Interdisciplinary Description of Complex Systems 24(4), 446-461, 2026

USING THE TECHNOLOGY OF LOOSELY CONNECTED COMPONENTS FOR THE SAFE PROCESSING OF PERSONAL DATA IN SOCIAL NETWORKS AND OTHER COMMUNICATION PLATFORMS

Artem Yurievich Zinchenko[1, *] and Olha Zinovievna Zinchenko[2]

[1]National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Educational and Research Institute for Applied System Analysis
Kyiv, Ukraine
[2]The National Transport University
Kyiv, Ukraine

DOI: 10.7906/indecs.24.4.6
Regular article

Received: 3 June 2023.
Accepted: 8 May 2026.

ABSTRACT

In this article, we consider the construction of architectural solutions based on the technology of loosely coupled software modules of automated information systems of enterprises and institutions for the secure processing of personal data in social networks and other communication platforms. The use of the Dependency inversion principle in the implementation of service-oriented architecture, the construction of web services (SOAP and REST), and the architectural model of SaaS cloud computing is analysed in detail. The main mechanisms of unauthorized access in service-oriented architecture and ways to overcome them are presented. We also consider how loosely coupled components such as blockchain and decentralized systems can be used to ensure the secure processing of personal data.

KEY WORDS

IoC, DI, DIP, SOA, blockchain

CLASSIFICATION

JEL: L86, L96

*Corresponding author, η: artem005@yahoo.com; +38 0671703717;
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Educational and Research Institute for Applied System Analysis, Beresteiskyi Ave 37-A, UA – 03056 Kyiv, Ukraine

INTRODUCTION

Nowadays, more and more organizations are moving from client-server to service-oriented architecture, in particular, using services built according to SOAP, XML-RPC, and JSON-RPC protocols, or on the REST architectural style. In addition, a lot of portal solutions have been introduced in Europe in recent years, including at the level of public administrations (for example, the European Data Portal – a pan-European repository of public sector information (https://data.europa.eu/en), a single portal of public services of Ukraine (https://diia.gov.ua), open platform for French public data (https://www.data.gouv.fr/fr), open data portal of the Italian Public Administration (https://www.dati.gov.it), etc.) in which security becomes one of the key issues affecting software deployment. Access to the company's confidential information, in particular personal data, is carried out using services deployed on distributed SOA components using the API. Therefore, security issues have become a part of the decision-making process for enterprises to adopt SOA.

The same trend is observed for CRM systems (customer relationship management systems) and ERP systems (enterprise resource planning systems), which as a rule were designed in a two- or three-tier client-server architecture 10-20 years ago. And among them, not all information systems had full-fledged application services. Currently, to effectively manage, analyze and improve customer relationships, the information technology (IT) of such automated systems must have a comprehensive set of cloud solutions that can support and accompany the user at every stage of the workflow (a system for managing repetitive processes and tasks that are performed in a specific order). This end-to-end solution should include cloud solutions for sales, service, commerce, marketing, and an AI-enabled customer data platform (CDP) that can operate online, offline and third-party data sources for a complete and dynamic presentation of them to the client. Among the latter, the most common model of cloud solutions is SaaS (software as a service), access to the software of it is provided remotely via network channels through a web interface or terminal.

The development of distributed information systems using Dependency Injection (DI) technology can be an effective way to solve the problem of personal data processing under dynamic requirements and variable conditions. One of the key aspects of using the technology of loosely connected components for the safe processing of personal data is their decentralization. Such systems can help avoid problems associated with the centralized processing of personal data, such as the possibility of intrusion by intruders and data leaks. The use of blockchain technologies can ensure the security and reliability of data storage since they are stored on different network nodes and protected by encryption. However, it should be noted that the use of such technologies can also have its drawbacks, such as limited data processing speed and high computational costs. Therefore, when applying these technologies to social networks and other communication platforms, it is necessary to balance the advantages and disadvantages in order to ensure the efficient and secure processing of personal data.

Since software component coupling is one of the fundamental characteristics of object-oriented design, the development of loosely coupled modules remains a relevant direction in the design of service-oriented and distributed information systems.

Despite the widespread adoption of service-oriented architectures, microservice platforms, and cloud-based services, the problem of secure processing of personal data in distributed systems remains highly relevant. Traditional centralized approaches introduce risks associated with a single point of failure, limited scalability, and strong interdependence between system components.

In this context, this work investigates the use of loosely coupled component technologies, DI mechanisms, and service-oriented architectures to improve the security, flexibility, and scalability of personal data processing systems in social networks and other communication platforms.

The main focus of the study is on analyzing approaches for reducing dependencies between components of distributed systems, the specifics of integrating SOA with blockchain technologies, and mechanisms for protecting against common cyber threats in service-oriented environments.

The scientific novelty of the work lies in the analysis of approaches to building secure SOA systems based on loosely coupled components, the investigation of the role of DI in improving system flexibility and security, and the examination of blockchain technologies as a means of ensuring the integrity and confidentiality of personal data.

In contrast to traditional centralized client-server architectures with tightly coupled components, the proposed approach simplifies service scalability, increases system flexibility, and reduces risks associated with centralized data processing.

