УДК 004.056.5

В.Л. Бурячок,
доктор технічних наук, професор,
Київський університет імені Бориса Грінченка, м. Київ,
С.О.Спасітєлєва,
кандидат фізико-математечиних наук, доцент,
Київський університет імені Бориса Грінченка, м. Київ,
П.М. Складанний
Київський університет імені Бориса Грінченка, м. Київ

ОРГАНІЗАЦІЯ РОЗРОБКИ БЕЗПЕЧНИХ .NET ПРИКЛАДНИХ ПРОГРАМ У ЗАКЛАДАХ ВИЩОЇ ОСВІТИ

Проаналізовано стан розроблення програмного забезпечення в Україні та у світі загалом. Розглянуто основні методи та засоби створення безпечного коду на всіх стадіях циклу розробки безпечного програмного забезпечення, а також процес створення безпечних додатків платформи .NET з використанням технологій статичного й динамічного аналізу коду для пошуку та усунення вразливостей коду на стадії реалізації ПЗ. Акцентовано увагу на високій вартості виправлення програмних помилок. Запропоновано застосування безпечного циклу створення програмних продуктів Security Development Lifecycle (SDL) та нових технологій пошуку універсальних ознак груп вразливостей з метою їх ефективного виявлення на всіх стадіях життєвого циклу програмного забезпечення. Запропоновано шляхи подальшого розвитку системного підходу до навчання сучасних фахівців, здатних вже на етапі написання програм виявляти й усувати знайдені в них вразливості. Запропоновано практичне застосування розглянутих методів та комплексу інструментальних засобів розроблення безпечного коду, таких як IDE MS Visual Studio та IBM Security AppScan засобів у навчальному процесі підготовки спеціалістів з інформаційної та кібернетичної безпеки. Запропоновано використання комплексу сучасних інструментальних засобів розробки безпечного коду в рамках вивчення дисципліни з основ програмування.

Ключові слова: *інформаційна безпека, безпечне програмне забезпечення, вразливість програмного коду, статичний аналіз коду, динамічний аналіз коду, життєвий цикл розробки безпечного ПЗ.*

Проанализировано состояние разработки программного обеспечения в Украине и в мире в целом. Рассмотрены основные методы и средства создания безопасного кода на всех стадиях цикла разработки безопасного программного обеспечения, а также процесс создания безопасных приложений платформы .NET с использованием технологий статического и динамического анализа кода для поиска и устранения уязвимостей кода на стадии реализации ПО. Акцентировано внимание на высокой стоимости исправления программных ошибок. Предложено применение безопасного цикла создания программных продуктов Security Development Lifecycle (SDL) и новых технологий поиска универсальных признаков групп уязвимостей с целью их эффективного обнаружения на всех стадиях жизненного цикла программного

обеспечения. Предложены пути дальнейшего развития системного подхода к обучению специалистов, способных уже на этапе написания программ выявлять и устранять найденные в них уязвимости. Предложено практическое применение рассмотренных методов и комплекса инструментальных средств разработки безопасного кода, таких как IDE MS Visual Studio и IBM Security AppScan средств в учебном процессе подготовки специалистов по информационной и кибернетической безопасности. Предложено использование комплекса современных инструментальных средств разработки безопасного кода в рамках изучения дисциплины по основам программирования.

Ключевые слова: *информационная безопасность, безопасное программное обеспечение, уязвимость программного кода, статический анализ кода, динамический анализ кода, жизненный цикл разработки безопасного ПО.*

The state of software development in Ukraine and in the whole world is analyzed. The main methods and means of creating secure code at all stages of the safe software development cycle are discussed, as well as the process of creating .NET secure applications using static and dynamic code analysis techniques for searching and eliminating code vulnerabilities at the software implementation stage. Attention is focused on the high cost of correcting program errors. The application of the secure development cycle of the Security Development Lifecycle (SDL) software and new technologies for searching for universal attributes of vulnerability groups with the purpose of their effective detection at all stages of the software life cycle is proposed. The ways of the further development of the system approach to the training of specialists able at the stage of writing programs to identify and eliminate vulnerabilities found in them are suggested. The practical application of the methods and a set of tools for developing secure code such as IDE MS Visual Studio and IBM Security AppScan in the training process of training specialists in information and cyber security is offered. The use of a complex of modern tools for the development of secure code within the framework of studying the discipline on the basics of programming is proposed.

