КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (23), 2024
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2024.23.7184
УДК 004.056

Куперштейн Леонід Михайлович
кандидат технічних наук, доцент, доцент кафедри захисту інформації
Вінницький національний технічний університет, Вінниця, Україна
ORCID 0000-0001-6737-7134
kuperstein.lm@gmail.com

Луцишин Геннадій Леонідович
магістр з кібербезпеки
Вінницький національний технічний університет, Вінниця, Україна
ORCID 0009-0001-5753-5407
abitstudent16@gmail.com

Кренцін Михайло Дмитрович
аспірант кафедри захисту інформації
Вінницький національний технічний університет, Вінниця, Україна
ORCID 0000-0002-1792-9401
mishatron98@gmail.com

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ МОНІТОРИНГУ БЕЗПЕКИ ДАНИХ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. У статті запропоновано інформаційну технологію моніторингу безпеки даних програмного забезпечення з метою підвищення захищеності допоміжного та кінцевого програмного забезпечення в процесі його функціонування за рахунок універсальності архітектури з можливостями узгодження процесів перевірки безпеки даних у клієнт-серверних взаємодіях та інтеграції у програмні технології розробки програмного забезпечення з використанням уніфікованих інтерфейсів задання розширюваних наборів правил перевірки. Проведено аналіз предметної області, в ході якого було встановлено, що задачі перевірки безпеки даних програмного забезпечення входять до вимог відомих стандартів з безпеки даних, а пов'язані вразливості відзначаються як особливо важливі. Також було встановлено, що традиційного моніторингу лише вхідних даних програмного забезпечення недостатньо, тому актуальною є проблема моніторингу безпеки різних категорій даних програмного забезпечення. Можливості моніторингу безпеки та коректності даних програмного забезпечення існуючих засобів мають частковий характер: орієнтація на окремі категорії даних, платформна залежність, вузькі можливості інтеграції з іншими засобами розробки програмного забезпечення, обмеженість застосування, складність чи обмеженість розширення, низький рівень повторного використання готових перевірених рішень та інші. На основі аналізу існуючих засобів моніторингу безпеки даних програмного забезпечення було встановлено недоліки існуючих реалізацій. В якості рішення запропоновано універсальну архітектуру програмного фреймворку. Було проведено аналіз вразливостей різних категорій даних програмного забезпечення, а також досліджено рекомендовані методи імплементації безпеки даних, які було застосовано у запропонованій розробці. У статті розглянуто структуру запропонованої інформаційної технології, наведено універсальну архітектуру програмного фреймворку, продемонстровано перевірку роботи розробленого засобу та наведено оцінку ефективності використання фреймворку моніторингу безпеки даних при розробці програмного забезпечення. Запропонована архітектура та напрямки подальшого удосконалення фреймворку дозволяють суттєво розширювати його функціональні можливості та легко інтегрувати у популярні технології розробки програмного забезпечення. Передбачається, що розроблена інформаційна технологія моніторингу безпеки даних програмного забезпечення набуде широкого використання не лише в комерційній розробці програмного забезпечення, але і в навчальному та науковому застосуванні.

© Л. Куперштейн, Г. Луцишин, М. Кренцін, 2024

Ключові слова: безпека застосунку; моніторинг даних; Cross-site Scripting; SQL-ін'єкції; перевірка введених даних; помилки валідації.

ВСТУП

Сьогодні цифровізація життя призводить до перенесення все більшої кількості даних в електронну форму. Все більше процесів діяльності людини, держави та бізнесу відбувається в електронному вигляді [1]. Розподіленість та слабка зв'язність архітектур сучасного програмного забезпечення, децентралізація команд розробки, повторне використання сторонніх компонентів та різноманіття методів та засобів розгортання та хостингу відкриває широкі можливості конкурентоздатної розробки програмного забезпечення, але також значно підвищує рівень його вразливості та розширює спектр можливих атак. Отже, потреба в безпеці даних невпинно зростає. В такому контексті програмне забезпечення можна розглядати як систему трансформації даних, де її коректним функціонуванням в будь-який момент часу можна вважати стан коректності та безпеки даних системи. До таких даних відносяться вхідні, вихідні та внутрішні дані, які обробляються програмним забезпеченням. Для коректного функціонування програмного забезпечення необхідною є безпека усіх категорій даних, адже будь-яка вразливість хоча б однієї з них може призвести до компрометації інших категорій даних або програмного забезпечення в цілому.

Постановка проблеми. Проблема моніторингу безпеки та коректності даних програмного забезпечення сьогодні являється особливо актуальною, що підтверджується вимогами відомих стандартів з безпеки. У стандарті PCI DSS [2] вимога №6 вимагає розробки захищених систем та застосунків. Підрозділ №6.5.1 вимагає захисту від різних видів ін'єкцій, де перевірка безпеки даних відіграє важливу роль. Вимога № 6.5.5 визначає необхідність коректної обробки помилок для запобігання витоку конфіденційних даних. До переліку вразливостей CWE Top 25 входить як вразливість, спричинена неналежною перевіркою даних [3], так і похідні вразливості. Методичний фреймворк OWASP Security Knowledge Framework [4] визначає більше двадцяти методів реалізації перевірки даних, до числа яких входять перевірка вхідних даних, перевірка даних у серверному програмному забезпечення, некоректна обробка помилок та інші. У методичному фреймворку OWASP Web Security Testing Guide [5] кілька розділів присвячено важливості та методиці перевірки даних програмного забезпечення. Згідно з OWASP Top Ten [6], ін'єкції посідають третє місце серед найбільш поширених вразливостей.