IMPLEMENTATION OF INVERSION OF CONTROL IN DISTRIBUTED INFORMATION TECHNOLOGIES

The basic principles of coupling and cohesion were invented by Larry Constantine in the late 1960s as part of Structured Design to reduce maintenance and modification costs [1, 2]. In software engineering, coupling characterizes the degree of interdependence between software modules; this is a measure of how closely connected two routines or modules are [3]. Low code coupling is a sign of a well-structured and designed AIS. It often correlates with high cohesion and vice versa. If software coupling refers to the interdependencies between modules, then cohesion describes the measure of the relatedness of functions within a single module. Low cohesion means that the methods and classes of AIS components perform tasks that are not related to each other, which leads to maintenance and development problems during further development of the software and expansion of these components. Tightly coupling between AIS components (for example, client and server parts of the software) is a serious drawback of the software, since it makes it difficult to understand the logic of the modules, modify them, stand-alone testing and reuse. Loose coupling is the core principle of SOLID OOP, the foundation of The Dependency Inversion Principle (DIP), and one of Craig Larman's nine GRASP object-oriented design patterns [4] for solving common problems of assigning responsibility to classes and objects.

There are different methods to reduce the coupling of software modules. They are embodied in design patterns. Among the latter, one can distinguish such design patterns of a multilayer architecture as Model-View-Controller, Model-View-Presenter, Model-View-View Model and others. One approach to reducing coupling is functional design, which seeks to limit the responsibilities of modules with functionality. One of the key methods for implementing the technology of loosely coupled components of software is the use of inversion of control, that is, the implementation of the DI mechanism.

Inheritance is the basic principle of OOP and allows one class (child) to inherit the functionality of another class (parent). Often, inheritance relations are also called generalization or generalization. Inheritance defines the relationship "IS A".

Before proceeding to the implementation of the principle of inversion of control in distributed automated information systems, one should deeply understand the principles of object-oriented design: the difference between IoC, DI and DIP, between patterns, frameworks, libraries, dependencies, as well as the difference between the relationships of designed classes. Let us briefly consider the latter, that is, the main relationships between objects of classes, which will help us understand the relationship between entities when they are used in design patterns. There are several main types of relationships: inheritance, implementation, association, composition and aggregation.

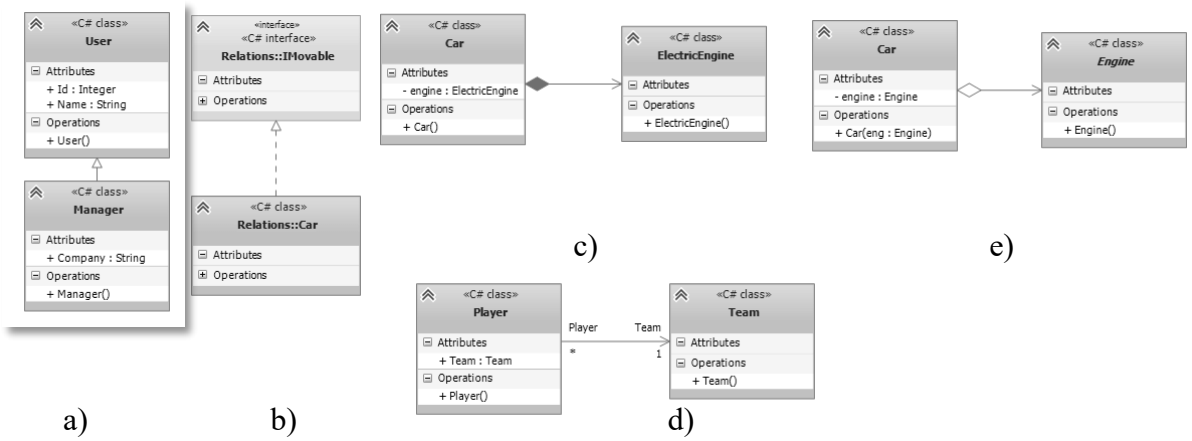

Figure 1. Relationships between classes and objects: a) inheritance, b) implementation, c) composition, d) association, and e) aggregation.

Implementation involves defining an interface and implementing it in classes. According to the Interface Segregation Principle (ISP), when creating a system of relationships between classes, you should program only at the level of interfaces, and not their specific implementations. That is, when designing an IT architecture, programmers should not be forced to implement interfaces where they are not needed. Interfaces here should be understood not only as programming language types defined using the interface keyword but also as a definition of a class's functionality without its specific implementation.

An association is a relationship in which objects of one type are related in some way to objects of another type. For example, an object of one type contains or uses an object of another type. At the same time, aggregation and composition are special cases of association, and they are characterized by the relation "HAS A". The difference is that composition implements hard coupling: a new instance of the class is created in the class constructor, which completely controls the life cycle of an object of this class. In the case of aggregation, a weak connection is implemented, that is, the object passed through the constructor is equal to the object of the class of the constructor to which it was passed. A reference to an already existing object is passed to the constructor, for which memory has already been allocated. And usually, a reference is passed not to a concrete class, but to an abstract class or interface, which increases the flexibility of the program. In addition, when inheriting, all the functionality of the descendant class is rigidly defined at the compilation stage, that is, when the program is executed, it is impossible to dynamically redefine it. This is why composition should be preferred over inheritance, and aggregation should be preferred over composition, as a more flexible way to reduce coupling through object-oriented design.

IoC is a fairly general concept that distinguishes a dynamic link library from a framework. Unlike traditional control flow, IoC inverts flow control. In the classic software architecture model, the called code controls the external environment and the timing and order of library method calls. However, in the case of a framework, the responsibilities are reversed: the framework provides some extension points on which it calls certain methods of user code.

Inversion of control is used to increase the modularity of the program and make it extensible [5]. The term was first used by Mattsson [6]. Callbacks, schedulers, event loops, DI, template method, and the Supervisor pattern are examples of design patterns that follow the inversion of the control principle.

The main technologies for implementing IoC are the use of such object-oriented design patterns as a service locator, template method, factory, and strategy, as well as using a contextualized lookup and DI. DI can be implemented through constructor injection, parameter injection, setter injection or interface injection.