Keywords*: information security, safe software, software code vulnerability, static code analysis, code analysis, life cycle development of secure software.*

Сучасний світ неможливо уявити без програмного забезпечення (ПЗ). Розробка ПЗ перетворилася на окрему галузь, що посідає вагоме місце у світовій економіці, у тому числі в Україні. У 2016 році світові витрати на ІТ-послуги склали 1229 млрд дол. США, і ця галузь зростає щороку, у тому числі спостерігається зростання на 3,2 % у 2017 році [1]. За останні шість років обсяги розробки ПЗ в Україні виросли майже втричі [1]. Україна нині – провідний центр із розробки програмного забезпечення у Східній та Центральній Європі й посідає четверте місце за експортом ІТ-продуктів і послуг у світі. При цьому із розвитком технологій, з новою технологічною революцією, цифровізацією світу значення розробки програмного продукту лише зростатиме. У зв'язку із зазначеним першочерговим завданням індустрії розробки ПЗ стає безпека.

Висока складність сучасного програмного забезпечення, яка зумовлена великим обсягом вихідного коду, інтегруванням методів обфускації, наявністю дефектів (вразливостей і некритичних помилок), є фундаментальною проблемою сучасного етапу розвитку інформаційних технологій. Аналіз захищеності різних

груп ПЗ, який проводиться експертами дослідницьких лабораторій у всьому світі, щороку знаходить в ПЗ мільйони вразливостей. Велика їх частина виявляється розробниками ще на ранніх стадіях розробки ПЗ за допомогою аналізу архітектури ПЗ, статичного аналізу вихідних текстів, тестування з точки зору реалізації загроз безпеці [2]. Впровадження подібних процедур в практику вітчизняних розробників програмного забезпечення підвищить рівень захищеності створюваного ПЗ і, як наслідок, значно зменшить кількість інцидентів інформаційної безпеки.

Пошук і усунення дефектів у ПЗ вимагає значних витрат, при цьому багато з них можуть залишитися невиявленими. За даними дослідження, проведеного на замовлення Національного інституту стандартів і технологій США, можна констатувати, що збитки, які виникають через недостатньо розвинену інфраструктуру процесу усунення вразливостей і некритичних помилок в ПЗ, складають нині від 22 до 60 мільярдів доларів на рік [3]. Вартість усунення дефекту, пропущеного на етапах розробки і тестування, може зрости на етапі експлуатації програми від 2 до 100 разів [4]. За оцінкою дослідників Гарвардського університету, виправлення помилки, виявленої на етапі тестування, в середньому коштує 960 $, а помилки, виявленої на етапі експлуатації, 7600 $.

Одним з варіантів вирішення зазначених проблем безпеки ПЗ може стати застосування безпечного циклу створення програмних продуктів (Security Development Lifecycle – SDL), який розпочинатиметься ще до початку їх розробки. У рамках SDL необхідно планувати, проектувати, реалізовувати, тестувати підсистеми безпеки на кожному етапі життєвого циклу програм, тобто займатися безпекою прикладної програми постійно. Такий підхід дозволить отримати на виході програмний продукт з продуманою системою безпеки, який не потребуватиме в перспективі багаторазової переробки через існуючі вразливості. Для практичної реалізації такого принципу великі фірми з розробки ПЗ радять у кожному програмному проекті мати хоча б одного спеціаліста з безпеки [3].

Усе викладене вище фактично дає можливість стверджувати, що в контексті нових загроз та тенденцій розвитку інформаційної безпеки та у зв'язку зі зростанням обсягу розроблюваної ПЗ проблеми створення безпечного коду стають особливо актуальними. Час, який зазвичай минає між моментом виявлення вразливості програмного коду і моментом її усунення, може становити до півроку, тому завдання написання коду, позбавленого певних вразливостей завдяки їх виявленню на початкових стадіях проектування та реалізації, дуже актуальне.

