МЕХАНИЗМЫ И СРЕДСТВА ЗАЩИТЫ ИНФОРМАЦИИ

УДК 621. 391:519.2:519.7

БЕЗПЕКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА БЕЗПЕЧНЕ ПРОГРАМУВАННЯ

І.Д. ГОРБЕНКО, О.І. ОЛЕШКО, А.М. ГЕРЦОГ

У роботі розглянуто сучасний стан галузі безпеки програмного забезпечення та безпечного програмування. Наведено огляд стандартів безпечного програмування та засобів автоматизованого контролю безпеки коду. Визначено напрямки подальших досліджень.

Ключові слова: безпечне програмування, статичний аналіз коду, вразливості програмного забезпечення.

ВСТУП

Із ростом ролі сучасних інформаційно-телекомунікаційних систем (ІТС) у сфері державного управління та роботі комерційних структур пропорційно збільшується складність таких систем і, як наслідок, кількість помилок у їх програмному забезпеченні. Як правило, основні інструменти захисту ІТС міжмережні екрани, системи виявлення або попередження вторгнень та антивірусні системи на основі аналізу сигнатур трафіку спрямовані на відбиття атак з мережі [1]. Інструменти, які використовують евристичний аналіз для виявлення невідомих раніше атак, на даний момент не здобули широкого застосування із-за низької ефективності. Проте, зазначені механізми не забезпечують захисту від використання помилок у програмному забезпеченні для створення вихідної точки атаки на ІТС. Помилки, знайдені зловмисником, за певних доробок можуть перетворитися на вразливості і призвести до компрометації ІТС.

Експлуатація вразливостей може мати різні наслідки:

– Отримання доступу до конфіденційної інформації неавторизованими користувачами.

– Порушення нормального режиму функціонування.

– Виконання довільного коду.

– Виконання команд з підвищеними привілеями користувачем, який їх не має.

За даними щорічного звіту центру скарг на злочини у всесвітній мережі Інтернет (Internet Crime Compliant Center) (США) збитки комерційних структур внаслідок кіберзлочинів склали 559,7 млн. доларів [2]. З них на злочини, пов'язані з використанням вразливостей програмного забезпечення, припадає: викрадення конфіденційних даних 14,1%, шахрайство з кредитними картками 10,4%, хакерські атаки 7,9%.

Зважаючи на наведені вище матеріали та аналізуючи сучасний стан галузі, можна стверджувати, що наведені тенденції будуть справедливі і надалі, тому актуальними напрямами досліджень у сфері захисту ІТС є методики та інструменти пошуку вразливостей програмного забезпечення. Існує ряд спеціалізованих засобів, які доступні користувачам для боротьби з використанням стандартних типів вразливостей. Проте, як правило, вирішення проблеми експлуатації вразливостей програмного забезпечення покладається на його розробника.

Причиною появи вразливостей у програмному забезпеченні автори [3] називають бажання компаній, які займаються створенням програмного продукту, захопити максимально широкі ринки збуту. Вважається, що більший комерційний успіх матиме продукт, який першим з'явився на ринку та має ширший набір функціональних можливостей. Працюючи за таким принципом, програма створюється за мінімальний проміжок часу, а ресурси та час на тестування виділяються у недостатній кількості, тобто виконується лише множина стандартних тестів, спрямованих на виявлення типових помилок при обробці вхідних даних. Розробники обирають тактику, за якою до процесу виявлення помилок залучаються кінцеві користувачі, про які вони сповіщають розробника за допомогою звітів про помилки.

Слід зазначити, що наявність помилок притаманна не тільки програмному забезпеченню, створеному великими корпораціями. Вільне програмне забезпечення з відкритим вихідним кодом (free open source software), яке не немає на меті отримання прибутку та здобуття комерційного успіху, незважаючи на підтримку численної спільноти також не позбавлене помилок. Так, у вільній та відкритій операційній системі Ubuntu Linux 10.04 з моменту її виходу у квітні поточного року спільнотою було знайдено 159 помилок (станом на 16.10.2010) [5].

