗ, переважно спрямовану на дослідження поведінки веб-додатків під час їх роботи [14]. Рішення з інтерактивного тестування зазвичай працюють за рахунок розгортання спеціальних агентів у додатку, що працює. Ці агенти безперервно аналізують те, як взаємодіють додатки (зазвичай ініціювати автоматизованими тестами), щоб виявити вразливості. Деякі рішення щодо інтерактивного тестування безпеки можуть не тільки активно відстежувати вразливості (наприклад, SQL-ін'єкції), але й перевіряти їх, демонструючи їхню придатність для використання зловмисниками. Для аналізу безпеки в такому підході часто використовуються методи форензики, тобто всі методи, пов'язані з розслідуванням вже скоєних комп'ютерних інцидентів.

Таким чином, можна відзначити, що динамічний аналіз більш популярний, ніж статичний, при оцінці безпеки ПЗ без вихідних текстів, однак найкращий результат досягається при їх сукупному використанні, оскільки саме статичний аналіз з використанням дизассемблера дозволяє отримати повне покриття коду та усунути ключовий недолік, властивий методів динамічного аналізу [4, 10, 15].

Слід зазначити, що техніка динамічного аналізу є найбільш популярною, при цьому особливо ефективною вона є в поєднанні з механізмами віртуалізації, оскільки це дає можливість дослідити поведінку зразка ПЗ у середовищі, наближеному за своїми характеристиками до цільової.

Особливої ефективності виявлення шкідливого програмного забезпечення слід очікувати при поєднанні статичного і динамічного аналізу, оскільки статичний аналіз забезпечує максимальне покриття коду. Однак в умовах відсутності вихідних текстів ПЗ це зробити важче, ніж у разі їх наявності, оскільки потрібно використовувати дизасемблер, що дозволяє приблизно відновити високорівневий код програми.

МОДЕЛЮВАННЯ ПРОЦЕСУ ФУНКЦІОНУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Завдання забезпечення безпечного функціонування ПЗ включає: забезпечення інформаційної безпеки для інформаційного середовища під час функціонування ПЗ; забезпечення якісної та надійної реалізації ПЗ своїх функцій.

Програмне забезпечення за функціональним призначенням можна поділити на системне, прикладне та сервісне. Програмне забезпечення є компонентом інформаційної системи (операційної системи, яка своєю чергою може бути частиною програмно-апаратного комплексу, розподіленої обчислювальної мережі).

Оцінка надійності ПЗ пов'язана з можливістю впровадження тексту програмних засобів на етапі розробки та/або модифікації програмних закладок, використовуючи які, зловмисник реалізує створені вразливості. У зв'язку з цим, необхідність сертифікації та перевірки ПЗ на відповідність вимогам чинних нормативних документів протягом усього

його життєвого циклу, не викликає сумнівів [4, 5, 10].

Для оцінки надійності програмних продуктів, що є складовою інформаційних систем, окремий інтерес представляють рандомізовані моделі та методи теорії надійності. Розглянемо основні види моделей оцінки надійності програмних продуктів із зазначенням їх переваг та недоліків.

Модель Джелінскі – Моранди заснована на припущеннях, що час до наступної відмови ПЗ (у разі збою або кібератаки) розподілено експоненційно, а інтенсивність відмов ПЗ пропорційна кількості помилок, що залишилися в програмі.

Перевагою моделі є простота розрахунків, а основним недоліком те, що при неточному визначенні величини початкової кількості помилок інтенсивність відмов програми може стати негативною.

Модель Шумана, передбачається, що дослідження безпеки та надійності ПЗ проводиться в кілька етапів, кожен з яких являє собою виконання ПЗ набору тестових даних. Виявлені протягом етапу тестування помилки реєструються, але не виправляються. Після завершення етапу виправляються всі виявлені помилки, коригуються тестові набори та проводиться новий етап тестування.