Безпека додатка має закладатися на етапі написання вихідного коду. Для цього необхідна підготовка сучасних спеціалістів у галузі інформаційної безпеки та інформаційних технологій, які мають не тільки навички програмування, а й добре ознайомлені з питаннями розробки захищеного програмного забезпечення. Сьогодні стає конче необхідним підвищення якості підготовки спеціалістів з кібербезпеки, одним із шляхів якого є введення в базові нормативні курси лінійки інструментальних засобів безпеки, які пропонуються провідними фірмами.

Враховуючи це, метою статті є пошук та обґрунтування шляхів підвищення захищеності програмних додатків, основними серед яких, як на наш розсуд, є: застосування нових технологій виявлення універсальних ознак груп вразливостей на всіх стадіях життєвого циклу ПЗ та подальший розвиток системного підходу

до навчання сучасних фахівців, здатних вже на етапі написання програм виявляти й усувати знайдені в них вразливості. Саме поєднанню цих напрямків присвячена стаття.

Сьогодні використання аналізаторів коду для ідентифікації вразливостей стає об'єктивною потребою для забезпечення захищеності сучасних програмних систем з високим рівнем довіри. Уведення нових стандартів розробки безпечного ПЗ підтверджує необхідність ефективних інструментів, що підтримують якість програмного коду [4]. Актуальна загальноприйнята класифікація дефектів (вразливостей і помилок) за типами представлена в базі Common Weakness Enumeration [5], список зареєстрованих вразливостей – у базі Common Vulnerabilities and Exposures організації MITRE [6]. Аналіз програмного коду на наявність уразливостей зазначених у цих документах на ранніх стадіях розроблення може значно підвищити захищеність додатків. На етапі проектування та розроблення можна ефективно виявляти та усувати такі вразливості, як порушення безпеки доступу до пам'яті (переповнення буфера, висячі вказівники), помилки перевірки введених даних (помилки форматування рядка, ін'єкція коду, SQL ін'єкція, міжсайтовий скриптинг у веб-додатках), помилки в плутанині та перевищення привілеїв, помилки синхронізації (взаємне блокування), витоки пам'яті та інших ресурсів системи, некоректна робота з тимчасовими файлами і іншими інтерфейсами ОС, уразливості безпеки (слабке шифрування, зберігання пароля в явному вигляді).

Методи аналізу коду. Технології пошуку вразливостей і захисту від них розробляються за трьома основними напрямами [4].

По-перше, використовуються системи автоматичного пошуку вразливостей (англ. Static Application Security Testing – SAST) за допомогою статичного аналізу вихідного коду програм, які можна застосовувати на самих ранніх етапах розробки ПЗ, що робить виправлення дефектів максимально дешевим. Метою статичного аналізу є виявлення та усунення потенційно уразливих конструкцій у вихідному коді програми, а також формування вихідних даних для виконання завдань динамічного аналізу й тестування на проникнення в рамках процесу кваліфікаційного тестування.

Системи статистичного аналізу обробляють прикладні програми в мільйони рядків коду, дозволяють виділити можливі уразливості в "статичному" (невиконуваному) вихідному коді. Вони мають прийнятний рівень хибних реакцій (false positive), за якого 30–70 % знайдених помилок виявляються істинними, а також аналізують усі можливі варіанти виконання програми одночасно.

Важливою складовою безпеки коду є застосування перевірених сторонніх компонентів – бібліотек або готових програм з відкритим вихідним кодом, тому статичний аналіз вихідного коду програми необхідно проводити також для існуючих компонентів та бібліотечних функцій. Зважаючи на таке найбільше поширення набули нині саме методи статичного аналізу вихідного коду ПЗ, які передбачають доступ не тільки до завантажувальних і об'єктних модулів, але й до вихідних текстів програмної системи, а також до інформації, пов'язаної із середовищем компіляції та виконанням програмних компонентів. При цьому статичний аналіз коду виконується зазвичай як частина етапу інспекції коду (code review), тобто дослідження програми методом "білого ящика" (white box) і виконується на етапі реалізації Security Development Lifecycle. Залежно від використаного

інструменту, глибина аналізу може змінюватися від визначення поведінки окремих операторів до глобальнішого аналізу, що включає весь наявний вихідний код.

