گزارش گام‌های برداشته شده برای ایمنی نرم‌افزار

تیم۳: علیرضا ایجی، امیررضا میرزایی، آرمان سلیمانی

حملات SQL Injection :

از آنجایی که نرم‌افزار از دیتابیس SQLite استفاده می‌کند، برای جلوگیری از این نوع حملات چندین اقدام از سمت سرور صورت گرفته است. اولین نکته، نحوه استخراج اطلاعات از دیتابیس است که در کلاس Database.java انجام می‌شود. برای این کار، هنگامی که کاربر قصد لاگین کردن داشته باشد، روش معمول این است که هنگام انجام query روی دیتابیس، نام کاربری و رمز عبور هردو بررسی شوند، که نقطه ضعف بزرگی محسوب می‌شود و یکی از رایج‌ترین راه‌ها برای دسترسی به حساب سایر کاربران است.

در این نرم‌افزار، در query صرفا نام کاربری استفاده می‌شود و چنانچه کاربری با این نام یافت شود، اطلاعات او به شکل یک شی از نوع User وارد کلاس UserController شده و با equals. بررسی می‌شود که آیا رمز عبور صحیح است یا خیر.

اما راهکار مهم‌تر، Input Sanitization است. برای جلوگیری از حملات مخرب، اطلاعاتی که قرار باشد در دیتابیس سیو شوند با سختگیری گزینش می‌شوند. نام‌های کاربری، رمز، اسم کالاها، و غیره، امکان استفاده از هیچ کاراکتر خاصی مانند فاصله، ستاره، کوتیشن و دابل کوتیشن و ... را ندارند و سرور هنگام برخورد با چنین مواردی پیام ارور به کلاینت ارسال می‌کند تا کاربر فیلد ارسالی را مجددا بازنویسی کند. این sanitization حتی در مواردی مثل کامنت‌های کاربران نیز حضور دارد تا امکان حملاتی که قصد DUMP یا غیره را دارند نیز گرفته‌شود.

```
String allData = username.concat(password).concat(name).concat(lastName).concat(email);
if(allData.contains("*") || allData.contains("|") || allData.contains("'") || allData.contains("\"") || allData.contains(" ")){
    return "Error: invalid characters!";
}
```

مشکلات ناشی از Improper Inputs :

برای حالات مختلف ورودی‌های غیرمجاز اقدامت مختلفی انجام شده‌است.

برای هر دستوری که ارسال می‌شود علاوه بر ارسال خود دستور طبق فرمت سرور، یک عدد تحت عنوان type نیز ارسال می‌شود. چنانچه دستور ارسال شده دارای type صحیح متناظر با آن نباشد یا دستور در لیست دستورات سرور موجود نباشد یک اکسپشن پرتاب می‌شود که در کلاس سرور هندل می‌گردد. در صورتی که این اتفاق رخ دهد، آدرس IP ارسال‌کننده وارد بلک‌لیست می‌گردد و هرگز از آن خارج نمی‌شود (بر خلاف برخی موارد که بعد از مدتی خارج می‌شوند) زیرا ناشناخته بودن دستور به این منزله است که از کلاینت ما ارسال نشده است.

برای مقابله با حالتی که کاربران قصد اشغال فضای سرور را داشته باشند، سرور برای ورودی هایی که از جنس رشته هستند محدودیت‌هایی را لحاظ می‌کند (مثلا مشخصات افراد حداکثر ۱۸ کاراکتر، کامنت‌ها حداکثر ۱۲۰۰ کاراکتر و ...) و عکس‌هایی که حجم بسیار زیادی داشته باشند را نمی‌پذیرد.

