DOI https://doi.org/10.32782/2521-6643-2022.1-63.1
УДК 004.75

Зінченко А. Ю., кандидат технічних наук,
доцент, завідувач кафедри комп'ютерних наук
Київського міжнародного університету
ORCID: 0000-0003-1586-3645

ПРОЕКТУВАННЯ РОЗПОДІЛЕНИХ ІНФОРМАЦІЙНИХ СИСТЕМ НА ОСНОВІ ВИКОРИСТАННЯ ТЕХНОЛОГІЇ СЛАБОЗВ'ЯЗАНИХ КОМПОНЕНТІВ

У даній роботі розглядається побудова архітектурних рішень на основі технології слабозв'язаних програмних модулів автоматизованих інформаційних систем підприємств та установ. Детально проаналізовано використання принципу інверсії залежностей у реалізації сервіс-орієнтованої архітектури, побудови веб-сервісів (SOAP і REST), а також архітектурної моделі хмарних обчислень SaaS. Зокрема, детально описано використання інверсії управління через ін'єкції залежностей (Constructor injection, Parameter injection, Setter injection, Interface injection) при проектуванні програмних модулів інформаційної технології. Для підвищення надійності захисту персональних даних при проектуванні архітектурних рішень розглянуто основні механізми несанкціонованого доступу в сервіс-орієнтованих системах, а також шляхи їх подолання.

Ключові слова: *IoC, DI, DIP, SOA, SaaS, ESB.*

Zinchenko A. Yu. Design of distributed information systems based on using the technology of loosely coupled components

This paper is devoted to the application of the technology of loosely coupled software components to the design of distributed information systems. By focusing on commonly asked questions, this paper provides techniques to help you discover and weigh the trade-offs as you confront the issues you, as a programmer, face as an architect. It analyzes in detail the use of the principle of dependency inversion for the implementation of service-oriented architecture, for the construction of web services (SOAP and REST), as well as for the architectural model of SaaS cloud computing. At the same time, the main method of reducing the connectivity of software modules is the use of Inversion of Control through the implementation of the dependency injection mechanism. This provides the implementation of the main principle of SOLID – The Dependency Inversion Principle.

The main types of relationships between classes for the design of a distributed architecture are considered in detail in the work. These are inheritance, implementation, aggregation, and composition. Since inheritance is a rigid type of coupling, in which all the functionality of the inherited class is defined at the compilation stage, and the programmer cannot dynamically redefine it during the execution of the program, so to ensure loosely coupled software components (in particular, through the mechanism of dependency injection) instead of inheritance, you need to use composition, and instead of use composition, if possible, use aggregation. Also, the paper describes in detail the use of control inversion through dependency injection (Constructor injection, Parameter injection, Setter injection, Interface injection) when designing software modules of information technology.

In addition, to increase the reliability of personal data protection when designing architectural solutions, the main mechanisms of unauthorized access (hacker attacks) in service-oriented systems are considered. This is because the received URLs, cookies and data in HTTP requests may be spoofed. Classic types of hacker attacks are analyzed in detail in work: permanent and passive cross-site scripting, SQL injections, and cross-site query forgery. In addition, methods of combating such attacks at both the software and network levels are provided.

Key words: *IoC, DI, DIP, SOA, SaaS, ESB.*

Постановка проблеми. Сьогодні все більше організацій переходять від клієнт-серверної до сервіс-орієнтованої архітектури, зокрема, використовуючи сервіси і гібридні хмарні технології, побудовані відповідно до протоколів SOAP, XML-RPC і JSON-RPC, або ж з використанням архітектурного стилю REST. Крім того, за останні роки в Європі було запроваджено багато порталальних рішень, у тому числі на рівні державних адміністрацій (наприклад, Європейський портал даних – загальноєвропейське сховище інформації державного сектора (https://data.europa.eu/en), Єдиний портал державних послуг України (https://diia.gov.ua/), відкрита платформа публічних даних Франції (https://www.data.gouv.fr/fr/), портал відкритих даних Італійського Державного Управління (https://www.dati.gov.it/) тощо), у яких безпека стає одним із ключових питань, що впливають на розгортання програмного забезпечення. Доступ до конфіденційної інформації компанії, зокрема персональних даних, здійснюється за допомогою сервісів, розгорнутих на розподілених компонентах сервіс-орієнтованої архітектури (SOA) за допомогою API. Тому питання безпеки стали частиною процесу прийняття рішень підприємствами щодо впровадження SOA.

Така ж тенденція спостерігається для CRM-систем (системи управління взаємовідносинами з клієнтами) і систем ERP (системи планування ресурсів підприємства), які, як правило, проєктувалися в двочи трьохланковій клієнт-серверній архітектурі 10-20 років тому. Хоча серед них сервіси застосувань мали

© А. Ю. Зінченко, 2022

одиниці. В теперішній час для того, щоб ефективно керувати, аналізувати та покращувати відносини з клієнтами, інформаційні технології таких автоматизованих систем повинні мати комплексний набір хмарних рішень, які можуть підтримувати та супроводжувати користувача на кожному етапі процесу потоку робіт (“Workflow”) – програмного графічного модулю автоматизованою інформаційної системи (АІС) для управління процесами і завданнями, які повторюються і виконуються у певному порядку. Таке комплексне рішення повинно включати в себе хмарні рішення для продажу, обслуговування, комерції, маркетингу, а також платформу клієнтських даних (CDP) із підтримкою штучного інтелекту, яка може оперувати онлайн, офлайн та сторонніми джерелами даних для повного та динамічного представлення їх клієнту. Серед останніх найпоширенішою моделлю хмарних рішень є SaaS (software as a service), доступ до програмного забезпечення якого надається віддалено по мережевим каналам через веб-інтерфейс чи термінал.