Аналіз останніх досліджень і публікацій. Наступні категорії даних програмного забезпечення розглядаються як об'єкти моніторингу [7], [8]:

- вхідні та вихідні дані кінцевого користувача;
- вхідні та вихідні дані прикладного програмного інтерфейсу (API) сервісів;
- моделі даних, що репрезентують бізнес рішення;
- моделі даних репозиторіїв програмного забезпечення;
- конфігураційні дані програмного забезпечення.

Вхідні та вихідні дані кінцевих користувачів є поширеним носієм загроз та вразливостей програмного забезпечення, оскільки інтерфейс взаємодії з користувачем є відкритою точкою входу до функціональних модулів програмного забезпечення. Як уже зазначалося, задача перевірки безпеки вхідних та вихідних даних входить до вимог відомих стандартів безпеки [2] – [6], де приділено увагу таким вразливостям, як XSS,

SQL-ін'єкції, ін'єкції коду, вразливості форматування текстової інформації, переповнення буферів, вихід за діапазони індексів масивів та значень даних та іншим.

Категорія вхідних та вихідних даних прикладного програмного інтерфейсу (API) сервісів подібна до вхідних та вихідних даних кінцевого користувача, однак в цьому випадку «користувачами» виступають інші сервіси. Згідно з CWE-20, було зареєстровано такі вразливості, як цілочислові переповнення, SQL-ін'єкції, переповнення буферів даних, нескінченні цикли та ін. OWASP REST Security [9] зазначає необхідність перевірки безпеки вхідних даних з виконанням таких вимог:

- не довіряти вхідним параметрам/об'єктам;
- перевіряти введені дані: довжину, діапазон, формат і тип;
- реалізувати неявну перевірку вхідних даних, використовуючи такі строгі типи, як числа, логічні значення, дати, час, фіксовані діапазони або значення даних;
- обмежити введення рядків регулярними виразами;
- відхиляти неочікуваний/незаконний вміст.

Моделі даних, що репрезентують бізнес рішення (бізнес логіку), є внутрішніми даними, які обробляються алгоритмами програмного забезпечення. Розділ №4.10.1 фреймворку OWASP WSTG [10] визначає такі методи перевірки даних бізнес логіки:

- визначення точок введення даних;
- верифікація того, що всі перевірки безпеки та коректності даних відбуваються на сервері, і їх неможливо обійти;
- аналіз обробки алгоритмами програми порушених форматів очікуваних даних.

Моделі даних репозиторіїв програмного забезпечення з точки зору його бізнес логіки можна вважати комбінацією вхідних (джерелами даних є репозиторії типу баз даних та ін.) та внутрішніх даних (моделі даних репозиторіїв можуть оброблятися алгоритмами програмного забезпечення) [11] – [13]. Якщо дані можуть надходити в репозиторії із зовнішніх джерел, а програмне забезпечення вважає такі репозиторії довіреним джерелом даних, то воно може бути вразливим. Якщо моделі даних репозиторіїв використовуються як внутрішні моделі даних та/або вхідні чи вихідні дані, то для них, відповідно, актуальні вразливості, які було наведено раніше. Якщо ORM рівень та його моделі даних розроблено як самостійний компонент, то перевірку таких даних варто реалізувати явно, вважаючи їх вхідними/вихідними даними API.

Конфігураційні дані програмного забезпечення використовуються для налаштування його роботи за заданими параметрами без необхідності зміни вихідного коду та/або повторного постачання чи розгортання. Відповідна категорія вразливостей часто носить назву Setting Manipulation [14]. Рекомендовано такі методи захисту:

- обмеження доступу до конфігураційних даних;
- архітектурний дизайн ПЗ: визначення набору допустимих значень конфігурації;
- перевірка достовірності значень конфігурації, які задаються адміністраторами;
- статичний аналіз (SAST).

Метою роботи є підвищення захищеності допоміжного та кінцевого програмного забезпечення за рахунок перевірки безпеки різнотипних даних серверних та клієнтських модулів програмного забезпечення в процесі його функціонування.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Аналіз існуючих засобів моніторингу безпеки даних. Доцільним є розгляд існуючих засобів перевірки безпеки даних, вибір яких ґрунтується на критеріях універсальності, популярності, підтримки та охопленні найбільш популярних платформ розробки [15].

Бібліотека FluentValidation — це бібліотека .NET для створення строго типізованих наборів правил перевірки даних [16]. Бібліотека дозволяє створювати ООП класи, так звані «валідатори», із заданням наборів правил перевірки даних. Недоліками є: задання правил перевірки лише у вихідному коді на етапі розробки програмного забезпечення, підтримка лише серверної перевірки даних ПЗ на платформі .NET, інтеграція лише з однією технологією ASP.NET, відсутність можливості відтворення та збереження наборів правил перевірки у форматах типу JSON чи XML, фіксована кількість рівнів критичності помилок (error, warning, info).