Для боротьби з помилками в програмному забезпеченні існує декілька методик. Окремо слід відзначити покращення механізмів розробки та архітектури апаратного забезпечення. Прикладами таких технологій є механізм StackGuard компілятору GCC (шляхом внесення збитковості на етапі виконання перевіряється стан стеку), режим безпеки /GS компілятору виробництва Microsoft,

технологія процесорів Intel Execute disable bit (забороняє виконання команд у сегменті стеку та сегменті тексту), механізми безпеки JVM (Java virtual machine) та CLR (common language runtime). Для безпосередніх розробників доступними є дві методики пошуку помилок у програмному забезпеченні з протилежними ідеологіями: статичний аналіз вихідного коду або динамічний аналіз програми під час виконання.

Метою даної роботи є дослідження стану галузі пошуку вразливостей програмного забезпечення та безпечного програмування, огляд існуючих методик та інструментів, визначення перспективних напрямів подальших досліджень.

1. БЕЗПЕЧНЕ ПРОГРАМУВАННЯ

Більшість помилок, яких припускаються програмісти, є типовими та зумовлені здебільшого особливостями мови програмування, якою виконується реалізація. Таке спостереження дозволяє зробити припущення, що для підвищення ефективності пошуку вразливості в першу чергу необхідно перевіряти множину «стандартних» вразливих місць[1,2].

Таке спостереження привело до появи нового напрямку досліджень у галузі розробки програмного забезпечення – безпечного програмування. Результати досліджень наведені в роботах [7, 8]. Предметом досліджень стали популярні мови програмування, за допомогою яких створюються великі та складні програмні продукти C, C++, Java, C# та ін. За результатами досліджень складено набори класифікованих множин вразливостей, які притаманні мовам програмування. Метою нової дисципліни є розробка прийомів програмування, використання яких попереджає виникнення помилок.

Існує декілька систем класифікацій вразливостей. Автор [7] розглядає мови програмування C та C++ і пропонує проводити класифікацію вразливостей за завданнями, в ході вирішені яких вони виникають. Іншим можливим методом класифікації є класифікація за помилками, допущеними під час програмування [8]. Розглянемо ситуації, які найчастіше стають причинами появи вразливостей.

Однією з найвідоміших помилок є помилка переповнення буфера. Причина помилки – невідповідність об'єму вхідних даних розміру буфера, призначеного для їх збереження. Таким чином, при спробі збереження вхідних даних виконується запис даних у ділянки пам'яті, які містять службові дані програми. Помилка стає можливою, коли програміст не виконує перевірку відповідності об'єму вхідних даних та розміру буферу для збереження. Переповнення буфера було використано Морісом для створенням першого мережного черв'яка, що спричинив масштабне зараження комп'ютерів.

Другою за популярністю помилкою, що призводить до появи вразливостей, є робота з числами. У багатьох сучасних мовах програмування не існує автоматичного контролю за ситуаціями, коли значення числової змінної виходить за межі значень, які можуть зберігатись у змінній цього типу. У випадку отримання контролю зловмисником над значенням змінної, можливо використати її для виділення помилкового об'єму пам'яті та подальшого переповнення буфера.

Помилки при роботі з форматним вводом та виводом ще один поширений вид помилок. Помилки виникають внаслідок некоректної роботи з даними, які мають бути оброблені згідно із заданими параметрами форматування. Можливими наслідками використання помилки є виконання зловмисником непередбачених дій з даними в процесі приведення до заданого формату.

Некоректна обробка помилок може стати причиною вразливостей програмного продукту. В ході роботи будь-якої програми можуть виникати помилкові ситуації, які потребують коректної обробки (вивільнення ресурсів, зміна порядку виконання програми). Внаслідок некоректних дій з боку програміста, при обробці помилок зловмисник має можливість змінити хід виконання програми або запустити на виконання власний код.

