УДК 004.41 DOI: 10.18698/2541-8009-2020-07-626

ПРОЦЕСС РАЗРАБОТКИ TCP-СЕРВЕРА ДЛЯ ЗАЩИЩЕННОГО ОБМЕНА ДАННЫМИ МЕЖДУ ПРИЛОЖЕНИЕМ И БАЗОЙ ДАННЫХ

А.В. Марченко mart0n@mail.ru
SPIN-код: 2567-4813

А.В. Ванин vaninav@student.bmstu.ru
SPIN-код: 2224-2566

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Показан процесс разработки TCP-сервера, который служит посредником между приложением, установленным на стороне клиента и базой данных, установленной на сервере. Задача разрабатываемого сервера состоит в обработке запросов, взаимодействия с базой данных и отправка полученного результата приложению. Данный механизм повышает эффективность защиты приложения от процесса декомпиляции, поскольку в коде клиентского приложения отсутствует описание названий полей и структура базы данных, вероятность утечки информации сводится к минимуму. Описанная выше возможность достигается благодаря тому, что запросы к серверу содержат только названия команд и идентификатор сессии, а наименование полей и структура базы данных описаны непосредственно в ядре TCP-сервера.

Ключевые слова

TCP-сервер, защищенность, обмен, данные, база данных, приложение, запрос, эффективность, разработка, программирование, взаимодействие

Поступила в редакцию 20.05.2020

Введение. В рамках статьи рассмотрены и описаны основные ключевые особенности работы и компоновки сервера. Приложение и все исходные коды для удобства размещены на хостинге GitHub [1].

Разработка архитектуры взаимодействия между приложением и сервером для разработчика является актуальной, поскольку в силу простоты реализации для приложения нередко выбирают клиент-серверную архитектуру [2], которая обеспечивает прямой обмен. Однако за этим скрывается опасность, связанная с таким явлением, как декомпиляция [3] или дизассемблирование приложения. Данный процесс позволяет потенциальному злоумышленнику разобрать исполняемый файл приложения на составляющие тексты исходных кодов [3], проанализировать их и выявить уязвимости приложения.

Огромный интерес в клиент-серверных приложениях представляет информация о базе данных, ее структуре, наименованиях таблиц и типах данных, содержащихся в таблицах. При выборе клиент-серверной архитектуры с прямым взаимодействием между приложением и базой данных разработчик описывает логику взаимодействия средствами языка программирования, на котором ве-

дется разработка, и SQL-запросов [4], позволяющих создавать запросы к нужным таблицам базы данных и извлекать из них информацию (рис. 1).

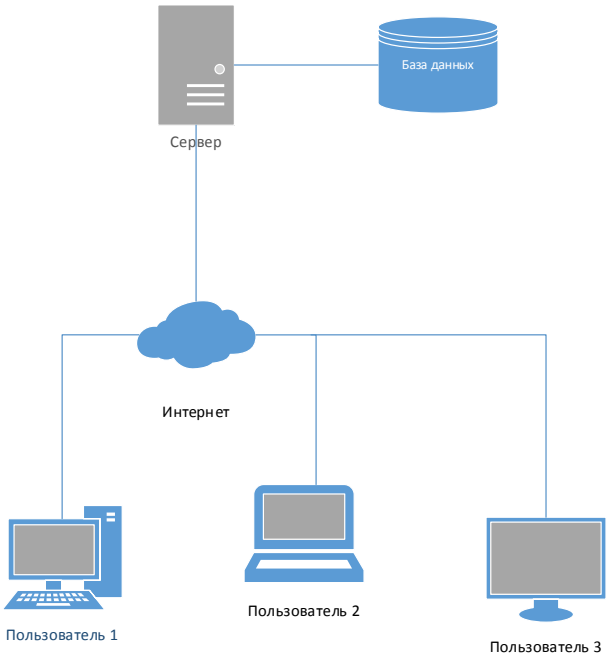

Рис. 1. Архитектура взаимодействия приложения с базой данных

При создании запроса к таблице указывают ее название и наименование интересующих пользователя столбцов. Поскольку запросов достаточно много, злоумышленник, воспользовавшийся процессом декомпиляции, получает полную информацию о структуре базы данных, типах переменных и логике ее работы. Полученные сведения он может использовать для извлечения информации, хранимой в базе данных, и ее последующей реализации в корыстных целях.

