به نام خدا

گزارش امنیت پروژه تیم 13

مقدمه

برای امن کردن ارتباط بین سرور-بانک از RSA استفاده کردیم به طوری که در اتصال اول هردو طرف کلید عمومی را برای یکدیگر ارسال میکنند. برای سرور-کلاینت مسیج با استفاده از توکن امضا میشود. همینطور برای امنیت بیشتر مسیج ها برای 2 ثانیه معتبر هستند.در ضمن تمهیدات زیر برای وقتی هست که کد سرور و دیتابیس ها پابلیک نباشند ولی کد کلاینت به کل در دسترس کاربر باشد.

Reply Attacks

برای جلوگیری از این اتفاق ما چند انتخاب داشتیم که به شیوه های مختلف زمان رو در پیام ارسال شده دخیل کنیم. بنابر این ساده ترین راه را انتخاب کردم که آن هم افزودن رشته ای خاص به اول پیام ارسالی. به این صورت که کلاینت قبل فرستادن هر مسیج یک پیام به سرور میدهد و زمان سرور را میگیرد. حال مسیجی که کلاینت به سرورش میفرستد شامل رشته ای هست که حاوی زمان گرفته شده از سرور میباشد. سرور وقتی مسیج را میگیرد دو حالت وجود دارد. یا مسیج به فرمت @getTime@0 هست که در اون صورت زمان را به کلاینت برمیگرداند؛ در حالت بعدی مسیج به صورت time@message هست که در اون صورت تایم رو چک میکنه با تایم خودش، اگر بیشتر از 2000 میلی ثانیه اختلاف داشته باشه مسیج رو پراسس نمیکنه و ارور برمیگرداند.حال اگر پیام ها شنود شده باشن و ذخیره شده باشن (دیکود نشده باشن با توجه به تعریف نوع اتک) ، بعد از 2 ثانیه غیر قابل استفاده خواهند بود. مستندات در زیر اورده شده اند.این فرایند برای هردو ارتباط بین کلاینت-سرور و سرور-بانک انجام شده.

```java
public void sendMessageToServer(String message) {
    this.message = message;
    String message1 = getTheEncodedMessage("@@getTime@");
    try {
        dataOutputStream.writeUTF(message1);
        dataOutputStream.flush();
        String string = "";
            try {
                string = dataInputStream.readUTF();
            } catch (IOException e) {
                e.printStackTrace();
            }
            getMessageFromServer(string);
        message = getTheEncodedMessage(time + message);
        dataOutputStream.writeUTF(message);
        dataOutputStream.flush();
        String string1 = "";
        do {
            try {
                string1 = dataInputStream.readUTF();
            } catch (IOException e) {
                e.printStackTrace();
            }
            getMessageFromServer(string1);
        } while (dataInputStream.available() > 0);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```java
public void processMessage(String message, DataOutputStream dataOutputStream, Socket socket) {
    if (TokenGenerator.getInstance().isTokenVerified(message, dataOutputStream)) {
        ServerController.getInstance().passTime(dataOutputStream);
        message = TokenGenerator.getInstance().getTheDecodedMessage(message);
        System.out.println(message);
        long date = 0;
            Pattern pattern = Pattern.compile("(\\d+)@.*");
            Matcher matcher = pattern.matcher(message);
            if(matcher.find()) {
                String date1 = matcher.group(1);
                message= message.substring(date1.length());
                date = Long.parseLong(date1);
            }
        if (new Date().getTime() - date > 2000 && !message.startsWith("@getTime@")) {
            ServerController.getInstance().sendMessageToClient( message: "@Error@InvalidMessage", dataOutputStream);
            return;
        }
```

```java
public String handleBankConnection(String data) {
    String response = null;
    try {
        dataOutputStream.writeUTF( str: "@@getTime@");
        dataOutputStream.flush();
        String time = dataInputStream.readUTF();
        dataOutputStream.writeUTF( str: time + "@" + data);
        dataOutputStream.flush();
        response = dataInputStream.readUTF();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return response;
}
```

```java
private String processMessage(String command) throws IOException {
    String response = "";
    long date = 0;
    Pattern pattern = Pattern.compile("(\\d+)@.*");
    Matcher matcher = pattern.matcher(command);
    if (matcher.find()) {
        String date1 = matcher.group(1);
        command = command.substring(date1.length());
        if (command.charAt(0) == '@') {
            command = command.substring(1);
        }
        date = Long.parseLong(date1);
    }
    if (new Date().getTime() - date > 2000 && !command.startsWith("@getTime@")) {
        return "@Errors@InvalidMessage";
    }
```