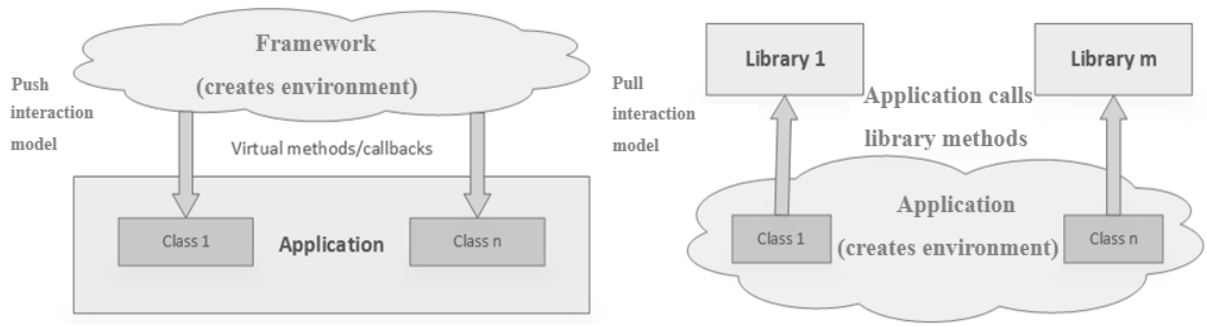

Figure 2. Inversion of control.

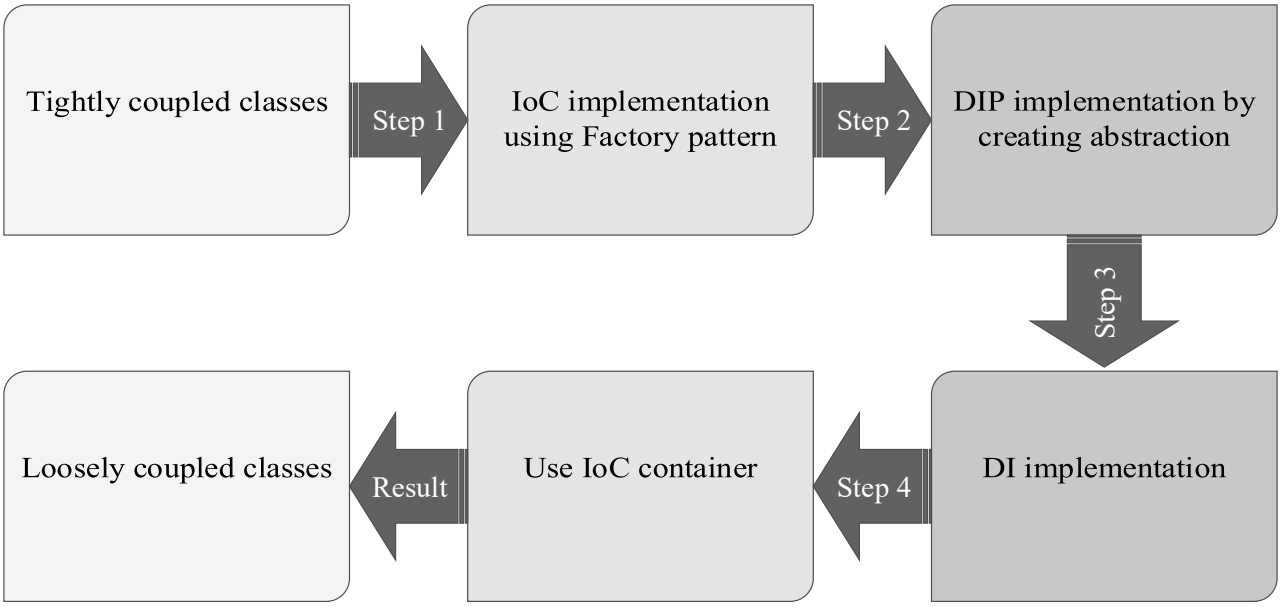

Figure 3. An example of an IoC implementation using the Factory pattern.

Let us consider DIP – the basic principle of SOLID, which gives recommendations on what the dependencies between high-level and low-level abstractions should be:

- modules of upper levels should not depend on modules of lower levels. At the same time, both types of modules must depend on abstractions;
- abstractions should not depend on details. Details should depend on abstractions;
- high-level modules implement business rules or logic in the system. Low-level modules deal with more detailed operations, for example, writing information to a database, passing messages to the operating system or services, etc.

In this case, class dependencies must be located at the current or higher level of abstraction. In other words, not every class, which requires an interface in a constructor, follows the DIP. Consider an example:

```
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));
        }
    }
```

The ReportProcessor class still accepts an “abstraction” in the ISocket constructor arguments, but this “abstraction” is several levels below that of generating and sending reports. The same is the case with the argument of the SendReport method: the “abstraction” of IStringBuilder does not comply with the DIP, as it operates with more low-level concepts than necessary. You should operate not with lines but with reports at this level. As a result, this code does not follow the DIP principle – this example uses DI.

DI is the main implementation mechanism of IoC – transfer of a class of its dependencies, it serves as a framework. As noted previously, there are several specific ways or patterns to use dependencies. Consider an example:

```
class ReportProcessor
{
    private readonly IReportSender _reportSender;
    // Constructor 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>
```

Different types of DI are designed to solve specific tasks. Mandatory dependencies of the class, without which the work of the class is impossible, are transferred through the constructor (IReportSender is a mandatory dependency of the ReportProcessor class). The method passes dependencies that only one method needs, not all methods of the class (IReportFormatter is only needed by the report submit method, not the entire ReportProcessor class). Only optional (usually infrastructure) dependencies should be set via properties for which there is a default

value (the Logger property contains a reasonable default value that can be changed later). For setter-based DI, the IoC container will call the setter methods of the Store class after calling the no-argument constructor or the no-argument static factory method to instantiate the *bean*.