Основною перевагою моделі є визначення всіх невідомих параметрів, необхідних для розрахунків, і відсутність необхідності звернення до інших моделей. Недоліком є припущення моделі про те, що при коригуванні програмного забезпечення не вносяться нові помилки. Також для розрахунків потрібна велика кількість зареєстрованих даних.

В основі моделі Модель Шика – Волвертона лежить припущення, за яким частота помилок пропорційна як кількості помилок в ПЗ, а й часу тестування, тобто, ймовірність виявлення помилок з часом зростає.

Модель дозволяє досить точно розраховувати надійність і проста в застосуванні.

Недолік моделі: для розподілів, відмінних від експоненціального, коли інтенсивність помилки змінюється з часом, необхідно використовувати умовні розподіли. При цьому умовна ймовірність для певного інтервалу тестування повинна відповідати проміжку часу від початку тестування до початку розгляду інтервалу тестування. Інакше цей процес відновлення за умови усунення виявлених помилок призведе до завищення ймовірностей безпомилкового функціонування на всіх ділянках тестування.

Модель Коркорена передбачає наявність у ПЗ багатьох джерел програмних відмов, пов'язаних з різними типами помилок та вразливостей, та різну ймовірність їх появи.

Перевагою моделі є те, що вона використовує ймовірності відмов, що змінюються, для різних типів помилок і оцінюється ймовірність безвідмовного виконання ПЗ на даний момент часу. Недоліком є виконання оцінки з урахуванням апріорної інформації чи даних попереднього періоду функціонування однотипних зразків ПЗ.

Модель IBM. Під час експлуатації користувачем поточної версії програмного забезпечення розробники, як правило, займаються його супроводом – вносять деякі покращення або виправлення в цю версію, не чекаючи, поки користувач цього вимагатиме. Супровід може включати також додавання нових функцій. З деякого моменту, коли розробник вважає своє завдання виконаним, починається пасивний супровід, коли виправлення вносяться вже за запитом користувача. При супроводі в кожну нову версію програмного забезпечення вноситься значна кількість нових помилок, разом із доопрацюваннями, змінами та виправленнями, що потребує виправлень також і в наступній версії.

Зокрема, в компанії IBM спробували передбачити кількість подібних виправлень від версії до версії, ґрунтуючись на велику кількість експериментальних даних, зібраних у ході супроводу операційної системи.

Модель Шнайдера пов'язує число помилок у програмі з витратами, виміряними в «людино-місяцях», числом

підпрограм і загальним числом *тисяч* операторів у програмі.

Перевагою моделі є низька обчислювальна складність і простота реалізації, до недоліків можно віднести необґрунтованість емпірично обраного числа виправлень і те що, що вразливості нульового дня не враховуються, а функції ПЗ не досліджуються детально.

МОДЕЛЬ БЕЗПЕЧНОГО ФУНКЦІОНУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Запропонована модель безпечного функціонування ПЗ має усувати недоліки, властиві розглянутим моделям. За результатами проведених досліджень моделей, можна зробити наступні висновки про їх недоліки: деякі моделі при реалізації вимагають значного обсягу обчислювальних ресурсів, це пов'язано як із необхідністю накопичення та обробки великої кількості даних, так і з обчислювальними ресурсами компонентів системи, які здійснюють аналіз безпеки; ймовірнісні моделі, як правило, ґрунтуються на припущеннях про те, що інтенсивність відмов/атак чи кількість помилок у ПЗ заздалегідь відомі, відомий розподіл (стандартний, пуассонівський, біноміальний), що не завжди вірно для реальних процесів і систем; відмови ПЗ та кібератаки, викликані експлуатацією вразливостей у ПЗ, розглядаються як сукупність, внаслідок чого моделі не враховують важливої особливості роботи шкідливого програмного забезпечення – а саме характеру його звернень з пам'яттю; жодна з розглянутих моделей не забезпечує комплексного представлення про процес функціонування ПЗ, у тому числі, погляд з боку інформаційної безпеки відсутній.