Brute force

خوب برای جلوگیری از این حمله از پیشنهاد داک استفاده کردیم بدین صورت که هر سوکت با وصل شدن به سرور در هش مپی ثبت میشود که سوکت را به ایپی اش مپ میکند. و همینطور هش مپی outputstream سوکت را به سوکت مپ میکند. و یک هش مپ داریم به اسم errorCounterForIp که هر ایپی را به اررِی لیستی از جنس لانگ مپ میکند. حال هر مسیجی که از سمت کلاینت دریافت میشود اگر جوابش شامل Error@باشد به اررِی لیست مپ شده توسط این ایپی در errorCounterForIp ، یک عدد long که مقدار ان برابر است با new Date().getTime() . حال برای ارسال ریسپانس به کلاینت با هر ارور یک عدد به این اررِی لیست اضافه میشود و چک میشود که اگر طول این اررِی لیست از 10 بیشتر باشد فاصله زمانی ده تای اخر چک میشود و اگر از 1 ثانیه کمتر باشد، همه کلاینت هایی که این ایپی دارند موقتا به مدت 5 دقیقه بن میشوند و با ارسال مسیج به سرور فقط ارور دریافت میکنند. دوباره این برای هر دو ارتباط کلاینت-سرور و سرور-بانک صادق هست.(در عکس ها، اخرین تغییرات زمان حداقلی آپدیت نشده است.)

```
private void startProcess() {
    try {
        serverSocket = new ServerSocket( port: 8080);
    } catch (IOException e) {
        e.printStackTrace();
    }
    while (true) {
        System.out.println("Waiting for Client...");
        Socket socket;
        try {
            socket = serverSocket.accept();
            InetSocketAddress sockaddr = (InetSocketAddress) socket.getRemoteSocketAddress();
            InetAddress inaddr = sockaddr.getAddress();
            Inet4Address in4addr = (Inet4Address) inaddr;
            String ip4string = in4addr.toString();
            socketIp.put(socket, ip4string);
            if(!ipDosChecker.containsKey(socketIp.get(socket)))
            ipDosChecker.put(socketIp.get(socket), new ArrayList<Long>());
            if(!errorCounterForIp.containsKey(socketIp.get(socket)))
                errorCounterForIp.put(socketIp.get(socket), new ArrayList<>());
        } catch (IOException e) {
            break;
        }
        System.out.println("Client Connected!!!");
        Socket finalSocket = socket;
        new Thread(() -> {
```

```
if(message.startsWith("@Error")&&!temporaryBlackList.containsKey(socketIp.get(socketDataOutputStreamHashMap.get(dataOutputStream)))){
    errorCounterForIp.get(socketIp.get(socketDataOutputStreamHashMap.get(dataOutputStream))).add(new Date().getTime());
    if(checkBruteForce(socketIp.get(socketDataOutputStreamHashMap.get(dataOutputStream)))) {
        codedMessage = TokenGenerator.getInstance().getTheCodedMessage(dataOutputStream, data: "@Error@You are banned for 5 minutes.");
        try {
            dataOutputStream.writeUTF(codedMessage);
            dataOutputStream.flush();
            return;
        } catch (IOException e) {
            System.out.println("Error in Sending Packets...");
            try {
                dataOutputStream.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

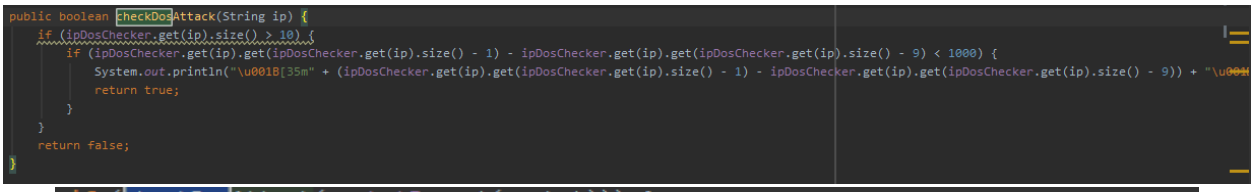
```
private void handleClientsConnection() {
    try {
        clientsServerSocket = new ServerSocket( port: 3030);
    } catch (IOException e) {
        e.printStackTrace();
    }
    while (true) {
        try {
            Socket socket = clientsServerSocket.accept();
            InetSocketAddress sockaddr = (InetSocketAddress) socket.getRemoteSocketAddress();
            InetAddress inaddr = sockaddr.getAddress();
            Inet4Address in4addr = (Inet4Address) inaddr;
            String ip4string = in4addr.toString();
            socketIp.put(socket, ip4string);
            if (!ipDosChecker.containsKey(socketIp.get(socket)))
                ipDosChecker.put(socketIp.get(socket), new ArrayList<Long>());
            if (!errorCounterForIp.containsKey(socketIp.get(socket)))
                errorCounterForIp.put(socketIp.get(socket), new ArrayList<>());
            new Thread(() -> handleClientRequests(socket)).start();
        } catch (IOException e) {
            System.out.println("Error in Client's Acceptance ...");
            break;
        }
```