SOA DESIGN IN DISTRIBUTED INFORMATION TECHNOLOGIES

Most enterprises have invested heavily in system resources over the years. Such enterprises have a huge amount of data stored in outdated corporate information systems (ERP and CRM systems), so it is not advisable to abandon existing systems. It is more profitable to develop and improve existing automated information systems. SOA solves this problem.

SOA is defined in many ways. One of them is "the paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [7].

In other words: SOA is an architectural style for creating software by which different types of services (information technology components for providing information services) can independently interact with each other. The foundation of SOA is the use of loosely coupled component technology, which promotes loose coupling between remote, standardized usage protocols and distributed software components so that they can be reused. SOA uses the "find-bind-execute" paradigm.

SOA is not a new concept. So, the Sun company defined the basic principles of SOA back in the late 1990s to describe the Jini network architecture for creating distributed systems based on the dynamic discovery and use of services in the network. In it, web services adopted the concept of services and were implemented through APIs using technologies such as XML (Extensible Markup Language), SOAP (Simple Object Access Protocol) – structured messaging and object access protocol, UDDI (Universal Description Discovery & Integration) – a platform-independent tool for hosting descriptions of web services. Modern SOA is design principles that are not tied to any implementation technology. As a rule, it is implemented using web services or microservices (APIs), although technologies such as REST, RPC, DCOM, CORBA, and others can be used. The most common implementation of SOA API is through the implementation of the REST architectural style (RESTful services) based on the JSON data transfer format, less often through the SOAP protocol based on the XML data transfer format.

More complex SOA implementations are specialized SOA implementations based on the construction and use of microservices, as well as hybrid cloud technologies of deployment models and cloud computing integrated into SOA. Microservice is an approach to creating small services. Each service has its dedicated memory area and can exchange messages. Each microservice works independently, but on the other hand, they are all loosely connected. This means, for example, that each microservice has its database. Unlike classical SOA, microservice architecture applications are designed to perform a single business task, are small in size, scale well, and use simpler data exchange protocols (HTTP) and a simple messaging system. Since this study is devoted to the study of loosely coupled components in the design of information technologies, we will not dwell on the fundamental advantages and disadvantages of different design technologies but will focus on security management, that is, information protection during the transmission and processing of personal information in these distributed technologies.

It was noted above that the main advantages of classical SOA are the use of loosely coupled component technology (for integrating the service with any system), reuse of the service to reduce the development time of a new component, and a standard message exchange format (as a rule, XML or JSON). To inject infrastructure objects into the SOA application layer, DI is usually used, which can be implemented as part of an artefact (assembly), for example, as part of a web API project or an MVC web program project. In the case of a microservice built

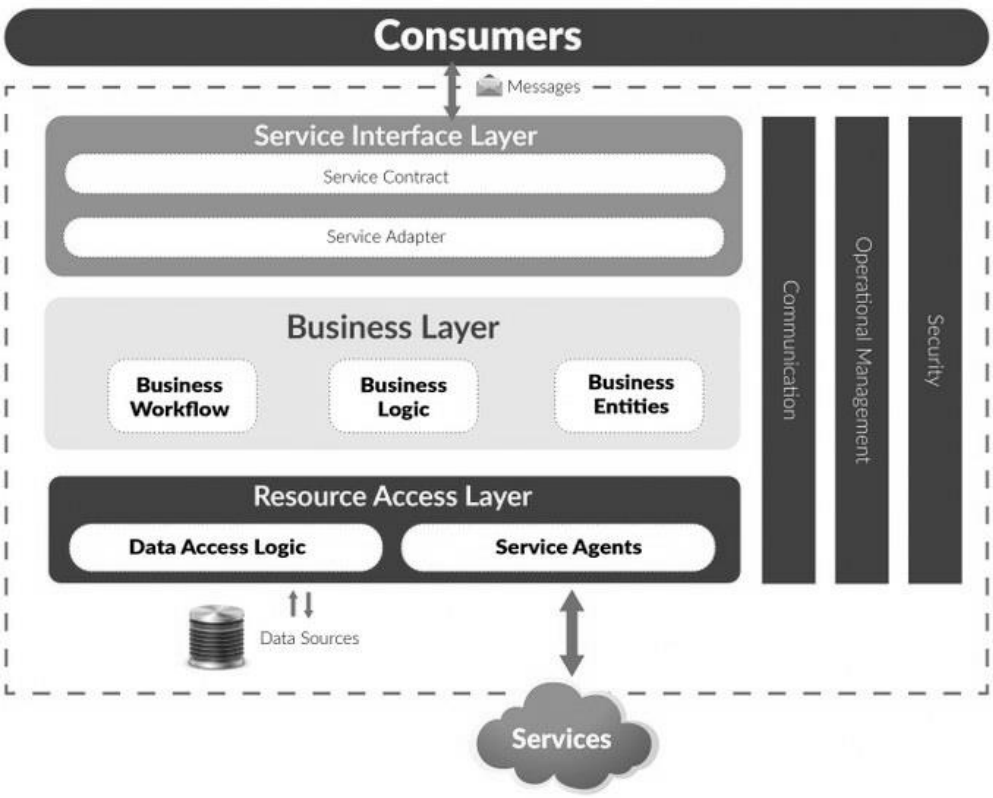

Figure 4. SOA model.

with a cross-platform programming language (ASP.NET Core, Java, or other), the application layer is typically a web API library. While these are advantages in the implementation of the Service Interface Layer, it greatly complicates the management of the entire information system, especially security management. If we consider the SaaS (software as a service) model – a software distribution model in which a cloud provider hosts applications and makes them available to end users over the Internet – in this model, personal data protection is more vulnerable, since the independent software vendor (ISV) may contract with a third-party cloud service provider to host the program. However, when two services interact with each other, the response time will increase, which requires servers with high bandwidth.