Специфікація Jakarta Bean Validation — це специфікація Java, яка дозволяє задавати обмеження на моделі об’єктів за допомогою анотацій або створювати власні обмеження з можливістю розширення та надає API для перевірки об’єктів та графів об’єктів, а також API для перевірки параметрів та значень, що повертаються методами або конструкторами [17]. До недоліків можна віднести: обмеженість задання власних алгоритмів перевірки даних анотаціями, складність розширення власними правилами перевірки, відсутність асинхронної перевірки, орієнтованість лише на платформу Java, відсутність підтримки сучасних веб-технологій, обмеженість зберігання правил перевірки в XML файлах, складність імплементації задання контексту перевірки, обмеженість результатів перевірки лише двома значеннями: Info та Error.

Бібліотека Yup — це засіб перевірки даних мовою JavaScript на основі схем перевірки заданих у вигляді JavaScript об’єктів [18]. Використовується для перевірки даних користувацьких форм введення та запитів API. Бібліотека надає інформативний та детальний інтерфейс результатів перевірки. До недоліків можна віднести: ускладнений API для використання контексту перевірки даних, ускладнений API для створення власних правил перевірки, генерація схем перевірки з формату JSON вимагає сторонніх бібліотек, орієнтованість лише на платформу JavaScript, відсутність інтеграції із серверними технологіями розробки ПЗ, обмеженість результатів перевірки значеннями true/false.

Формалізація актуальних проблем предметної області. На основі виявлених переваг та недоліків існуючих засобів моніторингу безпеки даних та рекомендованих методів захисту даних програмного забезпечення визначено наступні актуальні проблеми предметної області, що потребують вирішення:

- відсутність універсальних крос-платформних рішень перевірки безпеки різних категорій даних програмного забезпечення;
- відсутність рішень синхронізованої та інтегрованої перевірки безпеки даних між клієнтськими і серверними частинами програмного забезпечення;
- відсутність універсальних рішень, які б поєднували популярні інтерфейси задання правил перевірки даних, інтеграцію з популярними технологіями розробки програмного забезпечення та гнучкість розширення;
- підвищення рівня захищеності програмного забезпечення за рахунок повторного використання уніфікованої інформаційної технології, яка би скоротила зусилля на імплементацію рекомендованих методів перевірки безпеки різних категорій даних програмного забезпечення.

Формалізація вимог до засобу моніторингу безпеки даних. Досягнення поставленої мети роботи вимагає відповідності засобу таким вимогам. Вимоги до інтерфейсу інтеграції та використання:

- підтримка методів задання наборів правил перевірки: Fluent Interfaces на стадії розробки програмного забезпечення та серіалізація/десеріалізація на етапі його виконання;
- можливість задання правил перевірки як для всієї моделі даних, так і для її окремих полів;
- можливість задання власних правил перевірки у вигляді лямбда-виразів або inline-функцій;
- підтримка рівнів критичності Success/Error (true/false) для стандартного набору правил перевірки;
- можливість задання довільних рівнів критичності результатів моніторингу для власних правил перевірки;
- можливість задання результатів перевірки рівнів критичності Success та Error;
- можливість задання власних об’єктів результатів перевірки;
- можливість задання об’єкту контексту для власних правил перевірки;
- незалежність від локалізації за рахунок формату результатів перевірки «ключ/значення»;
- іменоване групування наборів правил перевірки.

Вимоги до процесу моніторингу даних:

- моніторинг безпеки даних згідно з рекомендованими методами;
- синхронний моніторинг за замовчуванням;
- параметризація моніторингу «до першої помилки» або «повна перевірка»;
- виконання групи правил перевірки за заданою назвою групи;
- результати моніторингу повинні містити назви полів об’єкту, ідентифікатори виконаних правил перевірки, результати перевірки кожного правила, перевірені значення полів об’єкту даних.

Структура інформаційної технології. Інформаційна технологія моніторингу безпеки даних програмного забезпечення складається з наступних процесів:

- Інтеграція — інтеграція засобу моніторингу в цільове програмне забезпечення на етапі його розробки;
- Задання правил — задання правил перевірки для об’єктів даних та їх полів;
- Перевірка даних — перевірка об’єктів даних за заданими наборами правил перевірки на етапі виконання цільового програмного забезпечення;
- Обробка результатів — обробка результатів перевірки об’єктів даних згідно з задачами програмного забезпечення.

На рис. 1 представлено схему процесів інформаційної технології.

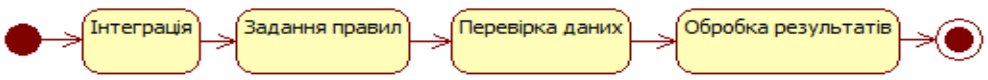

Рис. 1. Схема процесів інформаційної технології у вигляді UML-діаграми станів

Процес «Інтеграція» засобу моніторингу є тривіальним та включає два етапи: інсталяцію засобу моніторингу та інтеграцію контролера — компоненту управління засобом моніторингу. Процес можна представити так:

$$Int = \{Inst, Ctl_k\}, \quad (1)$$

де *Inst* — інстальований засіб; Ctl_k — набір k екземплярів управляючого API, $k > 0$.

