Chat assíncrono criptografado utilizando sockets TCP em Java

Sila Geroges Agiru Judick Siebert
UDESC

Wagner Luis Sousa da Luz
UDESC

November 26, 2016

Abstract

Neste artigo, uma aplicação de bate-papo para enviar mensagens criptografadas é proposta. O algoritmo de criptografia é caracterizado por inverter valor dos bits da mensagem. A aplicação é desenvolvida usando a linguagem de programação Java e utilizando sockets TCP. Este artigo resume as etapas de engenharia de software seguidas durante a implementação deste projeto.

1 Introdução

Aplicativos de mensagens instantâneas tornaram-se populares sendo usados diariamente pelas pessoas. A maioria dos usuários convencionais na Internet não percebe que suas conversas estão sendo transmitidas em texto claro e são vulneráveis a espionagem durante a transmissão. O projeto foi intitulado chat assíncrono criptografado e seu objetivo principal é implementar uma sala de bate-papo com criptografia nas mensagens. Objetivos secundarios foram a pesquisa sobre sockets e experiência prática dos autores no desenvolvimento de uma aplicação bate-papo baseada em Java utilizando conceitos de threads e sockets aprendidos em aula.

1.1 Requisitos

Nesta seção, descrevemos alguns dos requisitos gerais do nosso aplicativo de sala de bate-papo distribuído.

- a criptografia deve inverter valor dos bits da mensagem.
- Devem utilizar uma conexão TCP/IP
- não utilizar a classe bufferedreader e bufferedwriter

A figura 1 representa uma sala de bate-papo.

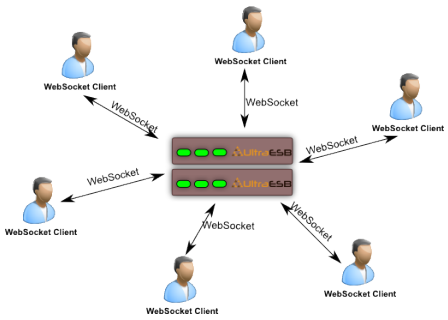

Figure 1: Sala de bate-papo

2 A sua Implementação

2.1 Visao geral

2.1.1 Diagramas de classe

A figura 2 representa o diagrama de classe da nossa implementação.

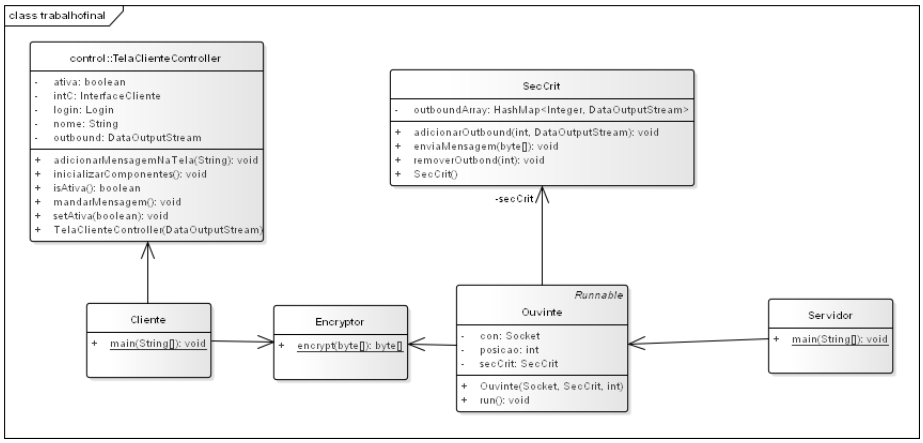

Figure 2: Class diagram

2.1.2 Diagramas de atividade

A figura 3 representa o diagrama de atividade da classe servidor da nossa implementação.