SaaS is one of the three main categories of cloud computing, along with infrastructure as a service (IaaS) and platform as a service (PaaS). SaaS applications are used by a wide range of IT professionals, business companies and private users. The services of this technology range from personal entertainment, such as the Netflix API, to advanced IT tools. Unlike IaaS and PaaS, SaaS products are often sold in businesses such as B2B (Business-to-Business) and B2C (Business-to-Consumer). SaaS works through a cloud delivery model.

The software provider either hosts the program and related data on the network using its servers, databases, network and computing resources, or it can be an ISV (Independent Software Vendor) contracting with a cloud service provider to host the program in the supplier data centre. The program will be available to any device with a network connection. SaaS programs are usually accessed through an API.

MECHANISMS OF UNAUTHORIZED ACCESS IN SOA AND WAYS TO OVERCOME THEM

The main vulnerabilities in SOA design for both microservices and hybrid cloud deployment models and cloud computing are:

1) identification and authentication of users in the system,
2) traffic transfer between clients and SOA services,
3) ensuring data integrity.

Although there are principles of security mechanisms The Global XML Web Services Architecture (GXA) with WS-Security specifications, SAML (Security Assertion Markup Language) secure data transfer standards, XACML (eXtensible Access Control Markup Language) policy description standard, however, there are many vulnerabilities in data transfer, in particular when using SSO (Single Sign-On) - a technology in which a user moves from one section of the portal to another or from one information system to another, not connected to the first system, without re-authentication. In addition, there are many viruses and hacker attacks, in particular DoS, on a computer system to bring it to failure, that is, the creation of such conditions under which conscientious users of the system will not be able to access the requested resources or this access will be difficult.

So, during 2020-2022, when the whole world was fighting the coronavirus pandemic, the landscape of cyber threats has grown significantly. This is due to accelerated informatization and the introduction of information services in various areas of life (provision of public services through IT, distance learning, etc.). A leader in network security cyber threat detection, Cisco Umbrella's global cloud architecture, has identified major threat trends that will have major impacts well beyond 2022. These results of the study are presented in Figure 5. Among them, the following trends are of serious concern [8]:

1) trojans and droppers are getting a second life as new forms of malware delivery,
2) orchestrated, multi-staged, evasive attacks are becoming the norm,
3) crypto mining is opening the door to other types of cyber threats,
4) attackers are taking advantage of pandemic-related content to propagate threats.

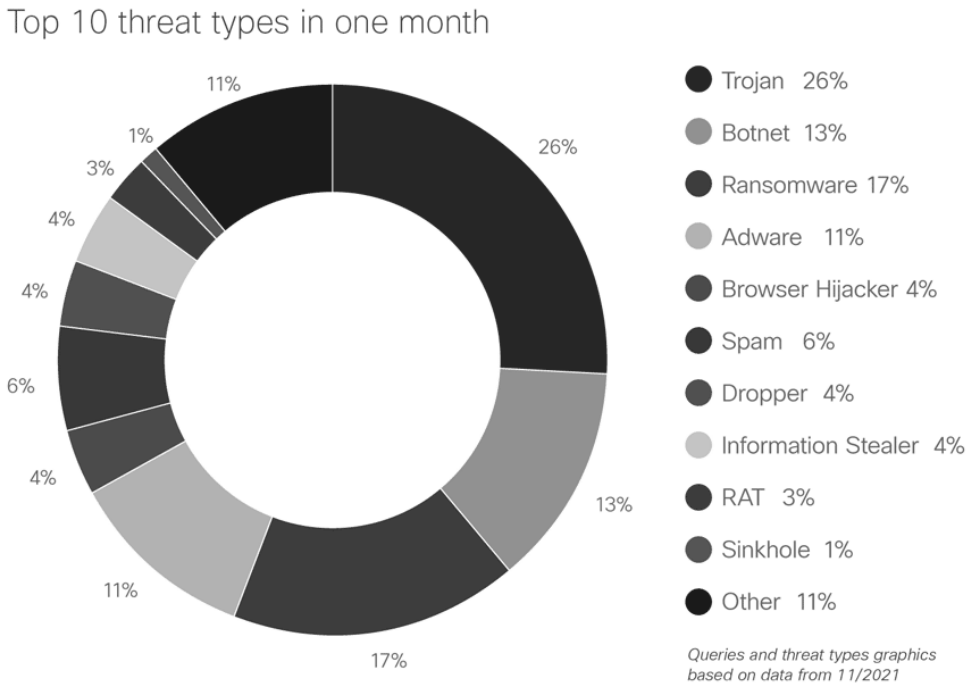

Figure 5. Types of cyberattacks on information systems for November 2021 according to the global cloud architecture of cyber security Cisco Umbrella [9].

Hacker attacks remain classic. These are persistent and passive Cross-Site Scripting (XSS) [10], SQL Injection [11] and Cross-Site Request Forgery (CSRF) [12]. It should be remembered that when using the HTTP protocol, data is transmitted in text form; accordingly,

intercepting the values sent to the server can be easily changed and disrupt the operation of the server program. However, any input data can be fake, in particular: incoming URLs, data received from a form, cookie sets, and data in HTTP headers. When receiving important data from the user's side, you need to make sure that the data was sent by the program and not spoofed, for this you can use encryption or add a special signature to the data, which will be checked by the server.