Процес «Задання правил» є чотирирівневим ітеративним процесом для множин об'єктів, полів, правил та можливих результатів перевірки. Модель процесу можна представити так:

$$RSD = \{DOS_n\}, \quad (2)$$

де DOS_n — набори правил перевірки для n об'єктів, заданих на етапі «Специфікація об'єктів даних», $n > 0$.

Етап «Специфікація об'єктів даних» можна подати у вигляді наступної моделі:

$$DOS = \{FS_m\}, \quad (3)$$

де FS_m — набори правил m полів об'єкту, заданих на етапі «Специфікація полів», $m > 0$.

Етап «Специфікація полів» представляє собою задання набору правил перевірки для кожного обраного поля об'єкту даних. Для одного поля об'єкту даних модель етапу можна подати так:

$$FS = \{RS_p\}, \quad (4)$$

де RS_p — набір заданих p правил перевірки значень поля об'єкту даних на етапі «Специфікація правил», $p > 0$.

На етапі «Специфікація правил» задаються правила для перевірки значення поля. Кожне правило являє собою функцію від значення поля та вектору параметрів перевірки:

$$RS = \{F(x, \langle p_s \rangle, Ctx), ReS\}, \quad (5)$$

де F — функція перевірки, реалізована у вигляді програмного алгоритму; x — значення поля об'єкту, яке перевіряється; $\langle p_s \rangle$ — вектор s параметрів правила перевірки (мінімально допустиме значення, максимально допустиме значення і т. д.), $s \geq 0$; Ctx — контекст перевірки для власних правил перевірки; ReS — набір результатів, заданих для можливих значень F на етапі «Специфікація результатів».

На етапі «Специфікація результатів» задаються результати перевірки для можливих значень функції F правила перевірки об'єкту даних. Модель етапу можна подати так:

$$ReS = R_v, \ F(x) = y_v, \quad (6)$$

де R_v — набір репрезентативних об'єктів можливих результатів перевірки, заданих для v можливих значень функції F правила перевірки, $v > 0$; y_v — множина v можливих значень функції F правила перевірки.

Кожен об'єкт R, що ставиться у відповідність можливому значенню функції F правила перевірки, представляє собою деяку множину даних:

$$R=\{y, d_w\}, \quad (7)$$

де y — можливе значення функції F; d_w — набір w полів даних використання у процесі «Обробка результатів», $w \geq 0$.

Процес «Перевірка даних» представляє собою перевірку об'єктів даних програмного забезпечення на основі правил перевірки, заданих у процесі «Задання правил». Модель процесу подано наступним чином:

$$DOV = \{Ctx, VE\}, \quad (8)$$

де Ctx — етап «Підготовка контексту» перевірки; VE — етап «Виконання перевірки».

Етап «Підготовка контексту» можна представити у вигляді моделі:

$$Ctx = \{d_y\}, \quad (9)$$

де d_y — набір y даних цільового ПЗ, які використовуються функцією F, $y \geq 0$.

ISSN 2663 - 4023

Етап «Виконання перевірки» можна подати у вигляді наступної моделі:

$$VE = Ctl_i(obj, Ctx, RSD), \quad (10)$$

де Ctl_i — i-й екземпляр управляючого API, як зазначено у моделі (1), $0 < i \leq k$; obj — об'єкт даних, безпека якого перевіряється.

Імплементація процесу «Обробка результатів» може бути різною в цільовому ПЗ, однак спільною є модель результатів перевірки, яка продукується процесом «Перевірка даних». Результатом перевірки є підмножина заданих у процесі «Задання правил» репрезентативних об'єктів ReS, описаних у моделі (6), вибраних відповідно до значень функцій F. Модель процесу «Обробка результатів» можна подати наступним чином:

$$VR = \{R_z\}, \ R[y]_z \in \{y_z\}, \quad (11)$$

де z — кількість результатів перевірки, вибраних в процесі перевірки об'єкту та його полів, а $0 < z \leq m \times p \times v$.

Таким чином, структура інформаційної технології виглядає так:

$$I = \{Int, RSD, DOV, VR\}. \quad (12)$$

Архітектура фреймворку. На рис. 2 наведено узагальнена модель роботи фреймворку на прикладі сценарію моніторингу безпеки даних, які вводить кінцевий користувач. Моніторинг даних інших категорій відбуватиметься тим же способом завдяки універсальності розроблюваної архітектури.