برای این که دستورات توسط اشخاص ثالث ذخیره و بعدا ارسال نشوند، در کنار هر دستور یک timestamp قرار می‌گیرد. سرور فقط پیام‌هایی که در یک دقیقه اخیر ارسال شده باشند را بررسی می‌کند.

```
try{
   LocalDateTime timestamp = LocalDateTime.parse(commandJson.get("timestamp").getAsString());
   if(LocalDateTime.now().isBefore(timestamp) || LocalDateTime.now().isAfter(timestamp.plusMinutes(1))){
      return "Error: invalid command";
   }
}catch (Exception e){
   return "Error: invalid command";
}
```

مقابله با Broken Authentication :

برای مقابله با چنین مشکلاتی، راه‌های بسیار متنوعی وجود دارد زیرا قسمتی از آن به نحوه عملکرد کاربر در مواجهه با برنامه و قسمتی دیگر به امنیت شبکه در مقابل حمله‌کننده‌ها بستگی دارد.

اولین گام برداشته شده این است که کاربر هنگام بستن نرم‌افزار و یا هر اقدامی که منجر به قطع شدن ارتباط او گردد، خود به خود log out می‌شود تا در سناریوهایی مانند رها کردن نرم‌افزار در کافی‌نت یا ... امکان دسترسی سایرین از حساب او از بین برود.

همچنین برای کاهش امکان دسترسی افراد نامربوط به حساب کاربری فرد، سرور امکان افتتاح حساب با رمز عبورهای ضعیف (صرفا عدد و بدون کاراکتر، طول کم و دارای مشابهت بالا با نمونه پسوردهای ضعیف مانند password یا password123 یا ...) را به کاربر نمی‌دهد.

علاوه بر این راهکار Inconclusive Error نیز استفاده شده است، که در صورت عدم صحت اطلاعات وارد شده هنگام لاگین، این که کدام قسمت از اطلاعات نادرست است به کاربر گفته نمی‌شود.

در نهایت برای افزایش ایمنی، سرور برای هر توکنی که برای کاربران لاگین شده در نظر می‌گیرد، زمان صدور آن را ثبت کرده و پس از مدت معینی (در هنگام نوشتن گزارش این مدت برابر ۳۶۰۰ ثانیه است) آن توکن را نامعتبر می‌کند. کاربر پس از این مدت باید مجددا لاگین کند.

```
ArrayList<String> deleteTheseTokens = new ArrayList<>();
for(String token:tokenMillis.keySet()){
    if(clock.millis() - tokenMillis.get(token) > 3600000){
        deleteTheseTokens.add(token);
    }
}
```

حملات brute force :

همانطور که در قسمت قبل ذکر شد، برای افزایش ضریب ایمنی رمز عبورها، سرور طول آنها را بررسی کرده و آنها را با نمونه پسوردهای ضعیف مقایسه می‌کند. چنانچه رمز عبور ضعیف باشد امکان ثبت‌نام به کاربر داده نمی‌شود. این امر باعث می‌شود زمان مورد نیاز برای حمله brute force به شکل نمایی افزایش پیدا کند.

```
protected void addIPToLoginDetention(){
    if(IPsFailedLogIns.containsKey(userIP)){
        IPsFailedLogIns.replace(userIP,IPsFailedLogIns.get(userIP)+1);
    }
    else {
        IPsFailedLogIns.put(userIP,1);
    }
    if(IPsFailedLogIns.get(userIP) >= 3){
        disabledIPsTime.put(userIP,clock.millis());
        Server.addIPBlocked(userIP);
    }
}

public void updateLoginDetention(){
    for(int ip:disabledIPsTime.keySet()){
        if(clock.millis() - disabledIPsTime.get(ip) > 60000){
            Server.removeIPBlocked(ip);
        }
    }
}
```

همانطور که در قطعه کد بالا (برداشته شده از کلاس AuthTokenHandler در سرور) قابل مشاهده است، سرور تعداد لاگین های ناموفق به ازای هر IP را ذخیره می‌کند. در ابتدا اگر کاربر ۳ لاگین ناموفق داشته باشد (نه الزاما متوالی) به بلک‌لیست اضافه می‌گردد و ۶۰ ثانیه ارتباط او با سرور قطع می‌شود. پس از آن به ازای هر یک لاگین ناموفق کاربر بایستی ۶۰ ثانیه صبر کند (این روند عمدا تصاعدی است تا با brute force مقابله شود)

این امر در کنار شرایط انتخاب رمز عبور باعث می‌شود زمان مورد نیاز برای تشخیص رمز عبور توسط حمله‌کننده بسیار طولانی گردد (از مرتبه سال) و موضوعیت حمله از نوع brute force ناچیز شود.