Статичний аналіз вихідних текстів ПЗ тісно пов'язаний із принципами роботи компіляторів. Багато підходів такого аналізу базується на деяких елементах компіляції, проміжне представлення вихідних текстів у статичних аналізаторах еволюціонує разом з розвитком теорії компіляторів. Сучасні методи аналізу з метою уніфікації алгоритмів використовують різні моделі представлення коду, наприклад, лексичний розбір, синтаксичне дерево, дерево Канторовича, графи потоку даних і управління тощо. Можливе також застосовування більш точних методів аналізу, методів верифікації програм для найважливіших ділянок програми для зменшення обсягу аналізованого коду.

По-друге, застосовуються системи динамічного аналізу бінарного коду програм, що дозволяють багаторазово запускати задану програму на автоматично генерованому наборі вхідних даних і відстежувати ситуації виникнення дефектів. Системи динамічного аналізу переглядають лише частину можливих наборів вхідних даних, але при знаходженні помилки відразу дозволяють отримати дані, на яких ця помилка проявляється (тобто не мають помилкових спрацьовувань). Застосування цих систем обмежене високими вимогами до ресурсів і обмеженнями на максимальний розмір аналізованої програми (зазвичай десятки тисяч рядків коду).

Динамічний аналіз являє собою сукупність всіх методів аналізу програмного забезпечення, що реалізуються за допомогою програм на реальному або віртуальному процесорі. Такі способи дедалі частіше використовуються при дослідженні програм методом "чорного ящика" (black box), коли є доступ лише до зовнішніх інтерфейсів програмного забезпечення без урахування їх структури, внутрішніх інтерфейсів і стану. До основних завдань, які вирішуються при динамічному аналізі, належить задача генерації наборів вхідних даних для покриття визначених шляхів виконання програми-"фаззінг" (fuzzing), запуск і трансляція програми, відстеження знайдених вразливостей. Слід зауважити, що застосування традиційної методики fuzz-тестування є малоефективним при виявленні вразливостей, пов'язаних з рідкісними поєднаннями вхідних даних, наприклад, програмних закладок. Спроба згенерувати всі можливі поєднання вхідних даних призводить до експоненціального зростання їх обсягу і є практично неможливою для великих проектів.

Таким чином, нині розробниками пропонується до вибору велика кількість інструментальних засобів статичного та динамічного аналізу для різних систем програмування [7]. Точне судження про архітектуру та алгоритми аналізу, покладені в основу цих систем, ускладнюється через їх закритість, але можна знайти багато матеріалу з порівнянням результатів їх роботи. Деякі інструменти аналізу коду починають переходити до інтегрованих середовищ розробки програм (Integrated Development Environment – IDE) [8]. Також розроблено велику кількість готових інструментальних засобів, у яких застосовується як статичний аналіз, так і динамічний аналіз, що дозволяє виконувати налагодження програми в процесі її виконання. Способів і інструментів аналізу існує велика кількість, але тільки комплексне їх застосування матиме максимальний ефект.

Нарешті, по-третє, розробляються технології захисту програми й її оточення від експлуатації наявних в ній вразливостей під час роботи програми. Цей напрям

є актуальним через те, що такий підхід дозволяє істотно ускладнити зловмисникові можливість експлуатації вразливості та формування універсального методу зламу конкретної програми.