Let us consider each attack on SOA in more detail and describe ways to protect against them. Cross-Site Scripting is one of the most widespread vulnerabilities. If an attacker can force the server to return arbitrary JavaScript to visitors, then such a script can manipulate those visitors' browser sessions.

Then the hacker can:

1) change the design of the site, and display additional content on it,
2) redirect the user to another site,
3) collect sensitive data (session IDs from cookies).

The fiddler web debugger can be used to intercept and replace HTTP requests. Let us write a primitive site and hack the admin page. For example, let the method on the controller for identifying the administrator look like this:

```
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();

    }
```

Then, to replace the verification script, it is enough to change the value of the IsAdmin parameter from false to true in the web configurator. Then we get full access to the site administration.

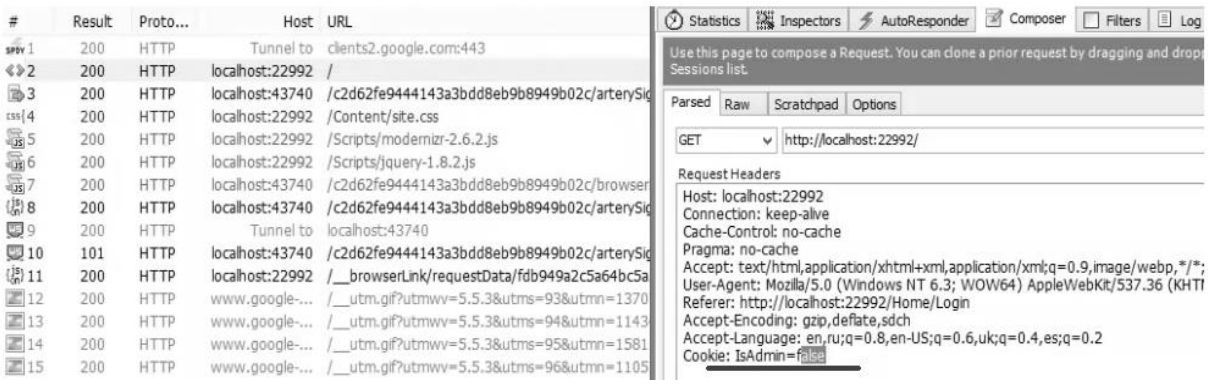

Figure 6. An example of a cyberattack on a website based on an XSS script.

XSS scripts are persistent and passive. Permanent XSS – content is entered into an interactive element (bulletin board, message), which will be stored in the database and then issued to other users. Non-persistent or passive XSS is the sending of malicious data in response to a request from our program. The attacker only needs to make the victim send such a request from their browser. Let us demonstrate this with an example. Having entered the bulletin board, we create a new message, but in the message field, instead of text, we enter HTML markup. In this way, we change the behaviour of the site (in this case, the size of the text).

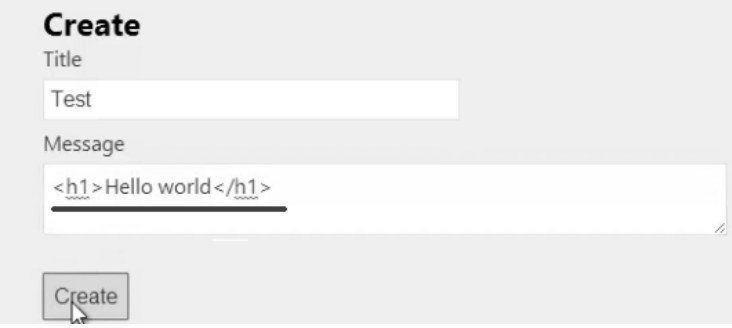

Figure 7. An example of a cyberattack on a site based on a persistent XSS script.

There are different options for countering XSS scripts. Among them are disabling validation of data received from the client at the SEO application level, and using the AntiXSS (Microsoft Web Protection Library) library, which allows you to check the data received from the client for dangerous markup.

One common way to hack web services that send their queries to a database remains SQL Injection. This attack consists of injecting custom SQL code into a query, which results in data changes on the page, which in turn leads to data leakage or database integrity violations. Suppose that the method on the controller for sending data to the database is as follows:

```
[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 the value x' OR 1=1 -- as the login value, and the password is empty, then
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, "Username or password entered incorrectly");
      }
   }
   return View();
}
```

Then the attack on the account might look like this:

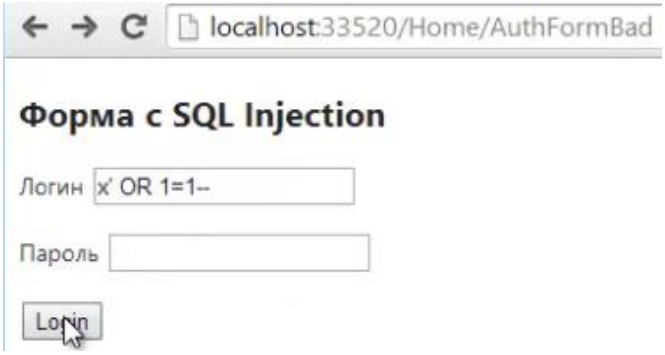

Figure 8. An example of a cyberattack based on SQL Injection.

