به نام خدا

عنوان پروژه:

امنیت

نویسندگان :

محمد مهدی ابوترابی

یاسمن زلفی موصلو

صادق مجیدی یزدی

گروه 5

Improper Inputs

در کارهای شبکه ممکن است که کلاینت یک درخواست اشتباه به سرور بفرستد و ممکن است در سرور ما به دلیل این که انتظار همچین ورودی را ندارد خطایی یا exception رخ دهد و سرور ما به مشکل برخورد. حال ما برای رفع این مشکل راهکارهایی را قرار داده‌ایم.

اولین کاری که انجام دادیم این است که کلاسی درون برنامه هم برای کلاینت هم برای سرور به نام RequestForServer وجود دارد که کلاینت برای درخواست دادن باید ابتدا یک شی از این کلاس بسازد و سپس آن را به جیسون تبدیل کرده و به سرور می‌فرستد.حال در سرور اگر رشته‌ی دریافت شده به شی این کلاس قابل تبدیل نباشد یک exception در سرور پرتاب میشود که catch می‌شود و سرور پیغام "not formatted input" را به سمت کلاینت می‌فرستد ولی خود down نمی‌شود و به کارکرد عادی خود ادامه می‌دهد.

```
RequestForServer requestForServer = null;
try {
    requestForServer = new Gson().fromJson(input, RequestForServer.class);
} catch (Exception exception) {
    try {
        dataOutputStream.writeUTF( str: "not formatted input");
        dataOutputStream.flush();
        return;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

حال اگر فرمت رشته ی ارسالی requestForServer بود به تابع handleRequest فرستاده میشود که آن جا بر اساس این که با چه کلاسی این درخواست کار دارد به یک تابع دیگر مخصوص آن کلاس فرستاده میشود یا اگر با کلاسی که وجود نداشت کار داشت بر اساس ساختار else if که در آن تابع وجود دارد باز پیغام "not a proper response" فرستاده به کلاینت میشود. اگر در هر کلاس هم با تابعی که در آن کلاس وجود نداشت کار داشت باز هم طبق ساختار-else if پیغام بالا نوشته میشود.اگر هم مثلا ورودی یه تابع باید از نوع long باشد ولی کلاینت درخواست بدی فرستاده باشد و ورودی درست نباشد در اون تابع exception رخ می‌دهد و این exception در نهایت در تابع اولیه catch شده و پیغام درتس نبودن به سمت کلاینت فرستاده میشود.پس هیچ جوره سرور ما خراب نمی‌شود.

```
        goodHandler(requestForServer);
    } else if (requestForServer.getController().equals("FilteringController")) {
        filteringControllerHandler(requestForServer);
    } else if (requestForServer.getController().equals("SortingController")) {
        sortingHandler(requestForServer);
    } else if (requestForServer.getController().equals("AccountAreaForSupporterController")) {
        accountAreaForSupporterHandler(requestForServer);
    } else if (requestForServer.getController().equals("AuctionsController")) {
        auctionControllerHandler(requestForServer);
    } else {
        dataOutputStream.writeUTF( str: "not a proper response");
        dataOutputStream.flush();
    }
```

```
try {
    handleRequest(requestForServer);
} catch (Exception e) {
    try {
        dataOutputStream.writeUTF( str: "exception occured in server");
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
    try {
        dataOutputStream.flush();
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
}
```

پس با ورودی اشتباه همانطور که دیدیم برای سرور ما اشتباهی رخ نمی‌دهد.حال اگر رشته ی ورودی فرستاده توسط کلاینت به سرور بزرگ باشد باز هم سرور همان اول برمی‌گردد و اجازه اشغال و کند شدن را نمیدهد.

در واقع همان اول چک می‌شود اگر طول رشته‌ی ارسالی بیشتر از 10000 بود سرور پیغام "not valid input" را به کلاینت می‌دهد و دیگر بر می‌گردد و الکی حافظه سرور ما اشغال نمی‌شود.

```
private boolean validSizeString(String input) {
    if (input == null)
        return false;
    if (input.length() > 10000)
        return false;
    return true;
}
```