Описание работы TCP-сервера. Разработка TCP-сервера (Transmission control protocol — протокол управления передачей) позволит затруднить или свести на нет возможность получения сведений о базе данных злоумышленником благодаря описанию логики взаимодействия с базой данных в ядре сервера. В этом случае в процессе анализа исходных кодов злоумышленник будет видеть только названия запросов, которые получает сервер. Стоит понимать, что необходимо обеспечить защиту самого TCP-сервера, для этого требуется контролировать открытие и закрытие клиентских сессий, внедрить блокировку IP-адресов [5], которая применяется при защите от перебора паролей, обеспечить историю обращений пользователей к серверу (логирование) [6], а также установить на сервер необходимые средства защиты информации: антивирус и средства обнаружения вторжений. Схема работы данной архитектуры показана на рис. 2. На данной схеме TCP-сервер показан в виде виртуальной машине, однако на практике для экономии материальных ресурсов он запускается непосред-

ственно на самом сервере, где находится база данных. Ключевым моментом для создания данной архитектуры является разработка процесса авторизации, который и будет контролировать начало и окончание сессии.

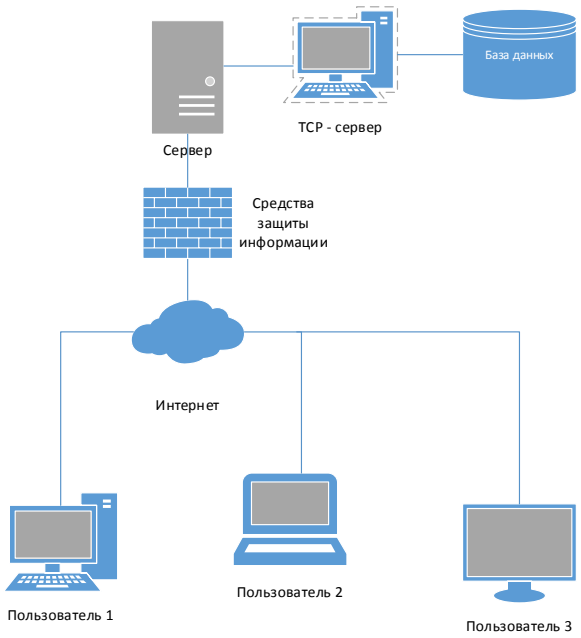

Рис. 2. Защищенная архитектура с TCP-сервером

Разработка клиентской части. При разработке TCP-сервера используется язык программирования C#. В рамках демонстрации работы системы сервер и клиент находятся на одном компьютере. Для создания соединения с сервером в языке C# используется библиотека Socket [7], которая находится по адресу System.Net.Sockets.

Основными настройками для установления соединения служат IP-адрес и порт TCP/IP [9] подключения. Всего существует три типа портов:

1) системные (0 – 1023);

2) пользовательские (1024 – 49 151);

3) частные (динамические) (49 152 – 65 535).

Для TCP-сервера выбирают порт из списка «Пользовательские», например 8888. Поскольку сервер запускается на текущем компьютере, IP-адрес является локальным (рис. 3).

```
static int port = 8888; //порт подключения
static string address = "127.0.0.1"; //адрес сервера
```

Рис. 3. Настройки подключения к серверу

Опишем функцию отправки запроса на сервер и получения ответа. Запрос будет иметь следующий формат: {текущая сессия: имя запроса, параметры за-

проса} (1234: CheckUser, login, pass). Двоеточие и запятую удобно использовать для разбиения строки и обработку их по отдельности.

Преимущество функции при программировании заключается в возможности ее многократного использования с различными параметрами вызова, поэтому описываемая функция принимает значение параметра-строки (рис. 4).

```
//функция отправки запроса на сервер
public string request(string dataRequest) //параметр типа string
{
    //инициализация переменной для создания подключения с сервером
    TcpClient client = null;

    try
    {
        //задаём настройки подключения
        client = new TcpClient(address, port);

        //выполняем подключение
        NetworkStream stream = client.GetStream();
```

Рис. 4. Описание функции

Класс TcpClient отвечает за создание клиента для соединения с сервером, а класс NetworkStream создает подключение.