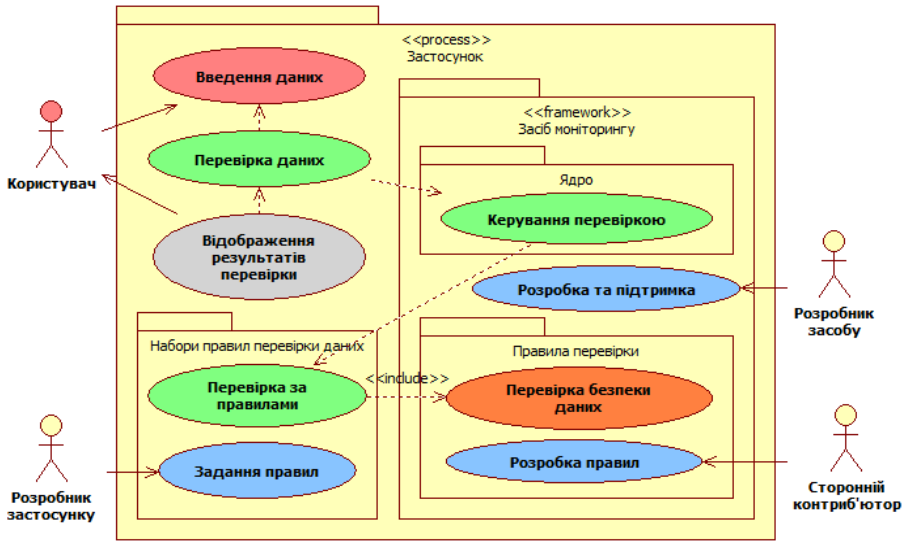

Рис. 2. Узагальнена модель роботи фреймворку

Основною загрозою застосунку в зображеній моделі є дані користувача «Користувач» (як легітимного, так і зловмисника) в процесі «Введення даних». При цьому «Застосунок» повинен запускати процес «Перевірка даних», який управляється ядром фреймворку «Засіб моніторингу» як «Керування перевіркою». Ядро фреймворку шукає «Набори правил перевірки даних», які створюються розробниками «Розробник застосунку» в процесі «Задання правил» з використанням модулів «Правила перевірки», які постачаються як самим фреймворком «Розробник засобу» в процесі «Розробка та підтримка», так і сторонніми розробниками «Сторонній контриб'ютор». В ході перевірки об'єктів даних «Перевірка за правилами» виконуються задані «Правила перевірки» які включають процес «Перевірка безпеки даних». В процесі перевірки даних

ISSN 2663 - 4023

генеруються результати перевірки, які презентуються користувачу в процесі «Відображення результатів перевірки».

Універсальну архітектуру фреймворку представлено на рис. 3, де наведено основні компоненти архітектури.

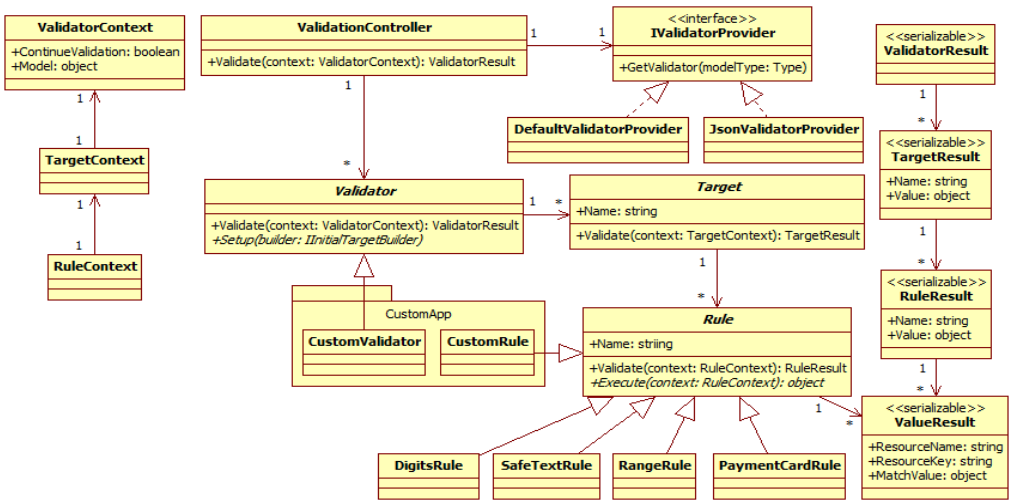

Рис. 3. Універсальна архітектура фреймворку у вигляді діаграми класів

Базовий клас Validator забезпечує перевірку об’єкту даних на основі набору правил. Окремо визначено базовий клас Rule, де похідні класи реалізують рекомендовані методи перевірки. Моніторинг даних на основі правил зводиться до перевірки всіх чи окремих її полів. Також можливою є перевірка всього об’єкту як цілісної сутності або як системи взаємопов’язаних полів. Для однотипного застосування алгоритму виконання наборів правил доцільно ввести абстрактний клас Target, який представлятиме ціль перевірки, а похідні класи застосовуватимуть зазначений алгоритм за відповідною стратегією залежно від типу цілі: примітивні типи полів об’єкту, всі значення поля-масиву, одне із значень поля-масиву та інші.

Управляюче API представлено класом ValidationController, до основних задач якого відносяться хостинг екземплярів класу Validator та запуск процесу моніторингу відповідних їм об’єктів даних. Екземпляри ValidationController можна створювати для різних доменів застосунку, щоб оптимізувати використання реалізацій класів Validator за стратегією «на вимогу» та використання оперативної пам’яті. Екземпляри IValidatorProvider відповідають за інстанціювання реалізацій класів Validator для типів об’єктів даних для послідуючої перевірки. Можливими реалізаціями IValidationProvider можуть бути: провайдер JsonValidatorProvider зконфігурованих валідаторів у форматі JSON, провайдер за замовчуванням DefaultValidatorProvider для задання валідаторів у програмному коді.

Вимога моніторингу даних з урахуванням контексту виконання забезпечується набором зв’язаних класів ValidatorContext, TargetContext та RuleContext. Програмний код, який використовує фреймворк моніторингу, передає власний екземпляр класу ValidatorContext з потрібними параметрами. Даний екземпляр інкапсулюється в RuleContext та TargetContext, який передається під час моніторингу в усі правила перевірки. Результати моніторингу об’єктів даних представлені набором серіалізованих класів ValidatorResult, TargetResult, RuleResult та ValueResult. Останній представляє об’єкти результатів як порушених, так і дотриманих правил.