Для усунення вищеописаних недоліків пропонується основою моделі покласти опис взаємодії сутностей, задіяних у процесі функціонування ПЗ. При цьому необхідно враховувати специфіку поведінки шкідливого програмного забезпечення на комп'ютері або іншому обчислювальному пристрої.

Дослідження показали, що з основних ознак зараження різного типу шкідливого програмного забезпечення наступні:

1. Уповільнення роботи комп'ютера або вузла мережі. Як правило, це викликано впровадженням шкідливого програмного забезпечення та виконанням своїх функцій, що вимагають обчислювальних ресурсів, внаслідок чого на виконання легітимних обчислювальних задач потрібно більше часу.
2. Виконання нових функцій, не властивих даному комп'ютеру: відправлення даних на сторонні веб-ресурси, запуск процесів, які раніше не виконувались, і т.д.
3. Проблеми з відкриттям файлів, які раніше успішно відкривалися на комп'ютері, а також зміни розширення файлів (зникнення розширень або, навпаки, поява подвійних розширень).
4. Раптове завершення роботи додатків або поява спливаючих вікон при роботі додатків, що потенційно сигналізує про шкідливі процеси, що виконуються фоном.

Для шкідливого програмного забезпечення характерне надмірне використання обчислювальних ресурсів, а також операції з пам'яттю, що значно відрізняються від операцій, що виконуються легітимним ПЗ.

Так, наприклад, для шкідливого програмного забезпечення, що використовує обчислювальні потужності середовища для майнінгу, характерним є такий прояв у системі, як уповільнення роботи обчислювальних компонентів системи, поява обчислювальних процесів, що використовують велику кількість ресурсів, а також відправлення мережевого трафіку на сторонні веб-ресурси.

Таким чином, ключову роль описі роботи програмногозабезпечення грають: функції, що виконуються програмним забезпеченням; зміни у використанні обчислювальних ресурсів; зміни, пов'язані з роботою програмного забезпечення з пам'яттю.

На рис. 2 наведено графічне представлення моделі як набору взаємодіючих сутностей. Опис взаємодії сутностей моделі (рис. 2), відбиває як

надійність функціонування програмного забезпечення, що з виконанням заявлених функцій, а й безпеку функціонування програмного забезпечення.