```
if (!validSizeString(input)) {
    try {
        dataOutputStream.writeUTF( str: "not valid input");
        dataOutputStream.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return;
}
```

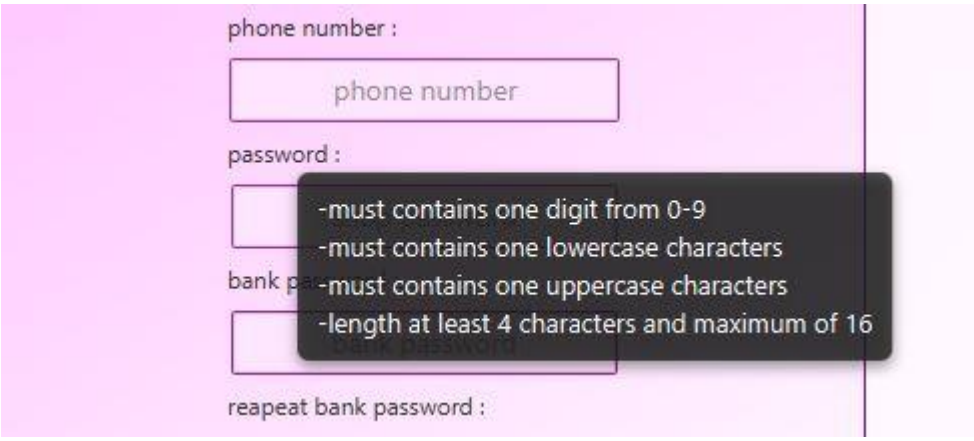
Replay Attacks

Reply attacks به این معنا است که یک فرد سومی پیغام‌های بین سرور و کلاینت را ذخیره کند و پس از مدتی آن را دوباره به سرور بفرستد که سرور نباید به این‌ها واکنش نشان دهد.خب یکی از راه حل‌ها این است که پیغام هایی که به سرور می‌فرستیم ، برای هر کدام یک expiredTime تعریف میشود که 10 دقیقه بعد از ارسال پیام از سمت کلاینت است و اگر از این تایم گذشته بود دیگر در سرور ما این پیام اعتباری ندارد. در کلاس requestForServer که در قسمت قبل به شما توضیح دادم یک فیلد لوکال تایم وجود دارد که در constructor آن هنگام ساخت تعیین میشود که 10 دقیقه پس از تایم الان هست.حال در سرور اگر رشته ی دریافتی قابل تبدیل به شی از این کلاس بود چک می‌شود که می‌شود که آیا از تایم انقضا گذشته است یا نه که اگر گذشته بود این پیغام معتبر نیست و به کلاینت این پیغام که این پیام دیگر معتبر نیست فرستاده می‌شود و آن درخواست اجرا نمی‌شود.

```
if (requestForServer.getExpireTime().isBefore(LocalTime.now())) {
    try {
        dataOutputStream.writeUTF( str: "your message is expired!");
        dataOutputStream.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return;
}
```

Broken Authentication

یکی از کارهایی که می‌توان برای حل این مشکل کرد این است که ما کاربر را مجبور می‌کنیم هنگام ثبت نام که از پسورد قوی استفاده کند.که این پسورد باید شامل حداقل یک عدد باشد.یک حرف کوچک و یک حرف بزرگ و همین‌طور باید شامل حداقل 4 حرف باشد. که این گونه پسورد کاربرهایمان قوی تر باشد و به نوعی حدس زدن آن سخت شود.



یکی از کارهای دیگه که انجام دادیم این است که وقتی کاربر لاگ اوت می‌کند دیگر توکنش توسط سرور منقضی می‌شود و اگر کسی با آن توکن به سرور درخواست بدهد پذیرفته نمی‌شود که خب این البته نرمال هست.همینطور اگر حتی کاربر لاگ اوت نکند و برنامه را ببنند هم توکن آن توسط سرور منقضی میشود. در واقع قبل از بسته شدن برنامه یک ریکوست مبنی بر لاگ اوت به سرور فرستاده میشود و هم آن کاربر از لیست کاربر های آنلاین حذف می‌شود هم توکنش منقضی می‌شود.

SQL injection

با توجه به این موضوع که در برنامه ما اتصال به دیتابیس، مپ کردن جداول به آبجکت ها و سایر کارهای مربوط به ارتباط برنامه و دیتابیس با استفاده از ORM قدرتمند JPA و هایبرنیت صورت میگیرد و تقریبا هیچ گونه کوئری به صورت دستی در برنامه نوشته نشده و تمام کوئری ها و تراکنش ها به صورت خودکار با استفاده از بیلدر های مربوط به JPA تولید و مورد استفاده قرار میگیرند، احتمال مورد تهاجم قرار گرفتن دیتابیس برنامه توسط کاربران حتی با وجود وارد کردن اطلاعات مخرب برای حذف و یا تخریب اطلاعات دیتابیس نیز عملا نزدیک به صفر شده و امنیت دیتابیس با درصد بالایی تامین می شود. اندک احتمالی که برای اینجکشن وجود دارد این است که در برنامه از کوئری های نوشته شده به صورت استرینگ و inline استفاده شود ولی چون در برنامه ما از این نوع کوئری ها بهره گرفته نشده این مورد نیز جای نفوذ ندارد. برای مثال به نمونه کوئری ساخته شده توسط jpa توجه کنید:

```
EntityManager entityManager = EntityManagerProducer.getInstanceOfEntityManager();
CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
CriteriaQuery<T> query = criteriaBuilder.createQuery(classOfT);
Root<T> root = query.from(classOfT);

query.select(root);
TypedQuery<T> typedQuery = entityManager.createQuery(query);
try {
    return typedQuery.getResultList();
} catch (NoResultException e) {
    System.out.println(e.getMessage());
    return new ArrayList<>();
} finally {
    entityManager.close();
}
```

که عملا اجازه نفوذ نمی دهد.