Зважаючи на те, що зв’язаність програмного забезпечення є метрикою програмного забезпечення та одним з основних принципів, який закладений в основу об’єктно-орієнтованого програмування (ООП), то побудова слабозв’язаних системних модулів при проєктуванні розподілених інформаційних систем, зокрема сервіс-орієнтованих, з метою покращення захисту персональних даних є темою актуальною.

Аналіз останніх досліджень і публікацій. Основні принципи зв’язаності та пов’язаності програмного забезпечення були закладені Ларрі Константином на прикінці 1960-х років як частина структурованого дизайну для зменшення витрат на обслуговування та модифікацію [1; 2].

Основний принцип об’єктно-орієнтованого конструювання The Dependency Inversion Principle (DIP) – детально проаналізований в [3], а також в класичній книжці Еріха Ґ., Річарда Г. Ральфа Д. та Джона В. «Шаблони проєктування: елементи повторно використовуваного об’єктно-орієнтованого програмного забезпечення» [4].

Шаблони архітектури корпоративних додатків написані як пряма відповідь на складні виклики, з якими стикаються розробники корпоративних додатків. Із сучасними поглядами на проєктування розподілених архітектурних рішень (від Smalltalk до CORBA, Java та .NET) можна ознайомитися в книзі відомого об’єктно-орієнтованого дизайнера Мартіна Фаулера [5]. Особливості проєктування високорозподілених архітектур, від керування розподіленими транзакціями до оптимізації операційних характеристик (таких як масштабованість, еластичність і продуктивність) для різних принципів програмування (імперативного, декларативного, включаючи функціонально-орієнтоване, конкатенативного та інших) добре описані в книгах [6; 7].

Мета статті – розглянути реалізацію технології слабозв’язананих компонентів програмного забезпечення (ПЗ) на основі інверсії управління (IoC), тобто реалізацію механізму Dependency Injection (DI), з метою проєктування розподілених інформаційних систем покращеного захисту персональних даних.

Виклад основного матеріалу. Реалізація IoC в розподілених інформаційних системах. В програмній інженерії міра звя’заності характеризує взаємозалежність між програмними модулями [8]. Низька зв’язаність програмного коду є ознакою добре структурованої і добре спроєктованої автоматизованої системи. Вона часто корелює з високою пов’язаністю (cohesion), та навпаки. Якщо зв’язаність програмного забезпечення відноситься до взаємозалежностей між модулями, то пов’язаність описує, наскільки пов’язані функції в межах одного модуля. Низька зв’язаність означає, що даний модуль виконує завдання, які не дуже пов’язані один з одним, і, отже, може створити проблеми, коли модуль стає великим. Жорстка пов’язаність є серйозним недоліком програмного забезпечення, оскільки утрудняє розуміння логіки модулів, їх модифікацію, автономне тестування, і навіть повторне використання. Слабка зв’язаність є основним принципом ООП SOLID – основою DIP, а також одним із 9 шаблонів об’єктно-орієнтованого проєктування GRASP Крейга Лармана [3] для вирішення загальних завдань щодо призначення відповідальності класам та об’єктам.

Існують різні методи зменшення зв’язаності програмних модулів. Вони втілені в шаблонах проєктування. Серед останніх можна виділити такі шаблони проєктування багатошарової архітектури як Model-View-Controller, Model-View-Presenter, Model-View-ViewModel та інші. Одним із підходів до зменшення зв’язаності є функціональний дизайн, який прагне обмежити обов’язки модулів з функціональністю. Одним із ключових методів реалізації технології слабо зв’язананих компонентів програмного забезпечення є використання інверсії управління, тобто реалізації механізму DI.

Перш ніж перейти до реалізації принципу інверсії управління в розподілених АІС, слід глибоко розуміти принципи об’єктно-орієнтованого проєктування: різницю між IoC, DI та DIP, між патернами та фреймворками, бібліотеками, залежностями, між проєктуванням взаємозв’язків між класами. Розглянемо коротко останні, тобто основні відношення між об’єктами класів, що допоможуть нам зрозуміти зв’язок між сутностями під час їх використання в патернах проєктування. Виділяють кілька основних видів взаємозв’язку: наслідування, реалізація, асоціація, композиція та агрегація.

Спадкування є базовим принципом ООП і дозволяє одному класу (спадкоємцю) успадкувати функціонал іншого класу (батьківського). Нерідко відносини успадкування ще називають генералізацією чи узагальненням. Наслідування визначає відношення “IS A”.

Реалізація передбачає визначення інтерфейсу та його реалізації у класах. Згідно з принципом поділу інтерфейсів (The Interface Segregation Principle, ISP) при створенні системи взаємозв’язків між класами потрібно програмувати лише на рівні інтерфейсів, а не їх конкретних реалізацій. Тобто при проєктуванні архітектури інформаційної технології програмісти не повинні примусово впроваджувати інтерфейси там, де вони не потрібні. Під інтерфейсами тут слід розуміти не тільки типи мови програмування, визначені з допомогою ключового слова interface, а й визначення функціоналу без його конкретної реалізації.

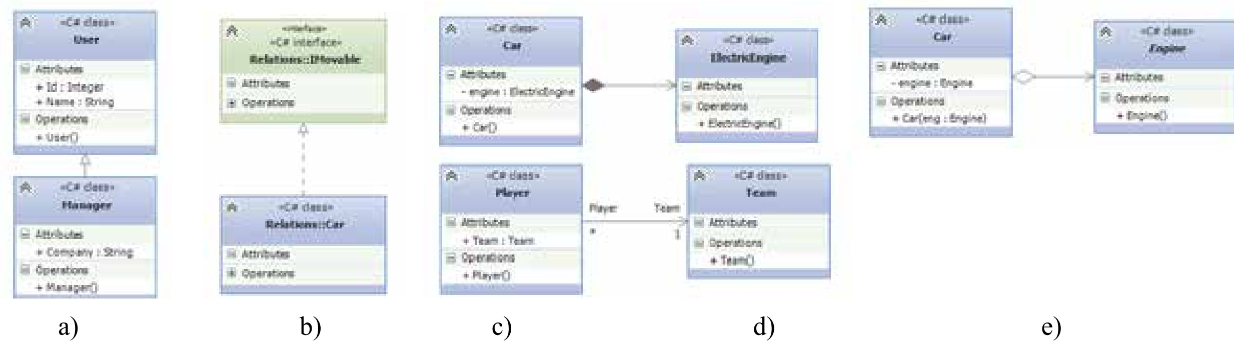

Рис. 1. Відношення між класами та об’єктами:
наслідування (a), реалізація (b), композиція (c), асоціація (d) та агрегація (e)