Стан перегонів є ще однією типовою помилкою, яка виникає у сучасних програмах. При створенні багатопроцесних або багатопотокових додатків виникає необхідність синхронізації усіх потоків виконання з метою забезпечення взаємовиключаючого доступу до сумісних ресурсів. В разі невиконання цієї вимоги виникає ситуація перегонів, коли стан ресурсу залежить від того, який з потоків виконанням першим отримає доступ до нього. Найбільш поширеними наслідками цієї помилки є подвійне вивільнення пам'яті або навпаки подвійний запис у пам'ять внаслідок чого частина даних втрачається, або замінюється на код зловмисника.

Наведені помилки програмування є найбільш поширеними та не описують усі можливі джерела вразливостей. Автори [7-8] наводять більш повний перелік можливих помилок програмування.

2. СТАНДАРТИЗАЦІЯ У ГАЛУЗІ БЕЗПЕЧНОГО ПРОГРАМУВАННЯ

Дослідження у галузі безпечного програмування проводилися в тому числі і в центрі реагування на комп'ютерні злочини (Computer emergency response team – CERT). За результатами роботи було створено стандарт з безпечного програмування мовою C [9]. Наразі триває робота зі створення стандарту безпечного програмування мовами C++ та Java.

Стандарт складається з переліку правил, обов'язкових для виконання та рекомендацій, які є бажаними. Цілі створення стандарту безпечного програмування наступні:

• впровадження принципів безпечного програмування серед компаній-розробників програмного забезпечення:

– допускається розширення стандарту правилами, специфічними для компанії;

– не припустиме виключення чи заміна правил;

- навчання спеціалістів;
- сертифікація програмістів;
- визначення вимог до інструментів аналізу готового продукту;
- сертифікація програмних систем.

Правилами вважаються прийоми програмування, які відповідають наступним вимогам:

– нехтування правилом може призвести до появи помилки, яка може бути використана для створення вразливості;

– є певна множина випадків, у яких порушення прийому програмування необхідне для забезпечення правильного функціонування системи;

– відповідність правилу може бути визначена за допомогою автоматизованого аналізу, формальних методів чи неавтоматизованих способів перевірки.

Рекомендаціями вважаються прийоми програмування, які:

– дозволяють підвищити рівень безпеки;

– відповідають не всім вимогам, які висуваються до правила.

Кожна рекомендація або правило мають унікальний ідентифікатор в межах стандарту, який складається з двох цифр (00-99) та однієї літери (A або C). Розробка стандарту є відкритою і до неї долучаються усі бажаючі, які займаються розробкою програмних продуктів певною мовою чи подальшою розробкою самої мови програмування.

3. АВТОМАТИЗОВАНІ ЗАСОБИ СТАТИЧНОГО АНАЛІЗУ КОДУ

Процес навчання та сертифікації спеціалістів з безпечного програмування потребує певних витрат часу, а пропускна здатність центрів з підготовки набагато нижча, ніж попит на програмістів. І все ж, навіть сертифікований та підготовлений програміст може припуститися помилки, яка може стати вразливістю. Організувати повну перевірку вихідного коду на наявність помилок навіть за наявністю стандарту безпечного програмування можливо лише з використанням автоматизованих засобів дослідження вихідного коду.

Дослідження вихідного коду за допомогою автоматизованих засобів отримало назву статичного аналізу. Його характерними особливостями є:

– можливість дослідження всього програмного продукту;

– відсутність потреби у виконанні програми;

– виявлення вразливих місць системи з деякою вірогідністю.

На даний момент існує велика кількість статичних аналізаторів для більшості популярних мов програмування як комерційних, так і вільних з відкритим вихідним кодом.

У своїх навчальних посібниках фахівці CERT у якості інструменту статичного аналізу коду рекомендують використовувати додаток під назвою Compass, який базується на програмному продукті ROSE.