```
if(response.startsWith("@Error")&&!temporaryBlackList.containsKey(socketIp.get(socketDataOutputStreamHashMap.get(dataOutputStream)))){
    errorCounterForIp.get(socketIp.get(socketDataOutputStreamHashMap.get(dataOutputStream))).add(new Date().getTime());
    if(checkBruteForce(socketIp.get(socketDataOutputStreamHashMap.get(dataOutputStream)))) {
        try {
            dataOutputStream.writeUTF(response);
            dataOutputStream.flush();
            return;
        } catch (IOException e) {
            System.out.println("Error in Sending Packets...");
            try {
                dataOutputStream.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

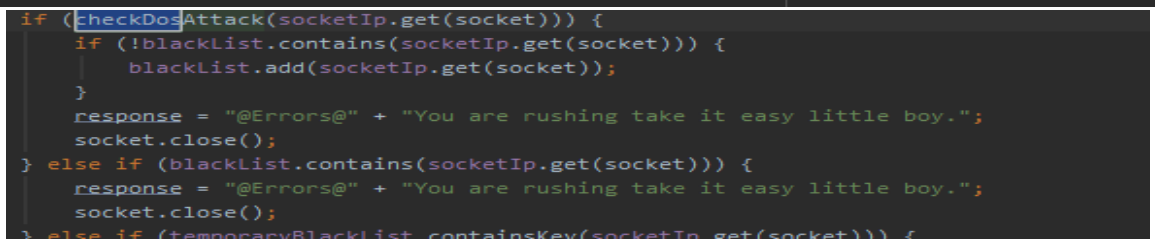
Dos

برای این مشکل، همانند مشکل قبلی از گفته های داک استفاده کردیم و تعداد مسیج های دریافتی از طرف یک آی‌پی را محاسبه میکنیم و اگر بالای ده بود چک میکنیم ده تا از اخرین مسیج ها در فاصله زمانی