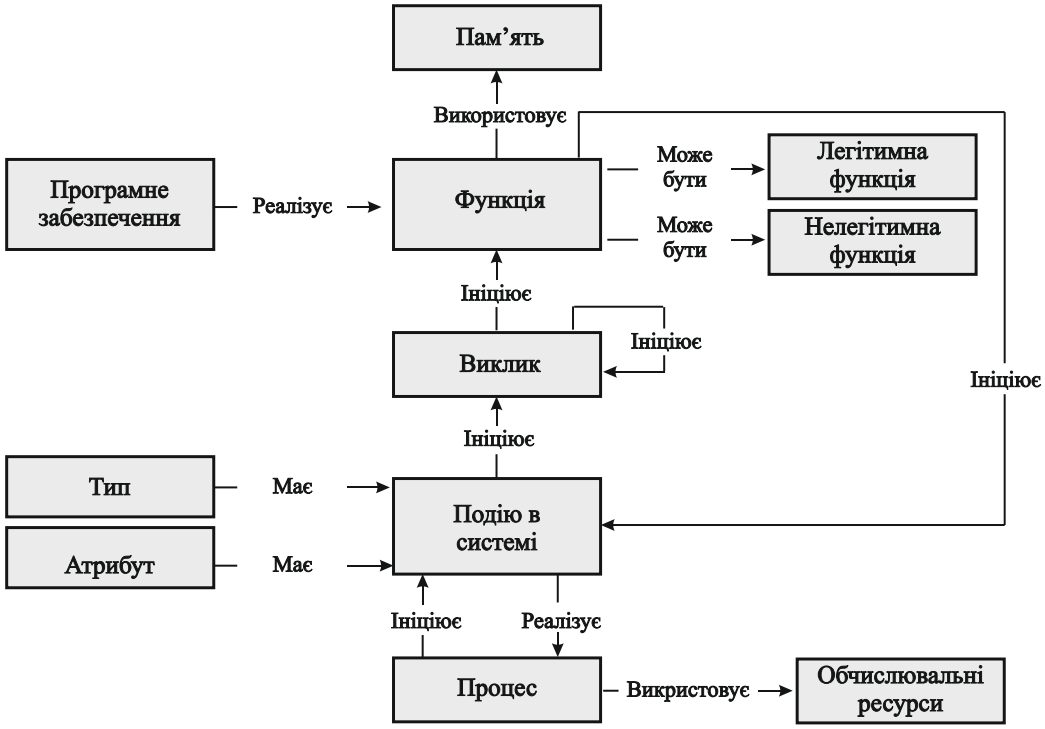


Рис. 2. Графічне представлення моделі безпечного функціонування програмного забезпечення

Fig. 2. Graphical representation of the model of secure software operation

Безпечним буде вважатися ПЗ, встановлення якого не веде до порушення легітимних відношень у системі, але також і те, що не має такого потенціалу внаслідок наявності у своєму складі певного шкідливого функціоналу. Для формалізації опису скористаємося математичним апаратом теорії множин. Опишемо модель безпечного функціонування ПЗ формально, представивши ПЗ ( ) як кортеж:

$$Software =< F, Sign, R, M, Memory > \qquad (1)$$

де:

1. $F = \{ f_1, f_2, ..., f_n \}$ - кінцева множина функцій, які виконує програмне забезпечення, що досліджується. Функції, що входять до множини F, поділяються на деструктивні F_{destr} та легітимні функції F_{leg}: $F = F_{leg} \bigcup F_{destr}$, $F_{leg} \bigcap F_{destr} = \varnothing$. При цьому, в будь якій із підмножин F_{leg} і F_{destr} можуть бути присутніми як документовані функції програмного забезпечення F_{doc}, так і не документовані F_{undoc}, $F_{doc} \bigcap F_{undoc} = \varnothing$.

2. $Sign$ – множина ознак, що характеризують виконання функцій програмного забезпечення. Маємо неоднозначне відображення множини F на множину $Sign: F \to Sign$, тобто кожному елементу множини F відповідає один або кілька елементів множини $Sign$. Використовуючи відповідні індекси, задамо відображення $F_{leg} \to Sign_{leg}$, $F_{destr} \to Sign_{destr}$, $F_{doc} \to Sign_{doc}$, $F_{undoc} \to Sign_{undoc}$.

3. R – обчислювальні ресурси, необхідні програмному забезпеченню для реалізації своїх функцій. Маємо відображення $F \to R$, внаслідок якого кожній f_i призначаються три чисельні значення: $\{ r_{i_{\min}}, r_i, r_{i_{\max}} \}$.

Очевидно, що оцінка необхідних обчислювальних ресурсів може змінюватись, проте в системі для кожного програмного продукту повинен бути виділений певний обсяг обчислювальних ресурсів, щоб система ефективно функціонувала.

4. M – множина методів дослідження, що застосовуються для виявлення шкідливого програмного забезпечення.

5. $Memory$ – множина операцій над пам'яттю, що виконуються при активації тієї чи іншої функції з множини F. Таким чином, маємо відображення $F \to Memory$, за якого кожній функції f_i зіставляється набір операцій над пам'яттю:

$$f_i \to \left\{ memory_{i_1}, memory_{i_2}, ..., memory_{i_k} \right\}.$$

Необхідно відзначити, що застосування методів M тільки в ідеальному випадку дозволить виявити множину ознак $Sign$, характерних для програмного продукту. На практиці слід говорити про визначення за допомогою методів M підмножини $Sign'$, $Sign' \in Sign$. На практиці вирішується завдання оцінки умовної ймовірності $P(A|B)$ події $A = \{Sign_{destr} = \varnothing \; and \; Sign_{undoc} = \varnothing\}$ при умові виконання події B реалізації деякого набору методів M', $M' \in M$ ′, що дозволяє виявити достатнє число ознак $Sign'_{destr}$ та $Sign'_{undoc}$,

$$B = \left\{ M' \to \left\{ Sign'_{destr}, Sign'_{undoc} \right\} \neq \varnothing \right\}.$$