ROSE – компіляторна інфраструктура з відкритим вихідним кодом призначена для виконання перетворень: вихідний код – вихідний код та аналізу широкого кола додатків [10]. ROSE може використовуватись для створення інструментів статичного аналізу коду, оптимізації програм, аналізу швидкодії та безпеки коду. Проміжне представлення, яке використовується в ROSE, добре підходить для побудови абстрактного дереву синтаксису. За ним виконується пошук сигнатур аналізаторами та оптимізація виконуваного коду. Інтерфейс програмного забезпечення розроблений таким чином, що його використання користувачами зводиться до викликів функцій бібліотек. Для аналізу вихідного коду доступні наступні дії: побудова графу викликів, аналіз потоку виконання, аналіз потоку даних, аналіз ієрархії класів, аналіз залежностей даних та аналіз шаблонів взаємодії. ROSE розповсюджується під ліцензією типу BSD та портована на операційні системи Linux та Mac OS X.

Ще одним інструментом для проведення статичного аналізу коду є бібліотека VivaCore розроблена компанією «Системы программной верификации» (Росія) на базі бібліотеки OpenC++ [11]. Ідея створення бібліотеки виникла під час роботи над статичним аналізатором коду Viva64, призначеного для пошуку помилок при перенесенні додатків Windows з 32 бітної на 64 бітну платформу. VivaCore – бібліотека для роботи з C та C++ кодом, призначена для реалізації на її основі систем рефакторингу коду, систем статичного та динамічного аналізу коду, систем трансформації та оптимізації коду, розширень мови програмування, створення підсистеми підсвічування синтаксису, систем документування вихідного коду. Бібліотека має наступні функціональні блоки:

– підсистема вводу даних;
– підсистема попередньої обробки коду;
– лексичний аналізатор;
– граматичний аналізатор;
– клас обходу дерева розбору;
– засоби мета програмування;
– засоби збереження результатів.

VivaCore розповсюджується під ліцензією, що дозволяє її вільне використання як у приватних, так і в комерційних цілях, вільну переробку чи доробку та не вимагає грошових відрахування на користь авторів. Варто відзначити, що бібліотека має основну властивість, необхідну для успішного розвитку відкритого та вільного продукту – підтримку розробника.

Використання наведених програмних продуктів аналізу коду потребує створення формалізованого переліку правил перевірки вихідного коду – шаблонів пошуку. Виконавши розбір

вихідних текстів програми у дерево абстрактного синтаксису або дерево розбору і застосувавши до нього реалізовані попередньо шаблони правил безпечного програмування стає можливим виявлення потенційних вразливостей.

Використання статичних аналізаторів дозволяє підвищити захищеність коду від атак з боку зловмисника. Проте, разом із перевагами цьому підходу притаманна певна множина недоліків. Нижче наведено порівняльну таблицю переваг та недоліків статичного аналізу коду.

Таблиця 1

Переваги	Недоліки
Повне покриття вихідного коду програми	Наявність помилок першого роду
Менші витрати часу у порівнянні з дослідженням шляхом виконання програми	Обов'язкова наявність вихідного коду
Можливість автоматизації	Залежність від мови програмування, якою написано вихідний код
Можливість використання на всіх етапах створення програмного продукту	Не здатен виявляти помилки часу виконання

ВИСНОВКИ

Широке розповсюдження інформаційно-телекомунікаційних систем вимагає перегляду дій, що спрямовані на забезпечення їх безпеки. Постійно зростаюча складність програмного забезпечення підвищує вірогідність появи вразливостей, які можуть бути використані для отримання несанкціонованого доступу до інформації. Реакцією на виклики безпеці програмного забезпечення стала поява нового напрямку досліджень, спрямованого на пошук потенційних вразливостей на етапі розробки програмного продукту. Безпечне програмування пропонує створення переліку правил програмування, які попереджають виникнення потенційних вразливостей. Об'єднання досягнень у галузі програмування, а саме створення стандартів безпечного програмування, дозволило створити потужні методики пошуку та діагностики вразливостей. Перелік правил безпечного програмування дозволяє створити формалізований набір правил для створення сигнатур пошуку статичного аналізатора. Наведені в роботі властивості статичних аналізаторів та постійно зростаючий інтерес до цієї галузі дозволяє прогнозувати подальший розвиток даного напрямку. Серед перспективних напрямків подальших досліджень можна виділити наступні:

– підвищення якості виявлення помилок, тобто зменшення кількості помилок 1-го роду;

– створення бібліотек для безпечної роботи програмного забезпечення;

– розширення набору правил та рекомендацій з безпечного програмування.

