امنیت

پیش گفتار :

پیش از شروع به توضیح نحوه رفع آسیب پذیری در حالت های مختلف باید به این نکته توجه کرد که کلیه پیام های رد و بدل شده رمزگذاری شـده اند. در ابتدای اتصـال، هر کلاینت با سـرور تبادل رمز عمومی می کنند و سـپس با اسـتفاده از شـیوه رمزگذاری RSA کلیدی متقارن را با یکدیگر به ا شتراک می گذارند. لازم به ذکر ا ست کلید متقارن هر کلاینت متفاوت ا ست و تنها می تواند پیام‌های دریافتی خود رو رمزگشایی کند. پس رمزگشایی پیام هر کلاینت برای هر شخص دیگر به جز آن تقریبا غیر ممکن است.

نحوه رفع Reply Attacks

همراه با هردرخواستی که کلاینت به سرور می‌فرستد زمان دقیق آن ذخیره می‌شود. هنگامی که پیام برای سرور فرستاده می‌شود به منظور اضافه شدن به صف پیام‌های دریافتی پیام باید ازمایش تابع زیر را رد کند. و در صورتی که زمان ار سال پیام با زمان دریافت آن فا صله قابل توجهی داشته باشند یا زمان معتبری ثبت نشده باشد. تابع مقدار false را باز می گرداند و پیام نادیده گرفته می‌شود.

```
private boolean checkReplayAttacks(String message) {
    Message receivedMessage = JsonConverter.fromJson(message, Message.class);
    long seconds = Duration.between(receivedMessage.getDate(), LocalDateTime.now()).toSeconds();
    return receivedMessage.getDate() != null && seconds <= MAXIMUM_REQUEST_TIME;
}
```

نحوه رفع Improper Inputs

برای جلوگیری از مختل شدن روند برنامه با ورودی های نامنا سب دو تدبیر اندیشیده شده که از خطر احتمالی جلوگیری کند. از آنجا که پیام ها به روش نا متقارن رمزگذاری شده اند تقریبا غیر ممکن است که فرد مهاجم بتواند پیامی بفرستد که پس از رمزگشایی دارای معنی باشد. اما با این حال تابعی در نظر گرفته شده که پیام رمز گشایی شده را چک میکند که حتما در قالب JSON باشد. همچنین سایز دستور ورودی نباید از مقدار ماکسیمم تعیین شده بیشتر باشد.

```
private boolean checkImproperInputs(String message) {
    return message.length() < MAXIMUM_INPUT_SIZE && parseChecker(message);
}
```

```
private boolean parseChecker(String message){
    Stack<Character> characterStack = new Stack<>();
    char[] chars = message.toCharArray();

    for (char character : chars) {
        if (character == '{')
            characterStack.push(character);
        if (character == '}' && characterStack.peek()=='{')
            characterStack.pop();
    }

    return  characterStack.isEmpty();
}
```

نحوه رفع Brute force attack

به منظور رفع این نوع حمله، به هر کلاینت تعداد دفعاتی که تلاش به لاگین کرده اما ناموفق بوده است نسبت داده شده است. حال اگر تعداد دفعات این رخداد از تعداد ماکسیمم تعیین شده بیشتر شود کاربر به مدت دلخواه (۲ دقیقه) در لیست سیاه سرور اضافه می‌شود و تمام درخواست های او در این مدت نا دیده گرفت می‌شود. پس از سپری شدن این مدت کاربر دوباره می‌تواند برای لاگین کردن تلاش کند.

```
private boolean checkBruteForce(String client) {
    AtomicInteger numberOfAttempt = failedTries.get(client);
    if (numberOfAttempt == null || numberOfAttempt.get() <= MAXIMUM_BRUTE_FORCE_ATTACKS)
        return true;
    else {
        blackList.add(client);
        // Removing Client Name From Blacklist After 2 Minutes
        new Thread(() -> {
            try {
                sleep( millis: SECONDS_TO_REMAIN_ON_BLACKLIST * 1000);
                blackList.remove(client);
                failedTries.replace(client, new AtomicInteger( initialValue: 0));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
        return false;
    }
}
```