There are the following ways to protect against SQL Injection: using stored procedures, queries by SQL parameters, using an ORM (for example, EntityFramework) and others. For example, the AuthFormBad method could be rewritten as follows:

```
[HttpPost]
public ActionResult AuthFormGood(string login, string password)
{
      string connectionString = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;
   using (SqlConnection connection = new SqlConnection(connectionString))
   {
      string query = "SELECT Login FROM Users WHERE Login=@Login AND Password=@Password";
      SqlCommand command = new SqlCommand(query, connection);
      command.Parameters.AddWithValue("Login", login);
      command.Parameters.AddWithValue("Password", password);
      connection.Open();
      object userLogin = command.ExecuteScalar();
      if (userLogin != null)
         return View("Completed", userLogin);
      else
         ModelState.AddModelError(string.Empty, "Username or password entered incorrectly");
   }
   return View();
}
```

Another common attack is Cross-Site Request Forgery (CSRF). The external domain hosts an HTML form with data returned to the attacking site. When a user lands on an external domain, their browser submits a form with malicious data to the attacking site. That is, in essence, there is a substitution of SOA components.

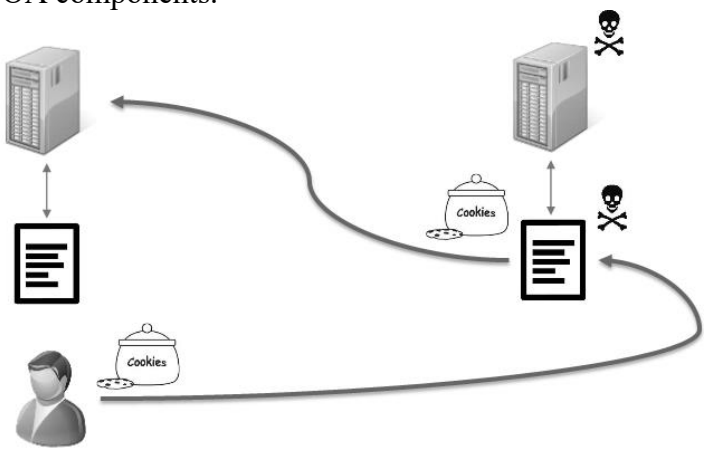

Figure 9. An example of a cyberattack based on CSRF.

There are different ways to protect against CSRF. Among them:

1) checking the incoming HTTP header “Referer”;
2) check if elements related to user data, such as an account password, have been added to dangerous requests;
3) use helper methods like @Html.AntiForgeryToken() for the view and [ValidateAntiForgeryToken] filters for the action methods in the controller.

It should also be added that the main mechanism for ensuring the implementation of a unified security policy and the integration of enterprise applications (EAI) into distributed information systems, in particular in SOA, is the use of an enterprise service bus (ESB). It implements the technology of loosely coupled components in data processing and serves as a framework for messaging. ESB provides data exchange between different services, in particular between the data layer and the application layer in SOA.

Its most common use case is Oracle ESB. Along with reverse proxies and load balancers such as NGINX and NGINX Plus, this component implements SOA unified security policy mechanisms.

THE USE OF BLOCKCHAIN FOR THE SECURE PROCESSING OF PERSONAL DATA

The emergence of digital technologies has facilitated the collection, storage, and processing of vast amounts of personal data. However, this has also raised concerns about data privacy and security. Blockchain technology, which provides secure, decentralized, and transparent data storage and processing, has the potential to address these concerns. In this section, we discuss the use of blockchain technology for secure processing of personal data.

Personal data refers to any information that relates to an identified or identifiable natural person. This includes names, addresses, phone numbers, email addresses, social security numbers, and other sensitive information. The use of personal data has become an integral part of many digital services, including social media platforms, e-commerce websites, and online banking services. However, the centralized storage of personal data has resulted in numerous data breaches and incidents of identity theft.

Blockchain technology provides a decentralized and secure method for storing and processing personal data. Blockchain is a distributed ledger technology that stores data across a network of nodes, with each node maintaining a copy of the ledger. Transactions are validated by consensus among the nodes, and once a transaction is added to the blockchain, it cannot be altered. This ensures the integrity and immutability of the data.

The main area of application of blockchain technologies remains the crypto industry. However, blockchain projects are used in a variety of areas, including healthcare, for example, to store medical information, which ensures its confidentiality. For example, once a medical record is created, it can be recorded on the blockchain, giving patients proof and confidence that the record cannot be changed. These personal health records can be encrypted and stored on the blockchain with a private key so that they can only be accessed by certain privacy guards [13, 14].

In the banking industry, blockchain helps to make all processes more reliable and transparent. The technology can be used for money transfers, settlements with securities, and routine tasks of banks in general. Blockchain is also used in electronic voting. In blockchain voting services, the service operator does not have the ability to change both the original data and the completed ballots, and the voting data cannot be lost. Blockchain voting can prevent election fraud and increase voter turnout. The blockchain-based system was used to vote in the summer of 2018 in the Swiss city of Zug and in November 2018 in West Virginia in the midterm elections. In addition, the blockchain can be used, for example, to store information about citizens – when creating digital identity cards to access government services.

The use of blockchain technology for processing personal data offers several benefits, including:

- Security – blockchain provides a secure method for storing and processing personal data. Data is encrypted and stored across a network of nodes, making it difficult for hackers to compromise the system. Additionally, transactions are validated by consensus, ensuring the integrity of the data.
- Transparency – blockchain provides transparency by allowing users to view all transactions on the network. This helps to prevent fraud and ensures the accuracy of the data.
- Decentralization – blockchain is a decentralized technology, meaning that there is no central authority controlling the data. This reduces the risk of a single point of failure and ensures that the data is not controlled by a single entity.

To demonstrate the possibilities of using blockchain technology and the DI mechanism within service-oriented architectures, let us consider a simplified example of the interaction between loosely coupled components in a secure personal data processing system.