حملات Replay :

با توجه به پیچیدگی این نوع حمله و امکان مقابله با راهکارهای امنیتی، برای مقابله با این نوع حمله گام‌های زیر برداشته شده‌اند.

همانطور که بالاتر اشاره شد، پیام‌های ارسالی به سرور دارای timestamp هستند و پس از مدتی اعتبار آنها از بین می‌رود.

```
try{
   LocalDateTime timestamp = LocalDateTime.parse(commandJson.get("timestamp").getAsString());
   if(LocalDateTime.now().isBefore(timestamp) || LocalDateTime.now().isAfter(timestamp.plusMinutes(1))){
      return "Error: invalid command";
   }
}catch (Exception e){
   return "Error: invalid command";
}
```

توکن‌هایی که در پیام‌ها قرار دارند نیز دارای انقضا هستند و پس از مدتی اعتبارشان را از دست می‌دهند. این راهکارها خود به خود تا حدی جلوی این حملات را می‌گیرند ولی به تنهایی کافی نیستند.

برای افزایش ایمنی، هنگام صدور توکن، آیپی‌ای که توکن را دریافت می‌کند ذخیره می‌شود. هنگام دریافت درخواست از سمت کلاینت، چنانچه توکن ارسالی معتبر باشد، سرور آیپی ارسال‌کننده را بررسی می‌کند. اگر این آدرس مطابق آیپی ذخیره شده از قبل نباشد، آیپی جدید وارد بلک‌لیست شده و از آن خارج نمی‌شود، و سرور امکان اتصال آن آیپی را از بین می‌برد.

```
public String getUserWithToken(String token){
   if(onlineUsersTokens.containsKey(token)){
      if(tokensIP.get(token)!=userIP){
         return null;
      }
      return onlineUsersTokens.get(token);
   }
   return null;
}
```

```
onlineUsersTokens.put(token, username);
tokenMillis.put(token,clock.millis());
onlineUsername.add(username);
tokensIP.put(token,userIP);
```

حملات DoS :

برای مقابله با این نوع حملات نیز همانطور که در داک اشاره شده‌است چنانچه از سمت کلاینتی به شکل رگباری درخواست ارسال شود کلاینت به بلک‌لیست آیپی ها اضافه می‌گردد.

```
if(suspiciousIPsConnection.containsKey(request.getLocalPort())) {
    if (suspiciousIPsConnection.get(request.getLocalPort()) > 1000) {
        blockedIp.add(request.getLocalPort());
        System.err.println("connection refused:too many");
        DoSBlackListTime.put(request.getLocalPort(),clock.millis());
        request.close();
        continue;
    }
}
if(clock.millis() - previousCommandMillis < 10 && previousIP==request.getLocalPort()){
    if(suspiciousIPsConnection.containsKey(request.getLocalPort())){
        int cnt = suspiciousIPsConnection.get(request.getLocalPort());
        suspiciousIPsConnection.replace(request.getLocalPort(),cnt, newValue: cnt+1);
    }else{
        suspiciousIPsConnection.put(request.getLocalPort(),1);
    }
}
updateDoSList();
```

البته محدودیت‌های لحاظ شده در حال حاضر زیاد هستند و چنانچه کاربری از کلاینت به شکل عادی ولی با سرعتی بسیار بالا استفاده کند (مدام روی قسمت های مختلف نرم‌افزار کلیک کند و درخواست های زیادی ارسال کند) او نیز وارد بلک‌لیست می‌گردد تا عملکرد بهینه سرور تضمین گردد.

همچنین با توجه به ساختار stateless سرور، چنانچه سرور مورد هدف این حملات قرار گیرد تاثیر مخربی روی سرور قرار نمی‌گیرد اما امکان خدمت‌رسانی به کلاینت ها تا حدی کاهش می‌یابد، و هدف از اقدامات طی شده مقابله با این کاهش خدمت‌رسانی است.