ISSN 2663 - 4023

Варто відзначити мінімально необхідний об'єм роботи для розробників кінцевого застосунку, де вимагається лише декларування валідаторів доменних моделей як «Набори правил перевірки даних» з повторним використанням готових правил перевірки. Всі інші процеси, такі як керування моніторингом, генерація помилок, а також постачання стандартного набору правил перевірки забезпечуються розробленим фреймворком.

Перевірка роботи та оцінка ефективності фреймворку. Інформаційну технологію моніторингу безпеки даних програмного забезпечення було реалізованою для платформи .NET у вигляді фреймворку згідно з розробленою архітектурою. Роботу розробленого засобу перевірено на прикладі форми логіна з тестовими прикладами деяких найбільш популярних загроз. Нехай задано форму логіна у вигляді ООП-класу, як показано нижче:

```
public class LoginForm {
    public string Password { get; set; }
    public string Username { get; set; }
}
```

Задано валідатор форми логіна, як показано у нижче наведеному фрагменті коду:

```
public class LoginFormValidator : Validator<LoginForm> {
    protected override void Setup(IInitialTargetBuilder<LoginForm> validate) {
        validate.Member(form => form.Password)
            .IsRequired().WithError("Errors", "Password.Required")
            .HasLength(12, null).WithError("Errors", "Password.MinLength");
        validate.Member(form => form.Username)
            .IsRequired().WithError("Errors", "Username.Required")
            .HasLength(8, 80).WithError("Errors", "Username.Length")
            .IsSafeText().WithError("Errors", "Username.Unsafe");
    }
}
```

Нижче розглянуто тестовий приклад SQL-ін'єкції. Форму логіна заповнено, як показано нижче:

```
var form = new LoginForm {
    Username = "\" or \"\"=\"",
    Password = "$ecureP@ssw0rd",
}
```

Запуск процесу моніторингу відбувається викликом контролера валідації, як показано у наступному фрагменті коду:

```
var controller = new ValidationController(new DefaultValidatorProvider(true));
var results = controller.Validate(form);
```

Юніт-тест верифікації коректної роботи фреймворку та отримання результату моніторингу з рівнем критичності «Error» наведено нижче:

```
var error = results.TargetResults.Single()
            .RuleResults.Single().ValueResults.Single();
Assert.That(error.ResourceName, Is.EqualTo("Errors"));
Assert.That(error.ResourceKey, Is.EqualTo("Username.Unsafe"));
```

ISSN 2663 - 4023

Отже, фреймворк успішно виконав моніторинг полів тестової форми логіна та визначив некоректні дані.

Розглянуто тестовий приклад Cross-site Scripting (XSS). Формі логіна задано код Reflected XSS скрипта, як показано нижче:

```
var form = new LoginForm {
    Username = "<script>document.location=” +
               ”\"https://grabber.com/?c=\"+document.cookie</script>",
    Password = "$ecureP@ssw0rd",
}
```

Процес запуску моніторингу, тест верифікації та отримані результати перевірки залишилися незмінними, оскільки правила перевірки було задано за принципом «білого списку» згідно з рекомендованими методами перевірки. В даному тесті фреймворк успішно виконав моніторинг полів форми логіна та визначив некоректні дані.

Розглянуто тестовий приклад Out-of-bounds Write. Найбільш популярною вразливістю згідно з CWE Top 25 є переповнення буферів даних, що може бути спричинено виходом за межі діапазону допустимого розміру даних (символьної довжини текстових даних). Нехай форма логіна містить текстові дані, що виходять за межі дозволеної довжини 80 символів, як показано нижче:

```
var form = new LoginForm {
    Username = "1234567890 1234567890 1234567890 1234567890 " +
               "1234567890 1234567890 1234567890 1234567890 ",
    Password = "$ecureP@ssw0rd",
};
```

Як і в попередніх прикладах, варто відзначити, що код запуску моніторингу залишається незмінним. Код тесту верифікації передбачає спрацювання правила перевірки LengthRule(8, 80), як показано у наступному фрагменті коду:

```
var error = results.TargetResults.Single()
            .RuleResults.Single().ValueResults.Single();
Assert.That(error.ResourceName, Is.EqualTo("Errors"));
Assert.That(error.ResourceKey, Is.EqualTo("Username.Length"));
```

Фреймворк успішно виконав моніторинг полів форми логіна та визначив некоректну довжину поля «Username».

Ефективність використання розробленого фреймворку в цільовому програмному забезпеченні для імплементації безпеки даних було обчислено як відношення кількості стрічок вихідного та виконуваного коду, необхідного для перевірки безпеки вище наведеної форми логіна з використанням розробленого фреймворку та без нього. Було використано метрики коду, які надає IDE Microsoft Visual Studio Community 2022, де розроблявся програмний фреймворк. Результати метрик вихідного коду представлено в табл. 1.