Между отправкой запросов на сервер и получением ответа проходит определенный промежуток времени, который зависит от качества интернет-соединения и загруженности самого сервера. Чтобы предотвратить разрыв соединения, необходимо использовать «бесконечный» цикл, который прерывается только после получения ответа от сервера (рис. 5).

```
//остаёмся подключенными до тех пор, пока не закрыто соединение
while (true)
{
    //преобразуем сообщения
    byte[] data = Encoding.UTF8.GetBytes(dataRequest);

    //отправка сообщения
    stream.Write(data, 0, data.Length);

    //получаем ответ
    data = new byte[64]; //буфер для получаемых данных

    //собирательная строка
    StringBuilder builder = new StringBuilder();
    int bytes = 0;

    //получаем ответ порциями по 64 байта
    do
    {
        bytes = stream.Read(data, 0, data.Length);
        builder.Append(Encoding.UTF8.GetString(data, 0, bytes));
    }
    while (stream.DataAvailable);

    //собираем порции в единый ответ
    result = builder.ToString();
```

Рис. 5. Цикл соединения

```
        //проверяем наличие ошибок
        if (result == "Error")
            continue;
        else
            return (result);
    }
```

Рис. 6. Обработчик ошибок

Передача данных на сервер осуществляется в байтовом формате [8] с кодировкой UTF-8 [9]. Ответ с сервера приходит порциями по 64 байта, конвертируется в строку и собирается в результат. Результат необходимо проверять на наличие ошибок с сервера, если сервер отправит код “Error”, необходимо повторить запрос. Для этого применяют обработчик ошибок, описанный на рис. 6.

Другой обработчик контролируется конструкцией {try … catch} и служит для определения доступности сервера. Обычно его применяют при отсутствии подключения к сети Интернет или профилактических работах на сервере. Во втором случае можно использовать дополнительные коды ошибок, чтобы предупреждать пользователей. Описанный обработчик ошибок представлен на рис. 7.

```
}
catch (Exception ex) //обработка исключений
{
    return "no_connect: \n" + ex;
}
finally
{
    client.Close(); //закрытие соединения
}
```

Рис. 7. Определение доступности сервера

Если отправка и получение сообщений прошли успешно, соединение необходимо закрыть с помощью конструкции finally.

Шифрование паролей. Неотъемлемой частью для осуществления безопасной передачи данных (в нашем случае — пароля) между сервером и базой данной, является алгоритм SHA-256, с помощью которого вычисляется хеш-функция пароля. Данная процедура сильно затрудняет расшифровку пароля для злоумышленника, так как количество всех возможных вариантов паролей составляет 2^{256}. В настоящее время подобрать пароль с помощью перебора всех вариантов невозможно. Исключением является выбор пользователем слабого или распространенного пароля, в этом случае скомпрометировать пароль не составит большого труда и не займет много времени.

Разработка базы данных. Для демонстрации работоспособности создадим базу данных, содержащую в себе четыре таблицы:

1) Users — хранит в себе логины и пароли пользователей системы;

2) Session — хранит в себе время начала и окончания рабочей сессии с приложением;

3) Bans — содержит список заблокированных пользователей с указанием причины;

4) Log — содержит журнал действий всей системы.

Наглядная структура созданной базы данных показана на рис. 8.

Разработка серверной части. Существуют два режима работы сервера [10]:

1) синхронный. Особенностью данного режима является обработка клиентов в порядке очереди. Главный недостаток такого режима — ожидание следующим клиентом завершения обслуживания предыдущего;

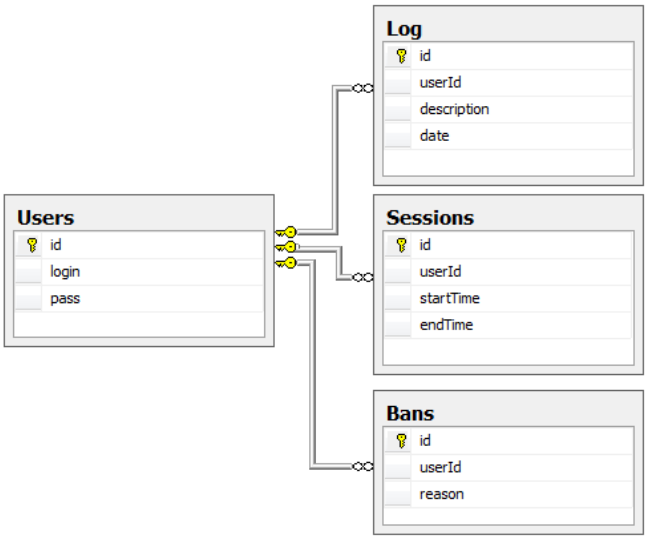

Рис. 8. Структура базы данных

2) асинхронный. Данный режим позволяет организовать работу сразу с несколькими клиентами, при этом следующий клиент подключается сразу, не ожидая окончания работы предыдущего.

