

My Vulnerable App

Niezabezpieczona aplikacja mobilna do testowania podatności na zagrożenia

Plan prezentacji

- Dane dotyczące bezpieczeństwa aplikacji mobilnych
- Bezpieczeństwo w systemie Android
- My Vulnerable App – kod, zastosowanie
- My Vulnerable App – znalezione podatności i ich załatanie
- Zadania

Statystyki dotyczące bezpieczeństwa aplikacji mobilnych

Wersja Androida



■ Starsza niż 2 lata - 82%

Szyfrowanie danych



■ Brak szyfrowania komunikacji - 35%

Hasło do urządzenia



■ Brak hasła - 43%

Liczba silnych zagrożeń



■ Przynajmniej jedno - 24,7%

Liczba zagrożeń bezpieczeństwa



■ Przynajmniej jedno - 50%

Porady dotyczące bezpieczeństwa aplikacji na Androida wg. Google

Pozwolenia (*permissions*)

- Minimalizacja ich liczby w celu uniknięcia nadużyć
- Przechowywanie danych w pamięci wewnętrznej zamiast zewnętrznej
- Ograniczenie liczby samodzielnie zdefiniowanych pozwoleń

Sprawdzenie danych wejściowych (*input validation*)

- Narzędzia zaimplementowane w platformę Android (np. `inputType`)
- Języki takie jak JavaScript czy SQL są bardziej podatne na zagrożenia z powodu *escape characters*
- Ręczna weryfikacja wprowadzanych danych

Ciąg dalszy tzw. *best practices*

- Prośba o podanie PINu przed dostępem do wrażliwych informacji
- Bezpieczne przechowywanie danych
- Walidacja danych z pamięci zewnętrznej
- Sprawdzanie dostępności nośnika pamięci zewnętrznej
- Aktualizacje zewnętrznych bibliotek
- Wybór najlepszych narzędzi kryptograficznych
- Testowanie aplikacji

Przerwa na prezentację kodu

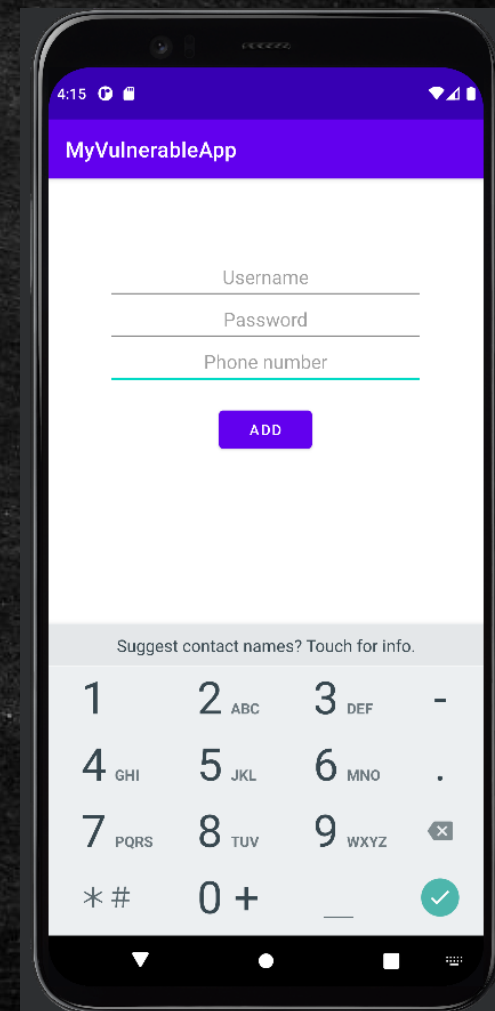
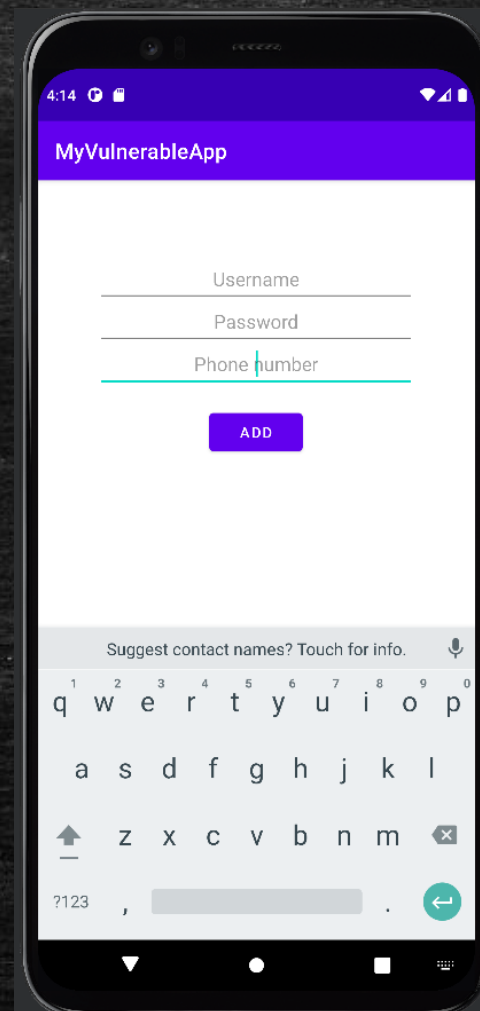
Podatności zostaną przedstawione wkrótce...