Таблиця 1

Кількість стрічок вихідного коду перевірки безпеки форми логіна

Категорія коду	Без використання фреймворку	З використанням фреймворку
Правило IsRequired	31	0
Правило HasLength	63	0
Правило IsSafeText	46	0
Зберігання результатів	40	0

Код запуску моніторингу	2	2
Опис правил перевірки	18	18
Всього	200	20

Отже, у випадку використання фреймворку розробнику цільового програмного забезпечення потрібно написати у 10 разів менше вихідного коду. Відповідно, результати метрик виконуваного коду представлено в табл. 2.

Таблиця 2

Кількість стрічок виконуваного коду перевірки безпеки форми логіна

Категорія коду	Без використання фреймворку	З використанням фреймворку
Правило IsRequired	2	0
Правило HasLength	7	0
Правило IsSafeText	8	0
Зберігання результатів	4	0
Код запуску моніторингу	2	2
Опис правил перевірки	4	4
Всього	27	6

Отже, у випадку використання фреймворку розробнику цільового програмного забезпечення потрібно написати у 4.5 рази менше виконуваного коду.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Досліджено процеси, методи та засоби моніторингу безпеки даних програмного забезпечення. Було встановлено, що вразливості та загрози різних категорій даних програмного забезпечення входять до списку найбільш актуальних і для його безпечного функціонування необхідним є моніторинг безпеки усіх категорій даних.

На основі моделювання процесів інформаційної технології було спроектовано універсальну архітектуру фреймворку моніторингу безпеки даних. Обраний підхід конфігурації на основі правил забезпечив відповідність засобу рекомендованим методам перевірки безпеки даних та розширюваність. Успішність розробленої архітектури було підтверджено реалізацією фреймворку для платформи .NET. Розроблений засіб було успішно перевірено на прикладах таких загроз, як SQL-ін’єкції, Cross-site Scripting та Out-of-bounds Write. Основні переваги розробленого засобу моніторингу:

- моніторинг безпеки даних згідно з рекомендованими методами;
- задання наборів правил перевірки як на етапі розробки, так і на етапі виконання цільового програмного забезпечення;
- задання власних правил перевірки у вигляді лямбда-виразів;
- можливість задання довільних рівнів критичності власних правил перевірки;
- можливість задання власних об’єктів результатів перевірки;
- незалежність від локалізації;
- іменоване групування правил перевірки;
- скорочення часу на інтеграцію рекомендованих методів перевірки безпеки даних у 10 разів на основі кількості стрічок вихідного коду та 4.5 рази на основі кількості стрічок виконуваного коду.

До основних труднощів, що виникли в ході розробки, можна віднести розгалуженість ієрархії класів для забезпечення розширюваності набору правил

перевірки та процес розробки ієрархії класів інтерфейсу задання наборів правил перевірки на етапі розробки програмного забезпечення.

Основні напрямки подальшого вдосконалення розробленого фреймворку:

- реалізація універсальної архітектури засобу для платформ Java та JavaScript;
- розширення стандартного набору правил перевірки;
- розробка адаптерів для інтеграції засобу моніторингу з відомими фреймворками та бібліотеками;
- підтримка об'єктно-орієнтованого стилю задання наборів правил перевірки.

Передбачається, що розроблений засіб набуде широкого використання як у комерційній розробці програмного забезпечення, так і в навчальній та науковій галузі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Demography of Europe - A growing population until 2020*. (n.d.). Language selection | European Commission. https://ec.europa.eu/eurostat/web/interactive-publications/digitalisation-2023.
2. *Official PCI Security Standards Council Site*. (n.d.). PCI Security Standards Council. https://www.pcisecuritystandards.org
3. *CWE - CWE-20: Improper Input Validation (4.14)*. (n.d.). CWE - Common Weakness Enumeration. https://cwe.mitre.org/data/definitions/20.html
4. *Security Knowledge Framework*. (n.d.). Security Knowledge Framework. https://www.securityknowledgeframework.org
5. *OWASP Web Security Testing Guide|OWASP Foundation*. (n.d.). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. https://owasp.org/www-project-web-security-testing-guide
6. *OWASP Top Ten|OWASP Foundation*. (n.d.). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. https://owasp.org/www-project-top-ten
7. Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
8. Martin, R. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson Education Asia.
9. *REST Security - OWASP Cheat Sheet Series*. (n.d.). Introduction - OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html#input-validation
10. *WSTG - Stable | OWASP Foundation*. (n.d.). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/10-Business_Logic_Testing.
11. *P of EAA: Data Transfer Object*. (n.d.). martinfowler.com. https://martinfowler.com/eaaCatalog/dataTransferObject.html
12. Smith, J. (2021). *Entity Framework Core in Action, Second Edition*. Manning Publications Co. LLC.
13. Lerman, J. (2010). *Programming Entity Framework: Building Data Centric Apps with the Ado. Net Entity Framework*. O'Reilly Media, Incorporated.
14. *A Taxonomy of Coding Errors that Affect Security*. (n.d.). Software Security|Setting Manipulation. https://vulncat.fortify.com/en/detail?id=desc.dataflow.cfml.setting_manipu-lation.
15. *Рейтинг мов програмування 2023*. (2023). Спільнота програмістів|DOU. https://dou.ua/lenta/articles/language-rating-2023
16. *FluentValidation — FluentValidation documentation*. (n.d.). FluentValidation — FluentValidation documentation. https://docs.fluentvalidation.net/en/latest
17. *Jakarta Bean Validation - Home*. (n.d.). Jakarta Bean Validation - Home. https://beanvalidation.org
18. *GitHub - jquense/yup at pre-v1*. (n.d.). GitHub. https://github.com/jquense/yup/tree/pre-v1