Работа TCP-клиента должна быть асинхронной, так как разрабатываемые приложения обычно используются для широкого круга лиц. За работу TCP-сервера в языке C# отвечает библиотека Sockets, которая располагается в Microsoft Visual Studio по адресу using System.Net.Sockets.

Зададим настройки сервера, они должны совпадать с настройками, указанными при создании клиентского приложения (рис. 9).

```
static int port = 8888; //порт подключения
static string ipaddress = "127.0.0.1"; //IP адресс
static TcpListener listener;
```

Рис. 9. Настройки подключения

Класс TcpListener содержится в библиотеке Sockets и служит для обнаружения подключений. В приведенном на рис. 10 коде показан алгоритм работы сервера в асинхронном режиме.

За распараллеливание работы отвечает класс Thread, который создает отдельный поток для каждого подключившего клиента. Его работа включает:

– контроль подключений и отключений пользователей;

– обработку запросов;

– отправку ответов.

Конструктор класса ClientObject состоит из одной строки и отвечает за описание подключаемых клиентов подключаемых клиентов (рис. 11).

```
static void Main(string[] args)
{
    try
    {
        listener = new TcpListener(IPAddress.Parse(ipaddress), port);

        //запуск слушателя
        listener.Start();

        Console.WriteLine("Ожидание подключений...");

        while (true)
        {
            //получаем входящее подключение
            TcpClient client = listener.AcceptTcpClient();
            ClientObject clientObject = new ClientObject(client);

            //создаём новый поток для обслуживания клиента
            Thread clientThread = new Thread(new ThreadStart(clientObject.ProcessTCP));
            clientThread.Start();
        }

    }
    catch (Exception ex)
    {
        Console.WriteLine("Ошибка: \n{0}", ex);
    }
    finally
    {
        if (listener != null)
            listener.Stop();
    }
}
```

Рис. 10. Описание работы сервера

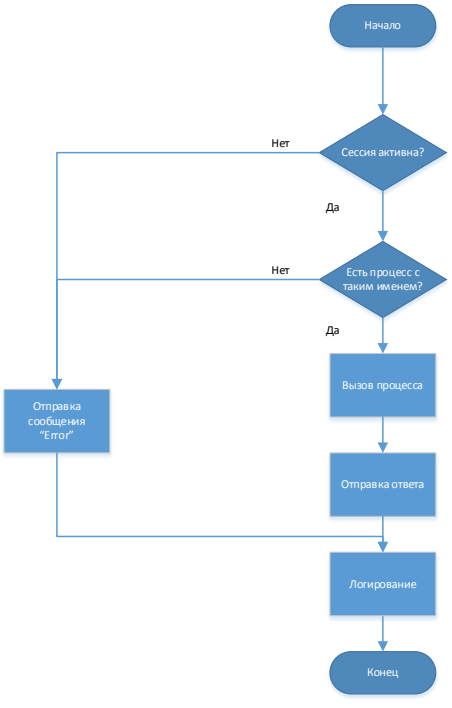

Рис. 12. Алгоритм работы обработчика

```
public TcpClient client;

//конструктор класса
public ClientObject(TcpClient tcpClient)
{
    client = tcpClient;
}
```

Рис. 11. Описание класса ClientObject

Основную задачу по обработке запросов берет на себя функция ProcessTCP, которая собирает запрос, анализирует его содержимое, передает информацию в обработчик и отправляет ответ. Она имеет структуру, аналогичную функции, которая отправляет запрос от клиента к серверу.

Рассмотрим структуру обработчика запросов. В его задачу входят: логирование, проверка активной сессии клиента, сопоставление имени запроса с вызовом функции. Для наглядности алгоритм работы обработчика и его структура показаны на рис. 12 и 13.