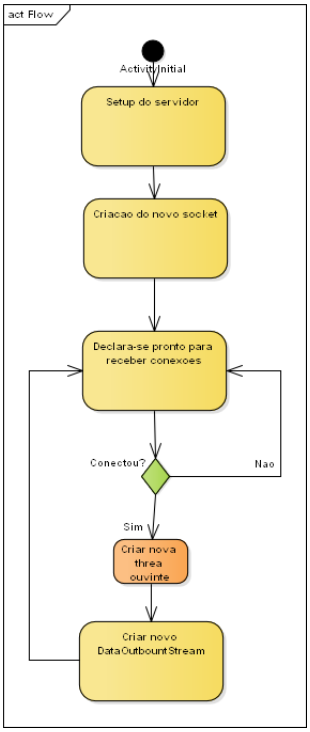

Figure 3: Activity diagram do servidor

2.2 Cliente

A tela do cliente deve ter um campo de texto no qual mensagens são exibidas à medida que chegam, e um campo para o usuário digitar mensagens de saída. Cada mensagem deve ter a hora em que foi enviada,o nome do remetente, a mensagem decriptada e uma versão encriptada de mensagem. Qualquer número de pessoas deve ser capaz de participar.

2.3 Codigo fonte

```
public class Servidor {

    public static void main(String[] args) {
        ServerSocket serverSocket;
        SecCrit b = new SecCrit();
        int numConexao = 0;
        try {
            Socket clientSocket;
            serverSocket = new ServerSocket(6666);
            while (true) {
                // declara-se pronto receber conexoes e bloqueia ate recebe-las
                clientSocket = serverSocket.accept();
                (new Thread(new Ouvinte(clientSocket, b, numConexao))).start();
                            DataOutputStream outbound = new
                                ↪ DataOutputStream(clientSocket.getOutputStream());
                b.adicionarOutbound(numConexao,outbound );
                numConexao++;
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

}
```

Figure 4: Codigo da classe Servidor

```
public class Cliente {

    public static void main(String[] args) {

        try {
            Socket clientSocket = new Socket("127.0.0.1", 6666);
            DataInputStream inbound = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream outbound = new DataOutputStream(clientSocket.getOutputStream());
            TelaClienteController tela = new TelaClienteController(outbound);
            String mesagemRecebida = "";
            String mensagemEncryptada;
            byte[] arrayMensagem = {};
            do {

                if (inbound.available() > 0) {
                    System.out.println("Cliente start with :" + inbound.available() + " avaible
                        ↪ bytes");
                    arrayMensagem = new byte[inbound.available()];
                    for (int i = 0; i < arrayMensagem.length; i++) {

                        arrayMensagem[i] = inbound.readByte();
                        System.out.println("Byte received");
                    }
                    // ler mensagens recebidas
                    mensagemEncryptada = new String(arrayMensagem);
                    arrayMensagem = Encryptor.encrypt(arrayMensagem);
                    mesagemRecebida = new String(arrayMensagem);
                    if (!mesagemRecebida.isEmpty()) {
                        System.out.println("Cliente recebeu mensagem " + mesagemRecebida);

                        tela.adicionarMensagemNaTela(mesagemRecebida + " ----> " +
                            ↪ mensagemEncryptada);
                    }
                }
            } while (tela.isAtiva());
```

Figure 5: Codigo da classe Cliente

```
public static byte[] encrypt(byte[] mensagem) throws UnsupportedEncodingException {

    byte[] encriptada = new byte[mensagem.length];
    for (int i = 0; i < mensagem.length; i++) {
        encriptada[i] = (byte) ~mensagem[i];
    }
    return encriptada;
}
```

Figure 6: Código da classe Encryptor

3 Avaliação da implementação

Capacidade e limitacoes da applicacao. Durante a implementação algumas dificuldades forma surgindo. A primeira deles sendo como manter todas as mensagens sincronizadas e na ordem em todas as janelas de batepapo. Outra dificuldade identificada foi reenviar as mensagens enviados de um cliente para o servidor, para todos os outros clientes. A ultima dificuldade foi implementar a criptografia. Para isto algumas estrategias foram tentadas, nenhuma funcionando ate finalmente trabalhar somente com vetores de bytes.

4 Conclusão

Síntese do que foi dito.
Lista dos resultados atingidos:

- resultado 1
- resultado 2 e teste de bibliografia [1]

Conclusão final e Trabalho Futuro.

References

[1] Thomas Bulfinch. *O Livro De Ouro Da Mitologia: Histórias De Deuses E Heróis*. Editora Ediouro, 1998.