Література:

[1] Hacking exposed web applications / J. Scambray, M. Shema ;McGraw-Hill – C. : Berkley, 2002. – 416c. – ISBN 0-07-222438-X

[2] 2010 IC3 Annual Report / Internet crime compliant center. – Режим доступу : http://www.ic3.gov/media/annualreports.aspx

[3] Взлом программного обеспечения, анализ и использование кода / Г. Хогланд, Г. Мак-Гроу ; Вильямс – М., 2005. – 400с. – ISBN 5-8459-0785-3

[4] The Physicist / www.wired.com. – Режим доступу : http://www.wired.com/wired/archive/3.09/myhrvold.html

[5] Vulnerability Report: Ubuntu Linux 10.04 / www.secunia.com. – Режим доступу : https://secunia.com/advisories/product/30524/

[6] Writing solid code / S. McGuire. –R. : Microsoft Press, 1993. – 247p. – ISBN 1-55615-551-4.

[7] Secure coding in C and C++ / R. Seacord. – Addison Wisley press, 2005. – 386 p. – ISBN 0-321-33572-4.

[8] *Howard, M.* 24 deadly sins of software security / M. Howard, D. LeBlanc, J. Viega. – 2-nd edition. – N.Y. : McGrawHill, 2010. – 443 p. – ISBN 978-0-07-162676-7.

[9] CERT C Secure Coding Standard – Введ. 2009. – Addison Wesley, 2009. – 720 p.

[10] ROSE project main page / ROSE compiler project. – Режим доступу: http://rosecompiler.org - 11.05.11 р. – Загл. з экрану.

[11] Сущность библиотеки анализа кодаVivaCore / Static code analyzer for C/C++ and 64x migration. – Режим доступу : http://www.viva64.com/ru/a/0013/ – 11.05.11 р. – Загол. з екрану.

[12] Константин Книжник: статический анализ, взгляд со стороны / Static code analyzer for C/C++ and 64x migration. – http://www.viva64.com/ru/a/0034/ – 11.05.11 р. – Загол. з екрану.

Надійшла до редколегії 7.04.2011

Горбенко Іван Дмитрович, доктор технічних наук, професор, завідувач кафедри БІТ ХНУРЕ. Область наукових інтересів: криптографія, захист інформації в ІТС.

Олешко Олег Іванович, кандидат технічних наук, доцент кафедри БІТ ХНУРЕ. Область наукових інтересів: безпечне програмування, захист інформації в ІТС.

Герцог Андрій Миколайович, аспірант кафедри БІТ ХНУРЕ. Область наукових інтересів: безпечне програмування, захист інформації в ІТС.

УДК 621. 391:519.2:519.7

Безопасность программного обеспечения и безопасное программирование / И.Д. Горбенко, О.И. Олешко, А.Н. Герцог // Прикладная радиоэлектроника: науч.-техн. журнал. – 2011. Том 10. № 2. – С. 244–248.

В работе рассмотрено современное состояние отрасли безопасности программного обеспечения и безопасного программирования. Приведен обзор стандартов безопасного программирования и средств автоматизированного контроля безопасности кода. Определены направления дальнейших исследований.

Ключевые слова: безопасное программирование, статический анализ кода, уязвимости программного обеспечения.

Табл. 1. Библиогр.: 10 назв.

UDC 621. 391:519.2:519.7

Software security and secure programming / I.D. Gorbenko, O.I. Oleshko, A.M. Gertsog // Applied Radio Electronics: Sci. Journ. – 2011. Vol. 10. № 2. – P. 244–248.

The modern state of software security and secure coding is considered. A secure coding standards and code security automatized control tools overview is provided. Directions of further research are determined.

Keywords: secure programming, static code analysis, software vulnerabilities.

Tab. 1. Ref.: 10 items.