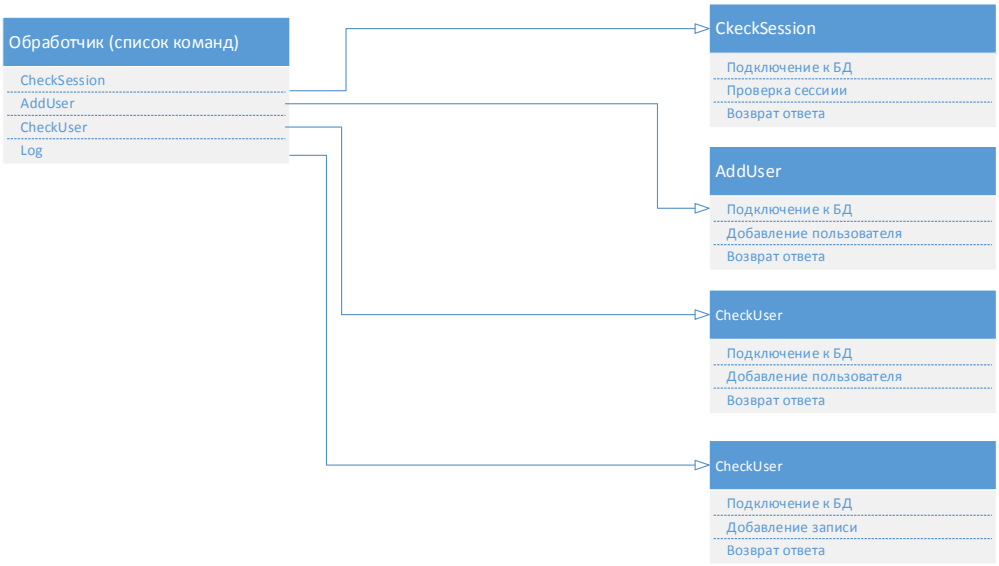

Рис. 13. Структура обработчика

Модульная структура обработчика позволяет удалять, изменять и добавлять функции, независимо друг от друга, не разрывая при этом связи.

Демонстрация работы. В данной статье для демонстрации работы системы создано простое приложение, содержащее в себе главную форму, форму авторизации, форму регистрации, окно успешной авторизации и окно блокировки. При запуске программы открывается главная форма с минимальным функционалом, который позволяет выполнить авторизацию или регистрацию (рис. 14).

Для получения доступа к функциям программы необходимо выполнить процесс регистрации, нажав кнопку «Регистрация» и введя имя пользователя и пароль (рис. 15).

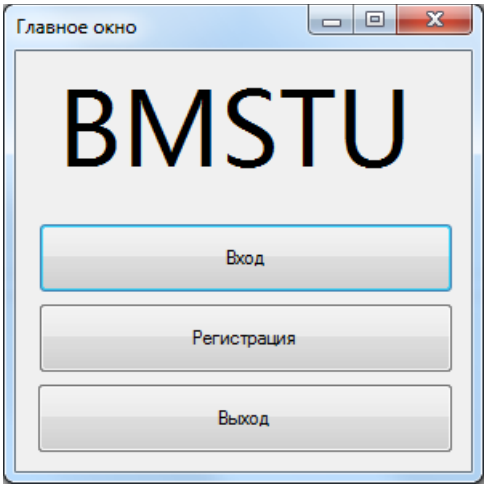

Рис. 14. Главная форма программы

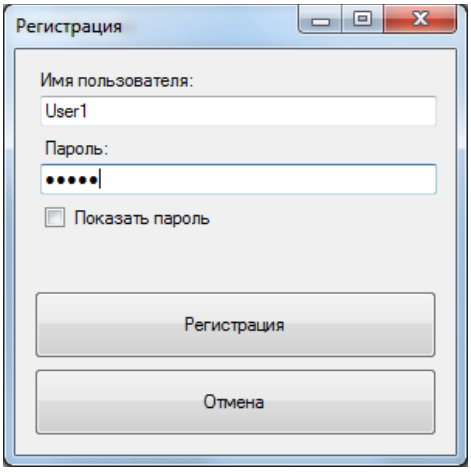

Рис. 15. Форма регистрации

Если данное имя пользователя занято, то программа отобразит сообщение об этом (рис. 16).