Тоді умовна ймовірність $P(A|B)$ визначиться як (1):

Сформулюємо критерії безпечного функціонування програмного продукту в термінах моделі:

1. Програмний продукт не реалізує деструктивних та шкідливих функцій: $F_{destr} = \varnothing$ та $F_{undoc} = \varnothing$.

2. Об'єм, задіяних для аналізу безпеки програмного продукту (за допомогою множини методів M) обчислювальних ресурсів мінімізовано:

$$M : R \to R_{\min}, R_{\min} = \left\{ r_{1_{\min}}, r_{2_{\min}}, ..., r_{n_{\min}} \right\},$$

таким чином, кожна функція програмного продукту в ідеалі повинна використовувати для реалізації мінімально допустимого об'єму обчислювальних ресурсів;

3. Не повинно бути обчислювальних ресурсів, що витрачаються на будь - яку множину функцій F', що не перетинається з множиною $F : \exists F' \to R$, оскільки присутність шкідливого програмного забезпечення в системі часто призводить до активації нових, нетипових для роботи системи функцій.

4. Множина операцій над пам'яттю $Memory$ повинна мати відображення на множини ознак $Sign$, оскільки, важливою ознакою наявності шкідливого програмного забезпечення в системі є нетипові операції з пам'яттю: $Memory \to Sign \neq \varnothing$.

5. Множина методів дослідження, застосовуваних виявлення шкідливого програмного забезпечення, має використовувати результати відображення $Memory \to Sign$ означає, що для вирішення поставленого завдання доцільно застосовувати ті методи, які враховують операції програмного забезпечення з пам'яттю.

6. Множина використовуваних методів має бути такою, що умовна ймовірність $P(A|B)$ максимізована, тобто $M : P(A|B) \to 1 : (|) \to 1$.

$$P(A|B) = P(A) \cdot P(B) = P\{Sign_{destr} = \varnothing \; and \; Sign_{undoc} = \varnothing\} \cdot \cdot P\left\{ M' \to \left\{ Sign'_{destr}, Sign'_{undoc} \right\} \neq \varnothing \right\} \quad (1)$$

Досягнення максимальної ефективності використовуваних методів M малоймовірно, однак у такому разі буде досягнуто рівність (2):

$$Sign_{destr} = Sign'_{destr}, \; Sign = Sign'_{undoc} \quad (2)$$

Розроблена модель відрізняється від розглянутих моделей оцінки безпеки,

надійності та якості функціонування програмного продукту тим, що вона враховує як особливості функціонування шкідливого програмного забезпечення на компоненті системи, так і необхідність мінімізації обчислювальних ресурсів при аналізі. Особливу увагу слід приділити питанням практичної реалізації розробленої моделі, оскільки вона повинна поєднувати високу ефективність виявлення шкідливого програмного забезпечення та оптимальне використання обчислювальних ресурсів. Програмне забезпечення з потенційно небезпечними можливостями реалізує одну або кілька функцій, наслідки виконання яких можуть безпосередньо загрожувати порушенням безпеки даних, або призводити до виконання коду, що представляє цю загрозу. Види небезпечних можливостей представлені на рис. 3.

Значимість їх систематизації виходить з особливостях роботи з пам'яттю, виділені можливості, наступні: передача управління в область модифікованих даних; самодифікація або зміна коду іншого програмного забезпечення в оперативній пам'яті або на зовнішніх носіях; самодублювання, підміна собою іншого програмного забезпечення або перенесення своїх фрагментів в область оперативної або зовнішньої пам'яті, що не належить програмі; збереження інформації з областей оперативної пам'яті, які не належать програмному забезпеченню; спотворення, блокування чи заміну інформації, що є результатом роботи інших програм; приховування своєї присутності у програмному середовищі.