Таким чином, можна зробити висновок, що на етапі розробки безпечних прикладних програм слід застосувати такі методи:

1) статичного аналізу програмного коду з метою винятку простих вразливостей, пов'язаних з помилками й недоліками в коді програми. Такий аналіз можна здійснювати засобами IDE. Для отримання кращих результатів додатково можна застосувати спеціалізовані програмні комплекси статистичного аналізу коду;

2) динамічний аналіз вихідного коду при виконанні програми. При цьому можна використовувати засоби інтегрованого середовища розробки та спеціалізовані інструментальні засоби;

3) тестування (функціональне, навантажувальне, на проникнення) програмного коду засобами середовища розробки програм [13].

Ці методи мають бути опановані студентами спеціальності кібербезпека в процесі навчання та створення власних прикладних програм. Залишається питання підбору IDE та інструментальних засобів аналізу коду, які можуть використовуватися в процесі розробки прикладних програм, а також для аудиту готових програм за вимогами інформаційної безпеки.

Інструментальні засоби аналізу коду, розглядалися за такими вимогами:

наявність програми співпраці з ЗВО України з надання безкоштовного доступу до продуктів;

якісні технології та алгоритми для глибокого аналізу коду та виявлення всіх вразливостей;

регулярно обновлювана база правил з можливостями гнучкого налаштовування й розширення;

надання вичерпних обґрунтувань наявності уразливості і докладних рекомендацій із її усунення;

зіставлення результатів аналізу при повторному скануванні відредагованого коду;

підтримка великої кількості мов програмування;

інтеграція із середовищами розробки, системами контролю версій і системами стеження за дефектами;

мінімальна кількість помилкових спрацьовувань;

представлення результатів аналізу в зручному для сприйняття вигляді. Наявність засобів автоматичного складання звітів.

Лінійка рішень від компанії IBM Application Security Solutions, призначених для перевірки додатків на захищеність та для управління процесом виявлення вразливостей протягом всього життєвого циклу, відповідає викладеним вимогам і може використовуватися разом з IDE MS Visual Studio.

Інструментальні засоби аналізу коду.

Як середовище розробки в дослідженні обрано сімейство інструментів Microsoft Visual Studio, яке містить інтегроване середовище розробки, сервіс для організації спільної роботи, комплексне рішення для реалізації повноцінного циклу розробки мобільних додатків – Visual Studio Mobile Center, багатоплатформовий редактор коду Visual Studio Code, що робить його одним із лідерів розробки різноманітного програмного забезпечення. Visual Studio 2017 можна

використовувати для розроблення прикладних програм для Android, iOS, Windows, Linux, веб-додатків, мобільних та хмарних додатків, систем баз даних. При цьому IDE містить набір додаткових інструментів, які дозволять розробникам ПЗ створювати надійний та захищений код. Для проблем безпеки, які можуть бути виявлені під час фази розробки програмного забезпечення, це дуже важливо, оскільки забезпечується негайний зворотній зв'язок із розробником. Цей негайний зворотний зв'язок дуже корисний, особливо якщо порівнювати з вразливостями, знайденими значно пізніше на стадіях тестування та експлуатації.

У процесі застосування IDE MS Visual Studio для реалізації однотипних прикладних програм з використанням баз даних серед студентів 2 курсу спеціальності кібербезпека проводилися дослідження з порівняння та аналізу написаного різними студентами програмного коду. При цьому тільки частина студентів використовувала засоби статичного аналізу коду. Отримані результати показали, що у проектах із застосовуванням вбудованих засобів аналізу коду дизайн системи якість коду в цілому є значно кращим.

Засоби статичного аналізу коду повністю інтегровані в інтерфейс IDE Visual Studio [8]. Під час побудови всі попередження, що виробляються для вихідного коду, відображаються у списку помилок. При цьому можна перейти до вихідного коду, який створив попередження, і переглянути додаткові відомостей про причини і можливі способи усунення проблеми. Аналіз коду працює в додатках .NET Framework і додатках баз даних. Такі можливості, як навігація по коду, рефакторинг, real-time функцій модульного тестування та перевірки залежностей, виправлення й налагодження для всіх підтримуваних мов середовища також значно покращують якість, безпечність коду та продуктивність розробки. Використовуючи можливості меню Analize можна виконувати профілювання програми, статичний аналіз коду рішення або обраного проекту, змінювати установки аналізу коду для всього рішення або проекту, робити розрахунок набору метрик (цикломатичний номер графа управління програмами, кількість операторів та описів у програмі, ступінь злиття класів та методів класу для рішення) [9]. Аналіз зазначених параметрів є важливим для оцінки якості коду і його надійності (рис. 1).