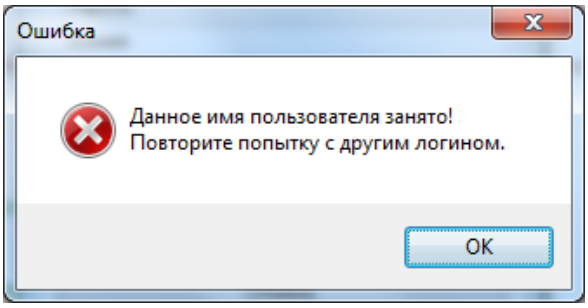

Рис. 16. Ошибка «Неуникальное имя пользователя»

После успешной регистрации для входа в приложение необходимо авторизоваться, ввести логин и пароль в форме авторизации (рис. 17).

Форма авторизации содержит необходимые проверки логина и пароля, которые выполняются на TCP-сервере. В случае успешного прохождения проверки открывается доступ к информационной системе, а в базе данных создается сессия, которая завершается в момент закрытия программы.

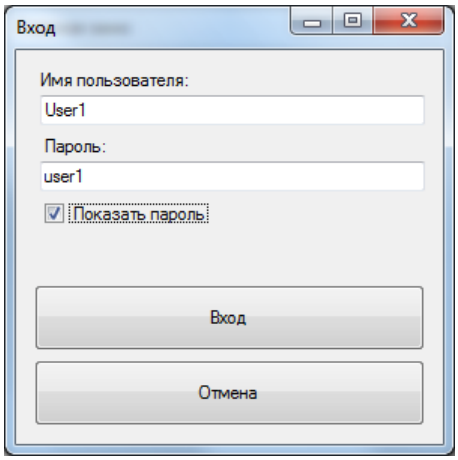

Рис. 17. Форма авторизации

Выводы. В данной статье продемонстрирован процесс разработки защищенного обмена данных между сервером и приложением с помощью TCP-сервера, который позволяет не только скрыть структуру базы данных от злоумышленника в процессе декомпиляции, но и распределить вычислительные мощности. В рамках статьи демонстрируется реализация защиты пароля с использованием криптографического алгоритма хеширования SHA-256, но разработчик может выбрать любой другой, в том числе и их комбинацию. Исходные данные описанного проекта выложены на хостинг GitHub и будут полезны при параллельном ознакомлении с настоящей статьей. Одной из основных перспектив по улучшению защиты информации является разработка сквозного шифрования и обмена ключами при передаче большого количества данных.

Литература

[1] AppTest. *github.com: веб-сайт*. URL: https://github.com/Maratan/AppTest (дата обращения: 26.02.2020).

[2] Рожин С.Н., Чикалов А.Н. Создание клиент-сервера для программы тестирования. *Труды Северо-Кавказского филиала МТУ связи и информатики*, 2016, № 1, с. 546–549.

[3] Красов А.В., Шариков П.И. Методика защиты байт-кода Java-программы от декомпиляции и хищения исходного кода злоумышленником. *Вестник СПбГУ технологии и дизайна. Серия 1. Естественные и технические науки*, 2017, № 1, с. 47–50.
[4] Ган И.Б. Оптимизация запросов SQL SERVER. *Windows IT Pro*, 2016, № 1, с. 44.
[5] Клочек М.С., Перфенова А.С. IP-адресс. *Инновационное развитие*, 2018, № 1(18), с. 17–18.
[6] Зачем нужно логирование. *javarush.ru: веб-сайт*. URL: http://javarush.ru/groups/posts/2293-zachem-nuzhno-logirovanie (дата обращения: 26.02.2020).
[7] Липатов В.С., Рукосуев А.С. TCP/IP сокеты. *Научные исследования*, 2016, № 9(10), с. 15–17.
[8] Байт. Что это такое. *chto-eto-takoe.ru: веб-сайт*. URL: https://chto-eto-takoe.ru/byte (дата обращения: 26.02.2020).
[9] UTF-8: Кодирование и декодирование. *habr.com: веб-сайт*. URL: https://habr.com/ru/post/138173/ (дата обращения: 26.02.2020).
[10] Когда использовать асинхронный или синхронный AJAX. *internet-technologies.ru: веб-сайт*. URL: https://www.internet-technologies.ru/articles/kogda-ispolzovat-asinhronnyy-ili-sinhronnyy-ajax.html (дата обращения: 26.02.2020).