Виявлення функцій, що здійснюють роботу з пам'яттю таким чином, що виникають потенційно небезпечні можливості, особливо значимо, тому що це основна характеристика поведінки шкідливого програмного забезпечення. Тому при реалізації моделі на практиці слід використовувати механізм, який забезпечує ефективний моніторинг звернень до пам'яті та її використання.

Рис. 3. Програмне забезпечення з потенційно небезпечними можливостями

Fig. 3. Software with potentially dangerous capabilities

Процес у віртуальному операційному середовищі з повним контролем дій програмного забезпечення та відстеження алгоритму його роботи здійснюється більш ефективно. Таким чином реалізація розробленої моделі має базуватися на гіпервізорі, що дозволить віртуалізувати системні ресурси обчислювальних систем.

ВИСНОВКИ

В роботі виконано систематизацію моделей надійного та безпечного функціонування програмного забезпечення. В результаті проведеного дослідження виділено три типи моделей: аналітичні; статистичні; емпіричні.

Розглянуто ряд найчастіше застосовуваних моделей, виділено їх недоліки та переваги з погляду розв'язуваної задачі опису безпечного функціонування програмного продукту, та розпізнавання шкідливого програмного забезпечення. За результатами проведених досліджень розглянуті моделі мають переваги у плані простоти їх практичної реалізації, проте водночас виділено наступні недоліки: деякі з розглянутих моделей при реалізації вимагають великого обсягу обчислювальних ресурсів – для аналізу безпеки та накопичення архівних даних; використання статистичними та імовірнісними моделями припущень про те, що інтенсивність атак/відмов чи кількість помилок у програмному забезпеченню мають заздалегідь відомий розподіл (біноміальний, стандартний або пуасонівський), що не завжди вірно для реальних процесів і систем; немає поділу на відмови програмного забезпечення і вихід з ладу внаслідок кібератак, не враховуються також вразливості нульового дня; не аналізуються звернення досліджуваного програмного забезпечення до пам'яті, що могло б дати важливу інформацію про його легітимність чи наявність шкідливих функцій; жодна з розглянутих моделей не забезпечує комплексного представлення про процес функціонування програмного забезпечення, у тому числі, аналіз з боку інформаційної безпеки відсутній.

Завдання розпізнавання шкідливого програмного забезпечення з кожним роком стає дедалі актуальнішим і складнішим у зв'язку з цифровізацією галузей діяльності людини та використанням програмного забезпечення для виконання бізнес-логіки та технічних процесів у складних системах. Внаслідок цього, чим більший обсяг програмного забезпечення в системі, тим потенційно більше в ньому помилок, при цьому через підключення сучасних систем до мережі Інтернет, програмного забезпечення часто поширюється по мережі, що дозволяє зловмисникам створити нові вектори кібератак на системи.

Запропонована модель безпечного функціонування програмного продукту має усувати недоліки, властиві розглянутим моделям. Запропоновано модель усуває наведені недоліки за рахунок того, що вона враховує характерні особливості прояву шкідливого програмного забезпечення на пристроях, а саме вплив шкідливого програмного забезпечення на обчислювальні ресурси системи та роботу з пам'яттю. Це дозволяє розробленій моделі враховувати як надійність функціонування програмного забезпечення, так і безпеку.

У термінах моделі сформульовані критерії безпечного функціонування програмного забезпечення, зроблено висновок про те, що для найбільш ефективної реалізації такої моделі на практиці має бути використаний гіпервізор. Вибір конкретного гіпервізора залежить від потреб, призначення та специфіки системи.