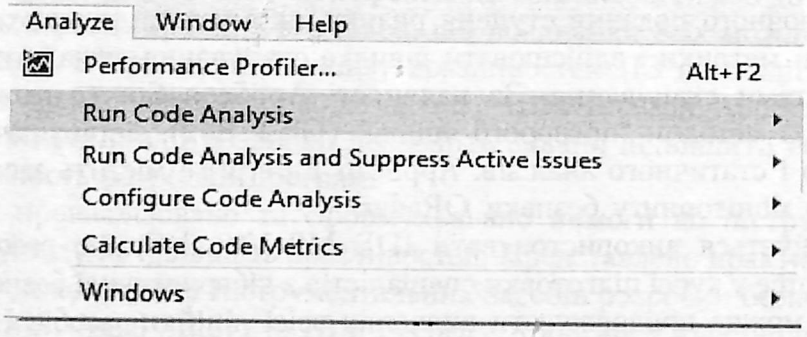

Рис. 1. Меню Analyze IDE MS Visual Studio

У вікні результатів аналізу виводиться список попереджень, обравши які можна знайти місце розташування підозрілого коду у файлах проекту та усунути відповідний дефект (рис. 2).

Порівняння спеціалізованих та вбудованого в IDE Visual Studio статичних аналізаторів коду свідчить про перевагу спеціалізованих засобів [10]. Різниця в кількості знайдених помилок спеціалізованим засобом та статичним аналізатором коду вбудованим в IDE може досягати таких цифр: 72 % та 12 % [8]