Марченко Антон Васильевич — студент кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Ванин Александр Владимирович — студент кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Научный руководитель — Левиев Дмитрий Олегович, старший преподаватель кафедры «Информационная безопасность», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Ссылку на эту статью просим оформлять следующим образом:
Марченко А.В., Ванин А.В. Процесс разработки TCP-сервера для защищенного обмена данными между приложением и базой данных. *Политехнический молодежный журнал*, 2020, № 07(48). http://dx.doi.org/ 10.18698/2541-8009-2020-07-626

PROCESS OF DEVELOPING A TCP SERVER FOR SECURE COMMUNICATION BETWEEN AN APPLICATION AND A DATABASE

A.V. Marchenko mart0n@mail.ru
SPIN-code: 2567-4813

A.V. Vanin vaninav@student.bmstu.ru
SPIN-code: 2224-2566

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

The process of developing a TCP server is shown, which serves as an intermediary between the application installed on the client side and the database installed on the server. The task of the server being developed is to process requests, interact with the database and send the resulting result to the application. This mechanism increases the efficiency of protecting the application from the decompilation process, since the code of the client application does not contain a description of the field names and the structure of the database, the probability of information leakage is minimized. The above possibility is achieved due to the fact that requests to the server contain only command names and session identifier, and the names of fields and database structure are described directly in the TCP server core.

Keywords

TCP server, security, exchange, data, database, application, request, efficiency, development, programming, interaction

Received 20.05.2020

References

[1] AppTest. *github.com: website*. URL: https://github.com/Maratan/AppTest (accessed: 26.02.2020).

[2] Rozhin S.N., Chikalov A.N. Creating client-server for testing program in C#. *Trudy Severo-Kavkazskogo filiala MTU svyazi i informatiki*, 2016, no. 1, pp. 546–549 (in Russ.).

[3] Krasov A.V., Sharikov P.I. Methods of protection byte code Java-programs from decompilation and theft of source code by an attacker. *Vestnik SPbGU tekhnologii i dizayna. Seriya 1. Estestvennye i tekhnicheskie nauki* [Vestnik of St. Petersburg State University of Technology and Design. Series 1. Natural and technical sciences], 2017, no. 1, pp. 47–50 (in Russ.).

[4] Gan I.B. query optimization of SQL SERVER. *Windows IT Pro*, 2016, no. 1, pp. 44 (in Russ.).

[5] Klochek M.S., Perfenova A.S. IP-address. *Innovatsionnoe razvitie*, 2018, no. 1(18), pp. 17–18 (in Russ.).

[6] Zachem nuzhno logirovanie [Why do you need logging]. *javarush.ru: website* (in Russ.). URL: http://javarush.ru/groups/posts/2293-zachem-nuzhno-logirovanie (accessed: 26.02.2020).

[7] Lipatov V.S., Rukosuev A.S. TCP/IP sockets. *Nauchnye issledovaniya*, 2016, no. 9(10), pp. 15–17 (in Russ.).

[8] Bayt. Chto eto takoe [Bite. What is it]. *chto-eto-takoe.ru: website* (in Russ.). URL: https://chto-eto-takoe.ru/byte (accessed: 26.02.2020).

[9] UTF-8: Kodirovanie i dekodirovanie [UTF-8: coding and decoding]. *habr.com: website* (in Russ.).URL: https://habr.com/ru/post/138173/ (accessed: 26.02.2020).

[10] Kogda ispol'zovat' asinkhronnyy ili sinkhronnyy AJAX [When to use asynchronous and synchronous AJAX]. *internet-technologies.ru: website* (in Russ.). URL: https://www.internet-technologies.ru/ articles/kogda-ispolzovat-asinhronnyy-ili-sinhronnyy-ajax.html (accessed: 26.02.2020).

Marchenko A.V. — Student, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Vanin A.V. — Student, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Scientific advisor — Leviev D.O., Senior Lecturer, Department of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation.

Please cite this article in English as:
Marchenko A.V., Vanin A.V. Process of developing a TCP server for secure communication between an application and a database. *Politekhnicheskiy molodezhnyy zhurnal* [Politechnical student journal], 2020, no. 07(48). http://dx.doi.org/ 10.18698/2541-8009-2020-07-626.html (in Russ.).