REFERENCES

1. Doktryna informatsiinoi bezpeky Ukrainy, zatverdzhenoi Ukazom Prezydenta Ukrainy vid 25 liutoho 2017 roku № №47/2017, 15s.
2. Derzhavnyi standart Ukrainy Zakhyst informatsii. Tekhnichnyi zakhyst informatsii. Osnovni polozhennia. DSTU 3396.0-96 [Elektronnyi resurs]. – Rezhym dostupu: http://www.dsszzi.gov.ua/dsszzi/control/uk/publish/article?art_id=38883&cat_id =38836.
3. Zakon Ukrainy «Pro osnovni zasady zabezpechennia kiberbezpeky Ukrainy» zi zminamy. Vidomosti Verkhovnoi Rady (VVR), 2017, № 45, st.403, zi zminamy vid 28.07.2022 roku. Rezhym dostupu:

https://zakon.rada.gov.ua/laws/show/2163
4. **Tsybulnyk S. O., Barandych K. S.** (2022). Tekhnolohii rozroblennia prohramnoho zabezpechennia. Chastyna 1. Zhyttievyi tsykl prohramnoho zabezpechennia : bool. Kyiv, KPI im. Ihoria Sikorskoho, 270.
5. **Lysenko S. M., Shchuka R. V.** (2020). Analiz metodiv vyiavlennia shkidlyvoho prohramnoho zabezpechennia v kompiuternykh systemakh. *Visnyk Khmelnytskoho natsionalnoho universytetu*, 2(283). 101–107.
6. **Lenkov S. V., Dzhulii V. M., Bernaz A. M., Muliar I. V., Pampukha I.V.** (2023). Metod prohnozuvannia vrazlyvostei informatsiinoi bezpeky na osnovi analizu danykh tematychnykh internet-resursiv. *Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka*, Issue78, 123-134.
7. **Lenkov S. V., Dzhulii V. M., Solodieieva L. V**. (2022). Metod protydii poshyrenniu ta vyiavlennia shkidlyvoi informatsii v sotsialnykh merezhakh. *Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka*, Issue 77, 103-117.
8. **Lenkov S. V., Dzhulii V. M., Orlenko V. S., Sieliukov O. V., Atamaniuk A. V.** (2020). Model bezpeky poshyrennia zaboronenoi informatsii v informatsiino-telekomunikatsi-inykh merezhakh. *Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka*, Issue 68, 53-64.
9. **Krepych S. Ya., Spivak I. Ya.** (2020). Yakist prohramnoho zabezpechennia ta testuvannia : bazovyi kurs. Ternopil, FOP Palianytsia V. A. Publ., 478.
10. **Zolotukhina O. A., Nehodenko O. V., Reznyk S. Yu., Razina S. Ya.** (2020). Iakist ta testuvannia informatsiinykh system : Navchalnyi posibnyk dlia samostiinoi roboty studentiv vyshchykh navchalnykh zakladiv, Kyiv, NNIIT DUT, 128.
11. **Vyshnia V. B., Havrysh O. S., Ryzhkov E. V.** (2020). Osnovy informatsiinoi bezpeky: book. Dnipro, Dniprop. derzh. un-t vnutrish. sprav, 128.
12. Cotsialni merezhi – realni zahrozy virtualnoho svitu. [Elektronnyi resurs]. – Rezhym dostupu : http://ogo.ua/ articles/view/011–02–23/26490.htm.
13. **Ostapov S. E., Yevseiev S. P., Korol O. H.** (2016). Tekhnolohii zakhystu informatsii: book. Kharkiv, Vyd-vo KhNEU, 476.
14. **Lienkov S. V., Dzhulii V. M., Bernaz N. M., Bozhuk S. O.** (2017). Analiz isnuiuchykh metodiv ta alhorytmiv vyiavlennia atak v bezdrotovykh merezhakh peredachi danykh. *Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka*. Kiyv, VIKNU, Issue 56, 124-132.
15. **Buriachok V. L. Toliupa S. V., Semko V. V.** Informatsiinyi ta kiberprostory: problemy bezpeky, metody ta zasoby borotby : Book. Kyiv, DUT-KNU, 178.
16. **Rybalchenko L. V., Kosychenko O. O.** (2019). Problemy bezpeky personalnykh danykh v Ukraini. *Rehionalna ekonomika*. Zaporizhzhia. 2019, 57-62.
17. **Dzhulii V. M., Miroshnichenko O.V., Solodieieva L.V.** (2022). Metod klasyfikatsii dodatkiv trafika kompiuternykh merezh na osnovi mashynnoho navchannia v umovakh nevyznachenosti. *Zbirnyk naukovykh prats Viiskovoho instytutu Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka*. Kyiv VIKNU, Issie 74, 73-82.
18. **Lavrov Ye. A., Perkhun L. P., Shendryk V. V.** (2017). Matematychni metody doslidzhennia operatsii: pidruchnyk. Sumy: Sumskyi derzhavnyi universytet, 212.
19. **Honchar S. F.** (2019). Otsiniuvannia ryzykiv kiberbezpeky informatsiinykh system obiektiv krytychnoi infrastruktury: monohrafiia. Kyiv, 175.
20. **Yemchuk L., Zhylinska O., Chornyi A., Dzhuliy V.** (2020). Organizational Network Analysis as a Tool for Leadership Assessment in Software Development Team. *Institute of Electrical and Electronics Engineers (30 September 2020)*; INSPEC Accession Number: 20008165; DOI: 10.1109/ACIT49673.2020.
21. Syhnatura ataky. Wikipedia [Elektronnyi resurs] – Rezhym dostupu do resursu: https://uk.wikipedia.org/wiki/Syhnatura_ataky
22. OPWNAI: Cybercriminals Starting to Use ChatGPT, January 6, 2023 [Elektronnyi resurs] – Rezhym dostupu do resursu: https://research.checkpoint.com/2023/opwnai-cybercriminals-starting-to-usechatgpt.