Асоціація – це відношення, у якому об’єкти одного типу певним чином пов’язані з об’єктами іншого типу. Наприклад, об’єкт одного типу містить або використовує об’єкт іншого типу. При цьому агрегація та композиція є окремими випадками асоціації, і їх характеризує відношення “HAS A”. Різниця полягає в тому, що при композиції реалізовується жорстка зв’язаність: в конструкторі класу створюється новий екземпляр класу, який повністю керує життєвим циклом об’єкта даного класу. При агрегації ж реалізується слабка зв’язаність, тобто переданий через конструктор об’єкт є рівноправним до об’єкту класу конструктора, в який його передали. В конструктор передається посилання на вже наявний об’єкт, під який вже виділено пам’ять. І, зазвичай, передається посилання не на конкретний клас, а на абстрактний клас чи інтерфейс, що збільшує гнучкість програми. Крім того, при успадкуванні весь функціонал класу-спадкоємця жорстко визначено на етапі компіляції, тобто під час виконання програми неможливо його динамічно перевизначити. Саме тому замість успадкування слід віддавати перевагу композиції, а замість композиції – агрегації як одному з підходів до зменшення зв’язаності через об’єктно-орієнтоване проєктування.

IoC – це досить загальне поняття, яке відрізняє бібліотеку динамічної компановки від фреймворка. На відміну від традиційного потоку управління IoC інвертує управління потоком. В класичній моделі архітектури програмного забезпечення код, який викликається, сам контролює зовнішнє оточення, а також час і порядок викликів бібліотечних методів. Проте у випадку з фреймворком обов’язки міняються місцями: фреймворк надає деякі точки розширення, через які він викликає певні методи користувацького коду.

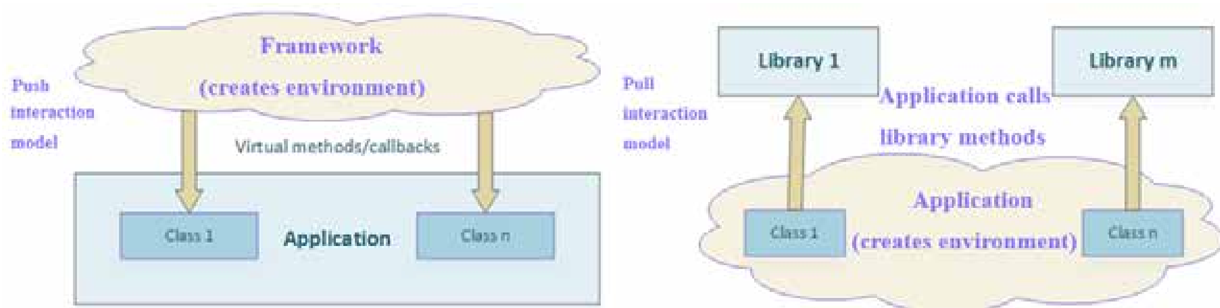

Рис. 2. Інвертація потоку управління інверсією управління [9]

IoC використовується для підвищення модульності програми і робить її розширюваною [10]. Вперше термін був використаний Майклом Меттссоном в праці [11]. Зворотні виклики, планувальники, цикли подій, ін’єкція залежностей, шаблонний метод і паттерн «Наглядач» є прикладами шаблонів проєктування, в яких використовується принцип інверсії керування.

Основними технологіями реалізації IoC є using a service locator pattern, using dependency injection (Constructor injection, Parameter injection, Setter injection, Interface injection), Using a contextualized lookup, Using the template method design pattern та Using the strategy design pattern.

Розглянемо DIP – основний принцип SOLID, який дає рекомендації, якими повинні бути залежності між високорівневими та низькорівневими абстракціями:

– модулі верхніх рівнів не повинні залежати від модулів нижніх рівнів. При цьому обидва типу модулів повинні залежати від абстракцій;

– абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій;

– модулі вищого рівня реалізують бізнес-правила або логіку в системі. Низькорівневі модулі виконують більш детальні операції, наприклад, запис інформації в базу даних, передачу повідомлень операційній системі або службам тощо.

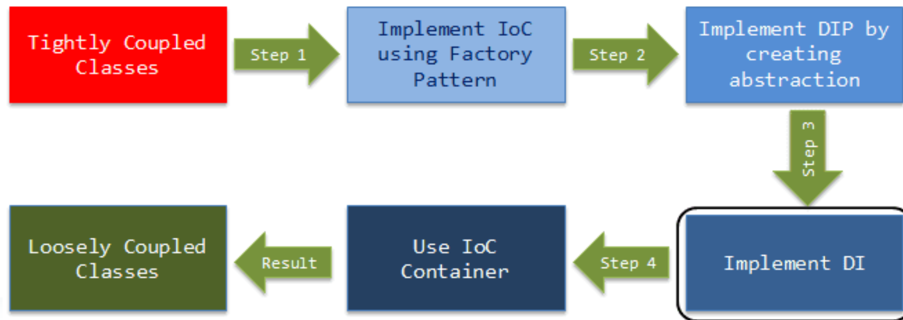

Рис. 3. Приклад реалізації IoC, використовуючи шаблонний метод [9]