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (23), 2024
ISSN 2663 - 4023

Leonid Kupershtein
PhD, Associate Professor of Information Protection Department
Vinnytsia National Technical University, Vinnytsia, Ukraine
ORCID 0000-0001-6737-7134
kuperstein.lm@gmail.com

Hennadii Lutsyshyn
Master in Cyber Security
Vinnytsia National Technical University, Vinnytsia, Ukraine
ORCID 0009-0001-5753-5407
abitstudent16@gmail.com

Mykhailo Krentsin
Ph.D student in Cyber Security, Information Protection Department
Vinnytsia National Technical University, Vinnytsia, Ukraine
ORCID 0000-0002-1792-9401
mishatron98@gmail.com

INFORMATION TECHNOLOGY OF SOFTWARE DATA SECURITY MONITORING

Abstract. The article proposes an overview of an information technology of software data security monitoring with the aim of increasing the security of auxiliary and end-user software in run-time using the designed universal architecture with the capabilities of synchronized data security validation processes in client-server interactions and integration into software development technologies using unified interfaces for specifying extensible sets of validation rules. An analysis of the subject area was conducted, during which it was established that the tasks of software data security validation are included in the requirements of well-known data security standards, and the related vulnerabilities are noted as highly important. It was also established that the traditional monitoring of only software input data is not enough, therefore the problem of security monitoring of various categories of software data is relevant. The capabilities of existing tools to monitor security and correctness of software data are incomplete: focus on certain categories of data, platform dependency, narrow integration capabilities with other software development tools, limited usage, complex or limited extensibility, difficult reuse of well-known verified solutions etc. Based on the analysis of the existing software data security monitoring tools, the cons of the existing implementations were identified, and a universal architecture of the software framework was proposed as a solution. The analysis of the vulnerabilities of various categories of software data was conducted along with the recommended methods of implementation of data security. Discovered methods of data security implementation were used in the proposed solution. The article examines the structure of the proposed information technology, provides the universal architecture of the software framework, demonstrates the verification of the work of the developed tool, and provides the assessment of the effectiveness of usage of the data security monitoring framework in software development. The proposed architecture and directions of further improvements of the framework allow significant extension of its functionality and easy integration into popular software development technologies. It is assumed that the developed information technology of software data security monitoring will be widely used in commercial software development as well as in educational and scientific appliance.

Keywords: application security; data monitoring; Cross-site Scripting; SQL-injections; input validation; validation errors.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. *Demography of Europe - A growing population until 2020.* (n.d.). Language selection | European Commission. https://ec.europa.eu/eurostat/web/interactive-publications/digitalisation-2023.

2. *Official PCI Security Standards Council Site.* (n.d.). PCI Security Standards Council. https://www.pcisecuritystandards.org
3. *CWE - CWE-20: Improper Input Validation (4.14).* (n.d.). CWE - Common Weakness Enumeration. https://cwe.mitre.org/data/definitions/20.html
4. *Security Knowledge Framework.* (n.d.). Security Knowledge Framework. https://www.securityknowledgeframework.org
5. *OWASP Web Security Testing Guide|OWASP Foundation.* (n.d.). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. https://owasp.org/www-project-web-security-testing-guide
6. *OWASP Top Ten|OWASP Foundation.* (n.d.). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. https://owasp.org/www-project-top-ten
7. Fowler, M. (2002). *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional.
8. Martin, R. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Pearson Education Asia.
9. *REST Security - OWASP Cheat Sheet Series.* (n.d.). Introduction - OWASP Cheat Sheet Series. https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html#input-validation
10. *WSTG - Stable | OWASP Foundation.* (n.d.). OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation. https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/10-Business_Logic_Testing.
11. *P of EAA: Data Transfer Object.* (n.d.). martinfowler.com. https://martinfowler.com/eaaCatalog/dataTransferObject.html
12. Smith, J. (2021). *Entity Framework Core in Action, Second Edition.* Manning Publications Co. LLC.
13. Lerman, J. (2010). *Programming Entity Framework: Building Data Centric Apps with the Ado. Net Entity Framework.* O'Reilly Media, Incorporated.
14. *A Taxonomy of Coding Errors that Affect Security.* (n.d.). Software Security|Setting Manipulation. https://vulncat.fortify.com/en/detail?id=desc.dataflow.cfml.setting_manipu-lation.
15. *Programming languages rating 2023.* (2023). Community of programmers|DOU. https://dou.ua/lenta/articles/language-rating-2023
16. *FluentValidation — FluentValidation documentation.* (n.d.). FluentValidation — FluentValidation documentation. https://docs.fluentvalidation.net/en/latest
17. *Jakarta Bean Validation - Home.* (n.d.). Jakarta Bean Validation - Home. https://beanvalidation.org
18. *GitHub - jquense/yup at pre-v1.* (n.d.). GitHub. https://github.com/jquense/yup/tree/pre-v1

(cc) BY-NC-SA

This work is licensed under Creative Commons Attribution-noncommercial-sharealike 4.0 International License.