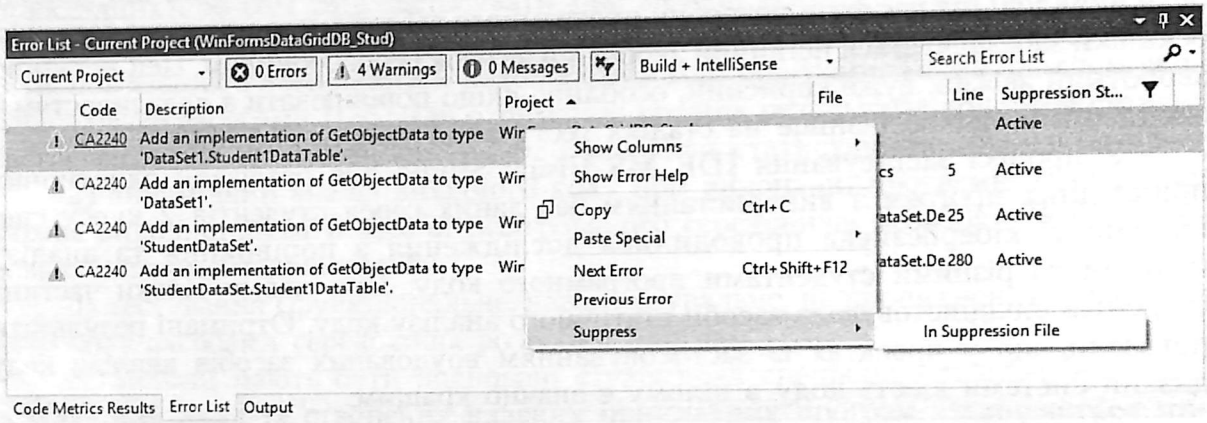

Рис. 2. Результати статичного аналізу коду проекту

Лінійка рішень від компанії IBM для аналізу ПЗ на відповідність вимогам інформаційної безпеки складається з таких продуктів: Security AppScan Standard, Security AppScan Source, Security AppScan Enterprise [11]. Security AppScan Standard – інструмент для автоматичного динамічного аналізу коду ("чорний ящик") та статичного аналізу ("білий ящик"). Застосовується на останніх етапах розробки та на етапі експлуатації програми. Відносно нескладно розгортається та налаштовується. Дозволяє відправляти повідомлення про знайдені вразливості до системи стеження за дефектами коду. IBM Security AppScan Source – інструмент для статичного аналізу коду ("білий ящик"). Він призначений для фахівців з інформаційної безпеки, формує повну картину вразливостей з прив'язкою до вихідного коду. AppScan Source містить засоби інтеграції з багатьма середовищами розробки, що дозволяє відстежувати уразливості на ранніх стадіях. Статичний аналіз підтримує 21 мову програмування. IBM Security AppScan Enterprise – інструмент для централізованого динамічного тестування додатків, обліку та наочного подання ступеня ризику, якому вони піддаються. Дозволяє обчислювати метрики і здійснювати швидке сканування, складати наочні звіти за результатами сканування. За наявності AppScan Source може проводити тестування за методом "прозорого ящика" (Glass Box), зіставляючи результати динамічного і статичного аналізів. AppScan Enterprise містить засоби інтеграції із системою моніторингу безпеки QRadar.

Пропонується використовувати IDE MS Visual Studio разом із Security AppScan Source у курсі підготовки спеціалістів з кібернетичної безпеки. В наступних курсах можна продовжувати вивчення всієї лінійки засобів IBM Security.

До складу Security AppScan Source входять інструменти пошуку й аналізу вразливостей у вихідному коді (Source for Analysis), автоматизації процесу розробки (Source for Automation) та інтеграції з середовищами розробки, такими як MS Visual Studio і Eclipse (Source for Development). Також до пакету входить база знань про уразливості і способи їх усунення (Source Security Knowledgebase)

і засіб для централізованого обміну інформацією про уразливість (Source Enterprise Server). Security AppScan аналізує всі вразливості зі списку OWASP Top 10 [12].

Виявлені уразливості можуть бути класифіковані за такими ознаками:

рівень ризику (високий, середній або низький);

тип вразливості (наприклад, впровадження SQL-коду або переповнення буфера);

файл, де уразливість була виявлена;

інтерфейс програми (виділяється підозрілий виклик API-функції і передані їй аргументи);

метод (функція або метод, звідки був зроблений підозрілий виклик);

розташування (номер рядка і стовпчика в файлі з вихідним кодом, де міститься підозрілий виклик);

класифікація (вразливість або виключення).

Не всі результати пошуку є вразливостями. Класифікація дефектів коду Security AppScan представлена в таблиці 1.

Таблиця 1

Класифікація дефектів коду IBM Security AppScan

Тип дефекту	Опис
Вразливість	Ділянка вихідного коду, що містить помилки, які дають можливість зловмисникові змусити додаток здійснювати незапланові дії, що призводить до несанкціонованого доступу до даних, пошкодження системи тощо.
Виключення типу I	Підозріла ділянка вихідного коду, швидше за все, є вразливістю, при цьому для віднесення її до цього класу не вистачає інформації. Наприклад, можливість проведення атаки може залежати від параметрів використання динамічних елементів або виклику бібліотечних функцій, відомостей про які бракує.
Виключення типу II	Підозріла ділянка вихідного коду, для якої неможливо оцінити ймовірність проведення атаки.

З метою забезпечення повної безпеки програми потрібно проводити додаткові дослідження для віднесення дефектів до вразливостей або до помилкових спрацьовувань. Можна зробити висновок, що поєднання всіх можливостей IDE MS Visual Studio із проведення аналізу вразливостей ПЗ на стадії розробки із застосуванням спеціалізованих засобів аналізу IBM Security AppScan Source та Standart на кінцевій стадії розробки дозволить значно підвищити ефективність, швидкість та якість розробки програм.