При цьому залежності класу повинні розташовуватись на поточному або вищому рівні абстракції. Іншими словами, не будь-який клас, який вимагає інтерфейс у конструкторі, слідує принципу інверсії залежностей. Розглянемо приклад:

```
class ReportProcessor
{
 private readonly ISocket _socket;
 public ReportProcessor(ISocket socket)
 {
 _socket = socket;
 }
 public void SendReport(Report report, IStringBuilder stringBuilder)
 {
 stringBuilder.AppendFormat(CreateHeader(report));
 stringBuilder.AppendFormat(CreateBody(report));
 stringBuilder.AppendFormat(CreateFooter(report));
 _socket.Connect();
 _socket.Send(ConvertToByteArray(stringBuilder));
 }
}
```

Клас ReportProcessor все ще приймає «абстракцію» в аргументах конструктора – ISocket, але ця «абстракція» знаходиться на кілька рівнів нижче за рівень формування та відправлення звітів. Аналогічно справи і з аргументом методу SendReport: «абстракція» IStringBuilder не відповідає принципу інверсії залежностей, оскільки оперує більш низькорівневими поняттями, ніж потрібно. На цьому рівні слід оперувати не рядками, а звітами. Як результат, цей код не слідує принципу DIP – в даному прикладі використовується DI.

DI – основний механізм реалізації IoC – передачі класу його залежностей, слугує фреймворком. Як зазначалося вище, існує декілька конкретних способів або патернів застосування залежностей. Розглянемо приклад [9]:

```
class ReportProcessor
{
 private readonly IReportSender _reportSender;
 // Constuctor Injection: passing a mandatory dependency
 public ReportProcessor(IReportSender reportSender)
 {
 _reportSender = reportSender;
 Logger = LogManager.DefaultLogger;
 }
 // Method Injection: passing mandatory method dependencies
 public void SendReport(Report report, IReportFormatter formatter)
 {
 Logger.Info("Sending report...");
 var formattedReport = formatter.Format(report);
 _reportSender.SendReport(formattedReport);
 Logger.Info("Report has been sent");
 }
 // Property Injection: installing optional "infrastructure" dependencies
 public ILogger Logger { get; set; }
}
public Store store()
{
```

```
 Store store = new Store();
 store.setItem(item1());
 return store;
}
<bean id = "store" class= "org.baeldung.store.Store" >
 <property name = "item" ref= "item1" />
</bean>
```

Різні види ін'єкції залежностей призначені для вирішення конкретних завдань. Через конструктор передаються обов'язкові залежності класу, без яких робота класу неможлива (IReportSender – обов'язкова залежність класу ReportProcessor). Через метод передаються залежності, які потрібні лише одному методу, а не всім методам класу (IReportFormatter необхідний лише методу надсилання звіту, а не класу ReportProcessor повністю). Через властивості повинні встановлюватися лише необов'язкові залежності (зазвичай, інфраструктурні), для яких існує значення за замовченням (властивість Logger містить розумне значення за замовченням, яке пізніше може бути замінено). Для DI на основі сетера IoC-контейнер буде викликати методи сетера класу Store після виклику конструктора без аргументів або статичного фабричного методу без аргументів для створення екземпляра bean.

Проєктування SOA в розподілених інформаційних технологіях. Більшість підприємств протягом багатьох років вкладали значні кошти у системні ресурси. Такі підприємства мають величезну кількість даних, що зберігаються у застарілих корпоративних інформаційних системах (ERP- і CRM-системах), тому відмовлятися від існуючих систем є недоцільним. Вигідніше розвивати та покращувати існуючі автоматизовані інформаційні системи. Цю проблему і вирішує сервіс-орієнтована архітектура.

SOA визначається різними способами. Один із них – це «парадигма організації та використання розподілених можливостей, які можуть бути під контролем різних доменів» [12]. Інакше кажучи: SOA – це архітектурний стиль для створення програмного забезпечення, за допомогою якого різні типи сервісів (компонентів інформаційної технології для надання інформаційних послуг) можуть незалежно взаємодіяти один з одним. Основою SOA є використання технології слабо зв'язаних компонентів, яка сприяє слабкому зв'язку між віддаленими за стандартизованими протоколами використання, і розподіленими програмними компонентами, щоб їх можна було використати повторно. SOA використовує парадигму «знайти-зв'язати-виконати».

SOA не є новою концепцією. Так компанія Sun визначила основні принципи SOA ще в кінці 1990-х років для опису мережевої архітектури Jini для створення розподілених систем на основі динамічного виявлення та використання сервісів у мережі. В ній веб-служби прийняли концепцію служб (сервісів) та були реалізовані через API за допомогою таких технологій, як XML(Extensible Markup Language), опису мови веб-служб і доступу до них WSDL, протоколу обміну структурованими повідомленнями та доступу до об'єктів (SOAP), платформово-незалежного інструменту для розміщення описів веб-сервісів UDDI (Universal Description Discovery & Integration). Сучасна SOA – це принципи проєктування, які не прив'язана до якої-небудь технології реалізації. Як правило, її реалізовують за допомогою веб-сервісів чи мікросервісів (API-інтерфейсів), хоча можуть бути використані такі технології, як REST, RPC, DCOM, CORBA та інші. Найпоширеніша реалізація API SOA – через реалізацію архітектурного стилю REST (сервіси RESTful) на основі формату передачі даних JSON, рідше – через протокол SOAP на основі формату передачі даних XML.