The presented example has a conceptual and demonstrative nature and is intended to illustrate the architectural principles of integrating blockchain services into a distributed environment using the DI mechanism. The primary focus is placed on the interaction logic between system components rather than on implementation details or algorithm optimization.

In the proposed approach, blockchain technology is used to ensure data integrity, while DI provides loose coupling between services and simplifies the scalability of distributed system components.

The complete implementation of the C# source code is available in the public repository [16]. Below, a simplified interaction model of the system components is presented.

1) Initialization of the blockchain service.
2) Registration of services in the DI container.
3) Receiving personal data from a user or external service.
4) Creation of a blockchain record.
5) Adding the record to the blockchain chain.
6) Verification of blockchain integrity.
7) Returning the data processing result.

A simplified implementation example is presented further in the text.

```
public class BlockchainService
{
    public void AddBlock(string data)
    {
        // Add a new record to the blockchain
    }
    public bool ValidateChain()
    {
        // Verify blockchain integrity
        return true;
    }
}

public class PersonalDataManager
{
    private readonly BlockchainService blockchainService;

    public PersonalDataManager(BlockchainService blockchainService)
```

```
        {
            this.blockchainService = blockchainService;
        }

        public void ProcessPersonalData(string personalData)
        {
            blockchainService.AddBlock(personalData);
        }
    }
```

In the presented example, the BlockchainService class is responsible for blockchain operations, while the PersonalDataManager class implements personal data processing functions. Dependency transfer through the constructor demonstrates the use of the DI mechanism for reducing coupling between system components.

The proposed approach improves the flexibility of service-oriented architectures, simplifies component maintenance and scalability, and reduces risks associated with centralized personal data processing.

In real distributed systems, such an architecture additionally requires the implementation of encryption mechanisms, access control policies, user authentication procedures, and secure network communication between services.

CONCLUSIONS

This article presents the basic principles of object-oriented design technology of loosely coupled components for the development of distributed information systems. The main method for reducing the coupling of IT software modules is the use of IoC through the implementation of the DI mechanism. There are several ways to transfer DI to a class, including DI types such as Constructor injection, Parameter injection, Setter injection, and Interface injection. The construction of SOA is considered to take into account the principle of loose coupling, in particular, based on the use of microservices and hybrid cloud technologies. Such ways of organizing the infrastructure are quite vulnerable to hacker attacks (because incoming URLs, Cookies, data received from forms and data in HTTP headers can be forged).

Over the past year, the main cyberattacks on information systems were: the use of trojans and droppers, in particular, when transferring traffic between clients and services of an automated system. The types of hacker attacks remain classic: persistent and passive Cross-Site Scripting, SQL Injection and Cross-Site Request Forgery. The main data protection mechanisms of cyberattacks have been analyzed in detail.

In addition, to ensure the secure processing of personal data, the use of blockchain technology with the implementation of the DI mechanism was analyzed during the design of SOA. The example code in C# has been written.

REFERENCES

[1] Stevens, W.P.; Myers, G.J. and LeRoy Constantine, L.: *Structured design*. IBM Systems Journal **13**(2), 115-139, 1974, http://dx.doi.org/10.1147/sj.132.0115,

[2] Yourdon, E. and Constantine, L.L.: *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Yourdon Press, 1979,

[3] –: *ISO/IEC/IEEE 24765:2010 Systems and software engineering*.

[4] Laplante. P.A.: *What Every Engineer Should Know about Software Engineering*. CRC Press, pp.105-106, 2007, http://dx.doi.org/10.1201/9781420006742,

[5] Johnson, R.E. and Foote, B.: *Designing Reusable Classes*.
Journal of Object-Oriented Programming **1**(2), 22-35, 1988,
[6] Mattsson, M.: *Object-Oriented Frameworks: A survey of methodological issues*. Ph.D. Thesis.
Lund University, Lund, 1996,
[7] Fayaza F.: *Service oriented architecture in enterprise application*.
Technical Report, Dept. Inf. Technol., Univ. Moratuwa, Katubedda, 2014,
[8] Cisco Umbrella: *The modern cybersecurity landscape: Scaling for threats in motion*.
https://umbrella.cisco.com/info/technical-paper-modern-security-landscape-scaling-threats-motion,
[9] Cisco Umbrella: *Global cyber threat intelligence and trends*.
https://umbrella.cisco.com/trends-threats/global-cyber-threat-intelligence-overview,
[10] Gupta, B.B. and Chaudhary, P.: *Cross-Site Scripting Attacks: Classification, Attack, and Countermeasures (Security, Privacy, and Trust in Mobile Communications).*
CRC Press, 2020,
http://dx.doi.org/10.1201/9780429351327,
[11] Galluccio, E.; Caselli, E. and Lombari, G.: *SQL Injection Strategies: Practical techniques to secure old vulnerabilities against modern attacks*.
Packt Publishing, 2020,
[12] Wenz, C.: *ASP.NET Core Security*.
Black & White, 2022,
[13] Hayes, A.: *Blockchain Facts: What Is It, How It Works, and How It Can Be Used*.
https://www.investopedia.com/terms/b/blockchain.asp#citation-38,
[14] Panwar, A.; Bhatnagar, V.; Khari, M.; Salehi, A.W. and Gupta G.: *A Blockchain Framework to Secure Personal Health Record (PHR) in IBM Cloud-Based Data Lake*.
Computational Intelligence and Neuroscience, No. 3045107, 2022,
http://dx.doi.org/10.1155/2022/3045107,
[15] Zinchenko, A.Y.: *Design of distributed information systems based on using the technology of loosely coupled components*.
Systems and Technologies **63**(1), 5-14, 2023,
[16] Zinchenko, A.: *quickstart*.
https://github.com/artem-zinchenko.