کمتر از 175 میلی ثانیه فرستاده نشده باشند در غیر اینصورت آن آی پی به صورت دایمی بن خواهدشد. باز هم برای کلاینت-سرور و سرور-بانک برقرار هست.(در عکس ها آخرین تغییرات زمان حداقلی اعمال نشده است.)



```
public boolean checkDosAttack(String ip) {
    if (ipDosChecker.get(ip).size() > 10) {
        if (ipDosChecker.get(ip).get(ipDosChecker.get(ip).size() - 1) - ipDosChecker.get(ip).get(ipDosChecker.get(ip).size() - 9) < 1000) {
            System.out.println("\u001B[35m" + (ipDosChecker.get(ip).get(ipDosChecker.get(ip).size() - 1) - ipDosChecker.get(ip).get(ipDosChecker.get(ip).size() - 9)) + "\u001
            return true;
        }
    }
    return false;
}
```



```
if (checkDosAttack(socketIp.get(socket))) {
    if (!blackList.contains(socketIp.get(socket))) {
        blackList.add(socketIp.get(socket));
    }
    response = "@Errors@" + "You are rushing take it easy little boy.";
    socket.close();
} else if (blackList.contains(socketIp.get(socket))) {
    response = "@Errors@" + "You are rushing take it easy little boy.";
    socket.close();
} else if (temporaryBlackList.containsKey(socketIp.get(socket))) {
```

Broken Authentication

برای این مشکل، محافظت از اطلاعات هویتی(نام کاربری ها و رمزها) مهم ترین نکته در برنامه هست. که به چند بخش تقسیم میشود. یک اینکه که با استفاده از پسوورد لیست سعی کنند وارد شوند. که خوب برای محافظت از این مورد اگر کلاینت بیشتر از پنج بار رمز غلط وارد کند به بلاک لیست سرور اضافه میشود و نمیتواند به سرور وصل شود. دو اینکه پیام‌هایی که به سرور فرستاده میشوند و در جواب شامل دیتای مهمی هستند. اولی getAllUser هست. که این مسیج وقتی فرستاده میشود چک میشود که کلاینتی که این پیام را فرستاده، به عنوان مدیر لاگین کرده باشد، در غیر اینصورت برنامه ارور برمیگرداند. دومی getUser هست. با اینکه این دستور برای همه کاربر ها در دسترس هست ولی چک میشود که کسی که این پیام را به سرور میفرستد کاربری که اطلاعات آن را میخواهد با همان کاربر لاگین کرده باشد در غیر اینصورت ارور دریافت میکند. سومی getAllRequests هست که میتواند شامل اطلاعات فروشنده های در جریان ثبت نام یا درخواست تغییر اطلاعات کاربران که مثل getAllUsers سیستم چک میکند طرف مقابل مدیر باشد. بغیر از این مسیج ها، اطلاعاتی که کاربر میتواند بگیرد لیست محصولات، حراج ها، مزایده ها و .. هست که دیتای اساسی برنامه نیست. البته در مسیج پراسسور با توجه به هر مسیج چک میشه یوزر اجازه داره یا نه. و خوب بعضی از مسیج ها نیازی ندارن حتما یوزری لاگین کرده باشد.

Sql injection

در خصوص جلوگیری از نفوذ به پایگاه داده یا همان sql injection ما در این پروژه سعی کردیم برای پرس و جو های خود یا همان Query ها از اشیای prepared statement استفاده کنیم. مزیت بزرگ نسبت به statement این است که جلوی نفوذ به پایگاه داده را می گیرد بدین صورت که به جای مقادیر مورد نیاز از علامت ? استفاده می شود تا در صورتی که حتی کاربر با استفاده از شروط همواره صحیح در زبان sql هم نتواند به تمامی داده ها از طریق زبان sql دست پیدا کند بدین صورت دسترسی افراد به تمامی داده های ذخیره شده در پایگاه داده محدود می شود.