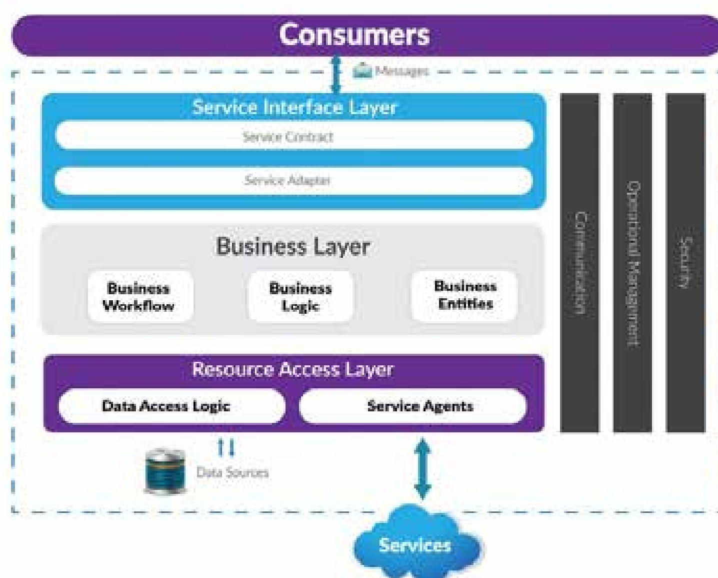

Рис. 4. Модель SOA

Більш складні реалізації SOA – це спеціалізована реалізація SOA на основі побудови та використанні мікросервісів, а також гібридні хмарні технології моделей розгортання і хмарного обчислення, інтегровані в SOA. Мікросервіс – це підхід до створення невеликих сервісів. Кожна служба має власну виділену область пам'яті та може обмінюватися повідомленнями. Кожен мікросервіс працює абсолютно незалежно, але, з іншого боку, всі вони слабо зв'язані між собою. Це, наприклад, означає, що кожен мікросервіс має власну базу даних. На відміну від класичного SOA додатки мікросервісної архітектури створені для виконання однієї бізнес-задачі, мають невеликий об'єм, добре масштабуються, використовують більш прості протоколи обміну даними (http) та просту систему обміну повідомленнями. Оскільки дане дослідження присвячене дослідженню слабо зв'язаних компонентів при обробці персональних даних в інформаційних технологіях, ми не будемо зупинятися на принципових перевагах та недоліках між цими технологіями проєктування, а акцентуємо увагу на управлінні безпекою, тобто захисту інформації при передаванні та обробці персональної інформації в даних розподілених технологіях.

Вище зазначалося, що основними перевагами класичної SOA є використання технології слабо зв'язаних компонентів (для інтеграції сервісу із будь-якою системою), повторне використання сервісу для скорочення часу розробки нового компоненту та стандартний формат обміну повідомленнями (як правило XML чи JSON). Для впровадження об'єктів інфраструктури у прикладний рівень SOA зазвичай використовують DI, який може бути реалізований як частина артефакту (зборки), наприклад, у рамках проєкту веб-API або проєкту веб-програми MVC. У випадку мікрослужби, створеної за допомогою кросплатформенної мови програмування (ASP.NET Core, Java чи іншої), прикладним рівнем зазвичай є бібліотека веб-API. Хоча це є перевагами в реалізації Service Interface Layer, це суттєво ускладнює управління всією інформаційною системою, особливо управління безпекою. Якщо ж розглядати модель SaaS (програмне забезпечення як послуга) – модель поширення програмного забезпечення, в якій хмарний провайдер розміщує програми та робить їх доступними для кінцевих користувачів через Інтернет – у цій моделі захист персональних даних є більш вразливим, оскільки незалежний постачальник програмного забезпечення (ISV) може укласти договір із стороннім постачальником хмарних послуг для розміщення програми. При цьому, коли два сервіси взаємодіятимуть один з одним, час відгуку збільшиться, що вимагає наявності серверів з високою пропускною здатністю.

SaaS – це одна з трьох основних категорій хмарних обчислень, поряд з інфраструктурою як послугою (IaaS) і платформою як послугою (PaaS). Цілий ряд IT-фахівців, бізнес-компаній та приватних користувачів використовують програми SaaS. Послуги даної технології варіюються від персональних розваг, таких як API Netflix, до передових IT-інструментів. На відміну від IaaS та PaaS, продукти SaaS часто продаються у таких сферах бізнесу як B2B (Business-to-Business) і B2C (Business-to-Consumer). SaaS працює через модель хмарної доставки. Постачальник програмного забезпечення або розміщує програму та пов'язані з нею дані в мережі, використовуючи власні сервери, бази даних, мережеві та обчислювальні ресурси, або це може бути ISV (Independent Software Vendor), який укладає контракт з постачальником хмарних послуг на розміщення програми в центрі обробки даних постачальника. Програма буде доступна для будь-якого пристрою з підключенням до мережі. Доступ до програм SaaS зазвичай здійснюється через API.

Механізми несанкціонованого доступу в SOA та шляхи їх подолання. Основними вразливими місцями в проєктуванні SOA як для систем з мікросервісами, так і для гібридних хмарних технологій моделей розгортання і хмарного обчислення, є:

1) ідентифікація та аутентифікація користувача в системі;
2) передача трафіка між клієнтами і сервісами SOA;
3) забезпечення цілісності даних.

Хоча і існують принципи механізмів безпеки The Global XML Web Services Architecture (GXA) зі специфікаціями WS-Security, стандарти безпечної передачі даних SAML (Security Assertion Markup Language), стандарт опису політику управління надання доступів XACML (eXtensible Access Control Markup Language), проте існує безліч вразливостей при передачі даних, зокрема при використанні SSO (Single Sign-On) – технології, при використанні якої користувач переходить з одного розділу порталу в інший або з однієї інформаційної системи в іншу, не пов'язану з першою системою, без повторної аутентифікації. Крім того, існує безліч вірусних та хакерських, зокрема DoS, атак на обчислювальну систему з метою довести її до відмови, тобто створення таких умов, за яких сумлінні користувачі системи не зможуть отримати доступ до запитаних ресурсів, що надаються, або цей доступ буде утруднений.