Information security model of functioning software

Serhii Lienkov, Volodymyr Dzhuliy, Oleksandr Yavorskyi, Kostyantyn Zatsepin

Abstract. The paper systematizes the models of reliable and safe functioning of the software. As a result of the research, three types of models were identified: analytical; statistical; empirical.

A number of the most frequently used models are considered, and their disadvantages and advantages are highlighted from the point of view of solving the problem of describing the safe functioning of a software product and recognizing malicious software. According to the results of the research, the considered models have advantages in terms of the simplicity of their practical implementation, but at the same time, the following disadvantages are highlighted: some of the considered models require a large amount of computing resources when implemented - for security analysis and accumulation of archival data; the use of statistical and probabilistic models of assumptions that the intensity of attacks/failures or the number of errors in software have a pre-known distribution (binomial, standard or Poisson), which is not always true for real processes and systems; there is no division into software failures and failures due to cyber attacks, zero-day vulnerabilities are also not taken into account; memory accesses of the investigated software are not analyzed, which could provide important information about its legitimacy or the presence of malicious functions; none of the considered models provides a comprehensive representation of the process of software functioning, including, there is no analysis from the information security side.

The task of recognizing malicious software is becoming more and more relevant and difficult every year in connection with the digitalization of human activities and the use of software for the execution of business logic and technical processes in complex systems. As a result, the larger the volume of software in the system, the more errors there are potentially, and due to the connection of modern systems to the Internet, the software is often distributed over the network, which allows attackers to create new vectors of cyber attacks on systems.

The proposed model of safe functioning of the software product should eliminate the shortcomings inherent in the considered models. The proposed model eliminates the mentioned shortcomings due to the fact that it takes into account the characteristic features of the manifestation of malicious software on devices, namely the impact of malicious software on the computing resources of the system and working with RAM. This allows the developed model to take into account both the reliability of software operation and security.

In terms of the model, the criteria for the safe functioning of the software are formulated, it is concluded that for the most effective implementation of such a model in practice, a hypervisor should be used.

Keywords: model, information security, vulnerabilities, attacks, confidential data, malware, secure operation.