У статті проаналізовано та сформульовано вимоги до інструментальних засобів аналізу коду на наявність вразливостей. Представлено практичне застосування методів та комплексу інструментальних засобів розробки безпечного коду, таких як IDE MS Visual Studio та IBM Security AppScan, у курсі підготовки спеціалістів з кібернетичної безпеки. Це дозволить розвивати системний підхід до навчання нових спеціалістів, здатних підвищити якість коду та скоротити кількість помилок і вразливостей ще на етапі проектування та розробки, істотно скоротити витрати на перевірку коду при розробці програм. При такому підході система підготовки спеціалістів з кібернетичної безпеки має забезпечити умови для

отримання практичних навичок з використання сучасних технологій галузі та формування майбутніх спеціалістів, які можуть адаптуватися до стрімких змін технологій та затребувані на ринку праці.

Основним висновком, сформульованим за результатами проведеного дослідження, є необхідність впровадження практик розробки безпечного програмного забезпечення в процеси життєвого циклу розробки. Завдання підвищення захищеності додатків можна вирішувати за допомогою безпечних компіляторів та засобів статичного та динамічного аналізу коду, які в автоматичному режимі допомагають виявляти вразливості і зменшити ризик їх виникнення, наприклад, за рахунок зміни небезпечних функцій на їх безпечні аналоги. Проведений аналіз технологій створення безпечних додатків показав, що необхідно створити добре налагоджений автоматизований процес аналізу коду і на практиці більше уваги приділяти безпеці з боку розробників ПЗ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зелена книга "Регулювання ринку розробки програмного забезпечення". URL: https://cdn.regulation.gov.ua/99/ac/6b/37/regulation.gov.ua_Green%20Book_ Software%20Development%20Market.pdf (дата звернення: 21.09.2017).

2. *Барабанов А.В., Марков А.С., Фадин А.А.* Статистика выявления уязвимостей программного обеспечения при проведении сертификационных испытаний. Вопросы кибербезопасности. 2017. № 2. С. 2–8.

3. *Gallaher M.P. and Kropp B.M.* Economic impacts of inadequate infrastructure for software testing. Technical report, RTI International, National Institute of Standards and Technology, US Dept of Commerce, May 2012.

4. *Аветисян А.И., Белеванцев А.А., Чукляев И.И.* Технологии статического и динамического анализа уязвимостей программного обеспечения. Вопросы кибербезопасности. 2014. № 3(4). С. 20–28.

5. База Common Weakness Enumeration. URL: http://cwe.mitre.org (дата звернення: 10.10.2017).

6. База Common Vulnerabilities and Exposures. URL: http://cve.mitre.org. (дата звернення: 15.08.2017).

7. Статический анализ кода: что могут инструментальные средства? URL: http://www.jetinfo.ru/stati/ staticheskij-analiz-koda-chto-mogut-instrumentalnye-sredstva (дата звернення: 20.11.2017)

8. Анализ качества кода C/C++ с помощью метода анализа кода. URL: https://msdn.microsoft.com/ru-ru/library /dd264897 (v=vs.140).aspx (дата звернення: 02.08.2017).

9. Using Code Analysis with Visual Studio 2017 to Improve Code Quality. URL: https://almvm. azurewebsites.net/labs/tfs/ codeanalysis (дата звернення: 30.11.2017).

10. How we compared code analyzers: CppCat, Cppcheck, PVS-Studio, and Visual Studio. URL: https://www. viva64.com/ en/a/0086 (дата звернення: 14.09.2017).

11. IBM Application security solution. URL: https://www.ibm.com/security/ application-security/appscan (дата звернення: 21.06.2017).

12. OWASP Top 10 - 2017 The Ten Most Critical Web Application Security RisksURL: https://www.owasp.org /images/7 /72/OWASP_Top_10-2017_%28en%29.pdf (дата звернення: 23.10.2017).

13. *Бурячок В.Л., Козачок В.А., Бурячок Л.В.* та ін. Пентестінг як інструмент комплексної оцінки ефективності захисту інформації в розподілених корпоративних мережах. Сучасний захист інформації. 2015. № 3. С. 4–12.

Отримано 20.12.2017
Рецензент Корченко О.Г., д.т.н., проф.