Так, протягом 2020–2022 років, коли весь світ боровся з пандемією коронавірусу, ландшафт кіберзагроз зріс в рази. Це пояснюється пришвидшеною інформатизацією та запровадженням інформаційних послуг в різних сферах життя (надання державних послуг через IT, дистанційне навчання тощо). Лідер в області виявлення кіберзагроз мережевої безпеки, глобальна хмарна архітектура Cisco Umbrella визначила основні тенденції загроз, які матимуть серйозні наслідки далеко за межами 2022 року. Дані результати дослідження представлені на рис. 4. Серед них серйозну занепокоєність викликають такі тенденції [13]:

1) трояни та дроппери отримують друге життя як нові форми доставки шкідливого ПЗ;
2) поетапні добре продумані атаки з ухилянням стають нормою;

3) зловмисники використовують контент, пов’язаний з пандемією, для поширення кіберзагроз;

4) криптомайнінг відкриває двері іншим типам кіберзагроз.

Хакерські ж атаки залишаються класичними. Це Cross-Site Scripting (XSS), постійні та пасивні XSS, SQL Injection та Cross-Site Request Forgery (CSRF). Варто пам’ятати, що дані при використанні протоколу HTTP передаються в текстовому вигляді, відповідно перехопивши значення, що надсилаються на сервер, можна легко змінити та порушити роботу серверної програми. При цьому будь-які вхідні дані можуть бути підробленими, зокрема: вхідні URL-адреси, дані, які були прийняті з форми, cookie-набори, дані в HTTP заголовках. При отриманні важливих даних з боку користувача потрібно переконатися в тому, що дані були надіслані програмою, а не підмінені, для цього можна використовувати шифрування або додавати до даних спеціальний підпис, який перевірятиме сервер.

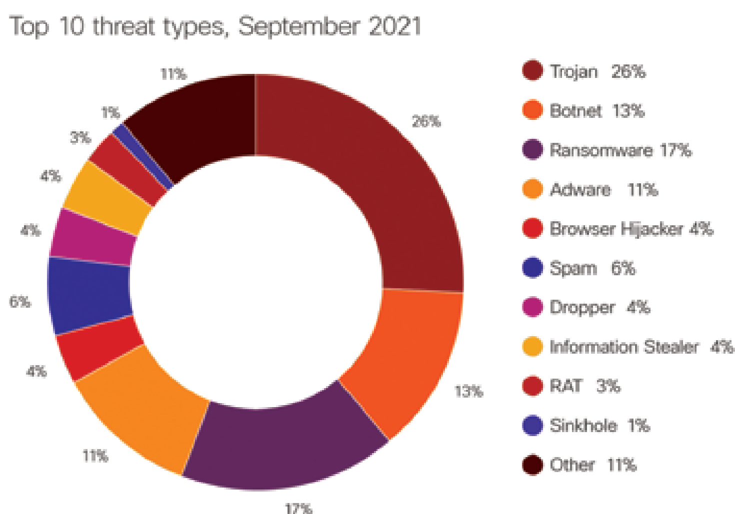

Рис. 5. Види кібератак на інформаційні системи за серпень 2021
за даними глобальна хмарної архітектури кібербезпеки Cisco Umbrella [14]

Розглянемо кожну атаку на SOA детальніше та опишемо шляхи захисту від них. Міжсайтовий сценарій або Cross-Site Scripting – одна з найбільш поширених в вразливостей. Якщо зломщик може змусити сервер повертати відвідувачам довільний сценарій JavaScript, то такий сценарій може керувати сеансами браузерів цих відвідувачів. Тоді зломщик може:

1) змінити дизайн сайту, вивести на ньому додатковий контент;

2) перенаправити користувача на інший сайт;

3) збирати конфіденційні дані (ідентифікатори сеансу з cookie наборів).

Для перехоплення та заміни HTTP запитів можна використовувати веб-наладчик fiddler. Напишемо примітивний сайт та зламаємо сторінку адміністратора. Нехай для прикладу метод на контроллері для ідентифікації адміністратора має вигляд:

```
public ActionResult Index()
{
HttpCookie adminCookies = Request.Cookies["IsAdmin"];
if (adminCookies != null)
{
if (adminCookies.Value.ToLower() == "true")
{
return View("AdminIndex");
}
else
{
return View("UserIndex");
}
}
return View();
}
```

Тоді для підміни сценарію верифікації достатньо у веб-наладчику змінити значення параметра IsAdmin з false на true. Після чого отримуємо повний доступ до адміністрування сайту.

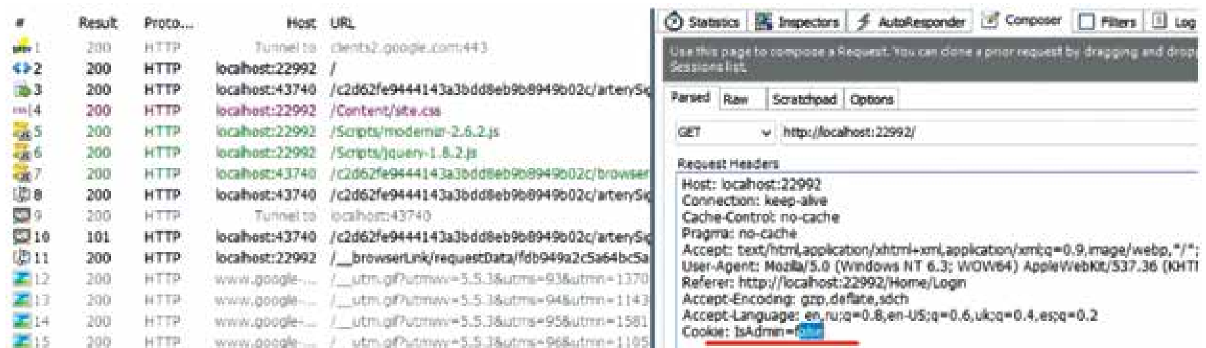

Рис. 6. Приклад кібератаки на сайт на основі сценарію XSS

Сценарії XSS бувають постійними та пасивними. Постійні XSS – в інтерактивний елемент (дошку оголошень, повідомлення) вноситься контент, який буде збережений в базу даних, а потім виданий іншим користувачам. Непостійні або пасивні XSS – це надсилання шкідливих даних у відповідь на запит нашої програми. Зломщику лише потрібно змусити жертву надіслати такий запит зі свого браузера. Продемонструємо це на прикладі. Зайшовши на дошку оголошень, ми створюємо нове повідомлення, але в полі повідомлення замість тексту ми вводимо html-розмітку. Цим самим змінюємо поведінку сайту (в даному випадку величину тексту).