نحوه رفع DoS

به منظور جلوگیری از این نوع حملات به هر کلاینت دو چیز مپ شده است. زمان اتصال آن به سرور و تعداد درخواست‌های ارسال شده او. حال پس از هر بار دریافت پیام مقدار میانگین این تعداد پیام بر ثانیه چک می‌شود و اگر از مقدار مجاز بیشتر بود، شخص برای مدتی وارد لیست سیاه شده و درخواست های آن در این مدت نادیده گرفته می شود. همچنین از آنجا که این درخواست ها در زمان کم انجام می شود زمان اولین اتصال هر ۱ دقیقه به مقدارجدید خود یعنی زمان حال ریست می‌شود.

```
private boolean checkDenialOfService(String client) {
    LocalDateTime firstConnection = connectionTime.get(client);
    if (firstConnection == null) return false;
    if (numberOfRequests.get(client) == null) return true;
    int requests = numberOfRequests.get(client).get();
    long seconds = Duration.between(firstConnection, LocalDateTime.now()).toSeconds();
    if ((requests / (seconds + 1)) <= MAXIMUM_REQUESTS_PER_SECOND)
        return true;
    else {
        blackList.add(client);
        // Removing Client Name From Blacklist After 2 Minutes
        new Thread(() -> {
            try {
                sleep( millis: SECONDS_TO_REMAIN_ON_BLACKLIST * 1000);
                blackList.remove(client);
                numberOfRequests.replace(client, new AtomicInteger( initialValue: 0));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
        return false;
    }
}
```

نحوه رفع Broken Authentication

برای جلوگیری از این نوع آسیب ها علاوه بر رمزگذاری آن ها تدبیری دیگر نیز اندیشیدیم. به این گونه که در آن نوع درخواست هایی که نیاز به تایید هویت شخص بود در ابتدای اجرای درخواست دارا بودن شرایط کلاینت را چک می کنیم. برای مثال برای لاگ اوت کردم در ابتدا چک می شود که کلاینت مورد نظر وارد اکانت خود شده باشد. در صورت دارا نبودن شرایط در هر کدام از این موارد یک اکسپشن به کلاینت پس فرستاده می‌شود. مانند مثال زیر :

```
public void acceptAddAuctionRequest(Message message) throws ClientException {
    loginCheck(message);
    if (message.getSender() == null) {
        throw new ClientException("invalid message!");
    } else {
        Account account = getAccount(message.getSender());
        AddAuctionRequest addAuctionRequest = message.getAcceptAddAuctionRequestMessage().getAddAuctionRequest();
        if (addAuctionRequest == null) {
            throw new ClientException("null message!");
        } else if (!(account instanceof Manager)) {
            throw new ClientException("You are not allowed to do that");
        } else {
            addAuctionRequest.accept();
            Server.getInstance().addToSendingMessages(Message.makeDoneMessage(message.getSender()));
            Server.getInstance().serverPrint( string: "Request got accepted!");
        }
    }
}
```

جمع بندی

به جز مورد آخر که در تابع مربوطه چک می‌شود بقیه موارد قبل از اضافه شدن پیام به لیست پیام های دریافتی سنجیده می‌شود و در صورت عدم موفقیت حتی یکی از این تست ها پیام کلاینت نادیده گرفته می‌شود. به صورت زیر :

```
private boolean validate(String client, String message) {
    return !(blackList.contains(client)) && checkReplayAttacks(message) && checkBruteForce(client) &&
            checkImproperInputs(message) && checkDenialOfService(client);
}
```

```
void addMessage(String clientName, String encryptedMessage) throws Exception {
    String message = Server.decryptSymmetric(encryptedMessage, symmetricKeyMap.get(clientName));
    if (validate(clientName, message)) {
        Server.getInstance().addToReceivingMessages(Message.convertJsonToMessage(message));
        numberOfRequests.replace(clientName, new AtomicInteger(numberOfRequests.get(clientName).incrementAndGet()));
        Message temp = JsonConverter.fromJson(message, Message.class);
        if (temp.getMessageType().equals(MessageType.LOGIN))
            failedTries.replace(clientName, new AtomicInteger(failedTries.get(clientName).incrementAndGet()));
    }
```

همچنین توابع رمز گذاری و رمزگشایی پیام در کلاس Server موجود است اما از آنجا که توضیح آن مفصل است به تشریح آنها نمی‌پردازیم.