Рис. 7. Приклад кібератаки на сайт на основі постійного сценарію XSS

Є різні варіанти протидії сценаріям XSS. Серед них це: відключення валідації даних, отриманих від клієнта, на прикладному рівні SEO, використання бібліотеки AntiXSS (Microsoft Web Protection Library), яка дозволяє перевіряти дані, отримані з боку клієнта, на наявність небезпечної розмітки.

Один із поширених способів злому веб-сервісів, які посилають свої запити базу даних, залишається SQL Injection. Ця атака полягає у впровадженні користувальницького SQL коду в запит, що призводить до зміни даних на сторінці, що в свою чергу веде до витоку даних або порушення цілісності бази даних. Припустимо, що метод на контроллері для відправки даних в базу даних має такий вид:

```
[HttpPost]
public ActionResult AuthFormBad(string login, string password) {
string connectionString = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
 using (SqlConnection connection = new SqlConnection(connectionString)) {
 string query = string.Format("SELECT Login FROM Users WHERE Login=‘{0}’ AND Password=‘{1}’",
login, password);
/* if the user passes as a value login dates x’ OR 1=1 -- , and the password will be blank so the following SQL query will be generated: SELECT * FROM Users WHERE Login=‘x’ OR 1=1 -- AND Password=“. Such a query will select all records from the Users table, since the value 1 is always equal to 1, even if the user named x is not in the database. Accordingly, the password will not be checked.*/
 SqlCommand command = new SqlCommand(query, connection);
 connection.Open();
 object userLogin = command.ExecuteScalar();
 if (userLogin != null)
 return View("Completed", userLogin);
 else
 ModelState.AddModelError(string.Empty, "Login or Pass has error");
 }
 return View();
}
```

Тоді атака на аккаунт може мати вигляд:

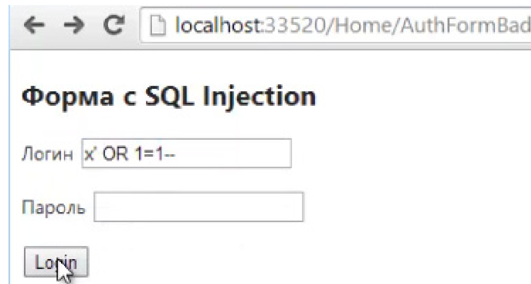

Рис. 8. Приклад кібератаки на основі SQL Injection

Існують такі способи захисту від SQL Injection: використання збережених процедур, запитів з SQL параметрами, використання ORM (наприклад, Entity Framework) та інші.

Ще однією поширеною атакою є Cross-Site Request Forgery (CSRF). На зовнішньому домені розміщується html-форма з даними, які повертаються на атакуючий сайт. Коли користувач потрапляє на цей зовнішній домен, він змушує свій браузер виконати відправку форми зі шкідливими даними на атакуючий сайт. Тобто по суті йде підміна компонентів SOA.

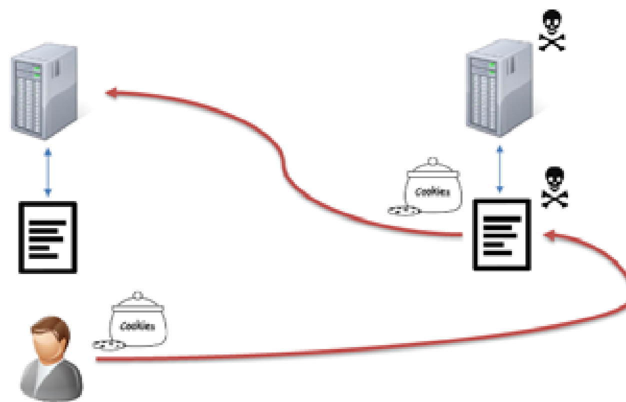

Рис. 9. Приклад кібератаки на основі CSRF

Існують різні способи захисту від CSRF. Серед них:

1) перевірка вхідного HTTP-заголовку referer;

2) перевірити чи не додавалися до небезпечних запитів елементи, пов'язані з даними користувача, наприклад, пароль облікового запису;

3) використання допоміжних методів, наприклад @Html.AntiForgeryToken() для представлення, та фільтрів [ValidateAntiForgeryToken] для методів дії в контролері.

Також слід додати, що основним механізмом в забезпеченні реалізації єдиної політики безпеки та інтеграції корпоративних додатків (EAI) у розподілені інформаційні системи, зокрема у SOA, є використання сервісної шини підприємства (ESB). Саме вона реалізує технології слабо зв'язаних компонентів при обробці персональних даних і слугує фреймворком для обміну повідомленнями. ESB забезпечує обмін даними між різними сервісами, зокрема між рівнем даних та прикладним рівнем в SOA.

Найпоширенішим її варіантом використання є Oracle Enterprise Service Bus (Oracle ESB). На ряду зі зворотними проксі-серверами та балансувальники навантаження, такими як NGINX та NGINX Plus, даний компонент реалізує механізми єдиної політики безпеки SOA.

Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі. В даній роботі представлені основні принципи об'єктно-орієнтованого проєктування технології слабозв'язаних компонентів для проєктування і розроблення розподілених інформаційних систем. Основним методом зменшення зв'язаності програмних модулів IT є використання IoC через реалізацію механізму ін'єкції залежностей. Існує кілька способів передачі ін'єкції залежностей у клас, включаючи різні типи DI, такі як ін'єкція конструктора, ін'єкція параметра, ін'єкція сетера та ін'єкція інтерфейсу. Побудова SOA розглядається з урахуванням принципу слабкого зв'язку, зокрема, на основі використання мікросервісів і гібридних хмарних технологій. Такі способи організації інфраструктури досить вразливі для хакерських атак (оскільки вхідні URL-адреси, файли cookie, дані, отримані з форм, і дані в HTTP-заголовках можуть бути підробленими). Так, за останній рік основними кібератаками на інформаційні системи стали: використання троянів та дропперів, зокрема, при передачі трафіку між клієнтами та сервісами AIC. Типи хакерських атак залишаються

класичними: постійний і пасивний міжсайтовий сценарій, SQL-ін'єкції і підробка міжсайтового запиту. В роботі детально проаналізовано основні механізми захисту даних від кібератак.

У перспективі планується розглянути сучасні механізми безпеки в SOA, зокрема застосування блокчейн у фінансових операціях та його інтеграцію з корпоративною шиною Oracle, яка реалізує технологію слабозв'язаності на рівні даних SOA.

Список використаних джерел:

1. Stevens, Wayne P.; Myers, Glenford J.; Constantine, Larry LeRoy (June 1974). Structured design. IBM Systems Journal. 13(2): 115–139. doi:10.1147/sj.132.0115.
2. Yourdon, Edward; Constantine, Larry LeRoy (1979). Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Yourdon Press. Bibcode:1979sdfd. ISBN 978-0-13-854471-3.
3. Philip A. Laplante, Philip A. Laplante. What Every Engineer Should Know about Software Engineering. CRC Press, 2007. P. 105–106. ISBN 978-1-4200-0674-2.
4. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides Design Patterns: Elements of Reusable Object-Oriented Software Published. Addison-Wesley Professional, 1994. 395 p. ISBN 0-201-63361-2.
5. Martin Fowler Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2022. 560 p. ISBN: 0321127420.
6. Neal Ford, Mark Richards, Pramod Sadalage, Zhamak Dehghani Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures. O'Reilly Media, 2021. 450 p. ISBN: 1492086894.
7. Mark Richards, Neal Ford Fundamentals of Software Architecture: An Engineering Approach. – O'Reilly Media, 2020. 396 p. ISBN: 1492043451.
8. ISO/IEC/IEEE 24765:2010 Systems and software engineering.
9. Zinchenko, A. Y. (2022). Dependency injection using for the develop of information technology for investigation of parametric var methods. Systems and Technologies, 62(2), 63–75.
10. Ralph E. Johnson, Brian Foote (1988). Designing Reusable Classes. Journal of Object-Oriented Programming, 1(2), 22–35. doi:10.1.1.101.8594.
11. Michael Mattsson (1996). Object-Oriented Frameworks. A survey of methodological issues. Department of Computer Science, Lund University, 130 p.
12. Faathima Fayaza (2014) Service oriented architecture in enterprise application. Dept. Inf. Tech.,Univ. Moratuwa, Katubedda, Sri Lanka, Tech. Rep., 8 p.
13. URL: https://umbrella.cisco.com/info/technical-paper-modern-security-landscape-scaling-threats-motion (Retrieved 2022-12-11).
14. URL: https://umbrella.cisco.com/trends-threats/global-cyber-threat-intelligence-overview (Retrieved 2022-12-11).

References:

1. *Stevens, Wayne P.; Myers, Glenford J.; Constantine, Larry LeRoy* (June 1974). "Structured design". IBM Systems Journal. 13(2): 115–139. doi:10.1147/sj.132.0115.
2. *Yourdon, Edward; Constantine, Larry LeRoy* (1979). Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. Yourdon Press. Bibcode:1979sdfd. ISBN 978-0-13-854471-3.
3. *Philip A. Laplante, Philip A. Laplante*. What Every Engineer Should Know about Software Engineering. – CRC Press, 2007. – P. 105–106. – ISBN 978-1-4200-0674-2.
4. *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides* Design Patterns: Elements of Reusable Object-Oriented Software Published. Addison-Wesley Professional, 1994. 395 p. ISBN: 0-201-63361-2.
5. *Martin Fowler* Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2022. 560 p. ISBN: 0321127420.
6. *Neal Ford, Mark Richards, Pramod Sadalage, Zhamak* Dehghani Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures. O'Reilly Media, 2021. 450 p. ISBN: 1492086894.
7. *Mark Richards, Neal Ford* Fundamentals of Software Architecture: An Engineering Approach. – O'Reilly Media, 2020. – 396 p. – ISBN: 1492043451.
8. ISO/IEC/IEEE 24765:2010 Systems and software engineering.
9. *Zinchenko, A. Y.* (2022). Dependency injection using for the develop of information technology for investigation of parametric var methods. *Systems and Technologies*, *62*(2), 63–75.
10. *Ralph E. Johnson, Brian Foote* (1988). Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2), 22–35. doi:10.1.1.101.8594.
11. *Michael Mattsson* (1996). Object-Oriented Frameworks. A survey of methodological issues. *Department of Computer Science, Lund University*, 130 p.
12. *Faathima Fayaza* (2014) Service oriented architecture in enterprise application. *Dept. Inf. Tech., Univ. Moratuwa*, Katubedda, Sri Lanka, Tech. Rep., 8 p.
13. https://umbrella.cisco.com/info/technical-paper-modern-security-landscape-scaling-threats-motion (Retrieved 2022-12-11).
14. https://umbrella.cisco.com/trends-threats/global-cyber-threat-intelligence-overview (Retrieved 2022-12-11).