inputType i konsekwencje jego braku

```
android:inputType="textPassword"
        android:inputType="textPersonName"
android:inputType="phone"
```

```
E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.example.myvulnerableapp, PID: 5569
java.lang.NumberFormatException: For input string: "hhh"
    at java.lang.Integer.parseInt(Integer.java:615)
    at java.lang.Integer.parseInt(Integer.java:650)
    at com.example.myvulnerableapp.MainActivity.addButtonClicked(MainActivity.java:43)
    at com.example.myvulnerableapp.MainActivity$1.onClick(MainActivity.java:33)
    at android.view.View.performClick(View.java:7448)
    at com.google.android.material.button.MaterialButton.performClick(MaterialButton.java:1119)
    at android.view.View.performClickInternal(View.java:7425)
    at android.view.View.access$3600(View.java:810)
    at android.view.View$PerformClick.run(View.java:28305)
    at android.os.Handler.handleCallback(Handler.java:938)
    at android.os.Handler.dispatchMessage(Handler.java:99)
    at android.os.Looper.loop(Looper.java:223)
    at android.app.ActivityThread.main(ActivityThread.java:7656) <1 internal call>
    at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:592)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:947)
```

Dla hasła można także użyć pola Password zamiast EditText przy budowie layoutu aplikacji.



Sprawdzanie inputów ciąg dalszy

_username

ty

test

aLitwo! Ojczyzno moja! ty jesteś jak zdrowie! Ile cię trzeba cenić, ten tylko się dowie, Kto cię stracił. Dziś piękność twą w całej ozdobie Widzę i opisuję, bo tęsknię po tobie. Panno święta, co J...

```
asfasfpackage com.example.myvulnerableapp;import android.content.ContentValues;import android.content.Context;import android.database.Cursor;import al
```


Sql injection i inne zabawy z bazą danych

```
sql',inj','c'); DROP TABLE users;---
```

...

```
123123123
```

```
public void addUser(User user){  
  
    SQLiteDatabase db = getWritableDatabase();  
  
    String query = "INSERT INTO " + TABLE_USERS + "(" +  
        COLUMN_USERNAME + "," +  
        COLUMN_PASSWORD + "," +  
        COLUMN_PHONE +  
        ") VALUES (" + user.getUserName() + "," + user.getPassword() + "," + user.get_phone() + ");";  
  
    db.execSQL(query);  
    db.close();  
}
```

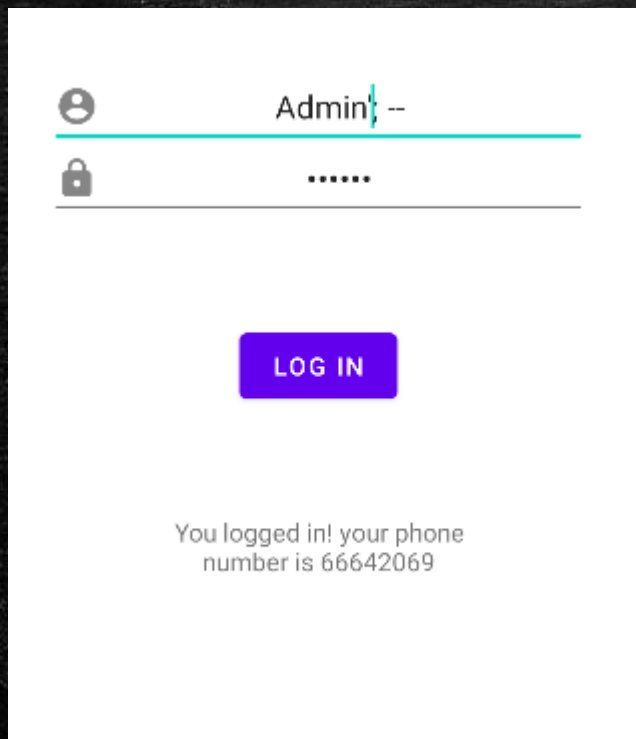
_phone

234

3

c

Sql injection i inne zabawy z bazą danych c.d.



Admin --

.....

LOG IN

You logged in! your phone number is 66642069

```
public String getPhone(String userName, String password){

    SQLiteDatabase sqLiteDatabase = getReadableDatabase();
    String query = "SELECT _phone FROM " + TABLE_USERS + " WHERE " + COLUMN_USERNAME + " = "
        + "'" + userName + "'" AND " + COLUMN_PASSWORD + " = '" + password + "'" ;";
    Cursor result = sqLiteDatabase.rawQuery(query, null);
    result.moveToFirst();
    if(result.getCount() < 1){
        return null;
    }else{
        return result.getString(0);
    }
}
```


Jak się przed nimi ustrzec

```
//METHOD TO ADD USER TO DB
public void addUser(User user){
    ContentValues values = new ContentValues();
    values.put(COLUMN_USERNAME, user.get_username());
    values.put(COLUMN_PASSWORD, user.get_password());
    values.put(COLUMN_PHONE, user.get_phone());
    SQLiteDatabase db = getWritableDatabase();
    db.insert(TABLE_USERS, nullColumnHack: null, values);
    db.close();
}
```

Funkcja insert automatycznie
binduje parametry do kolumn.

Jak się przed nimi ustrzec c.d.

```
public String getPhone(String userName){
    SQLiteDatabase sqLiteDatabase = getReadableDatabase();
    String query = "SELECT _phone FROM " + TABLE_USERS + " WHERE " + COLUMN_USERNAME + " = " + "'" + userName + "'";
    Cursor result = sqLiteDatabase.rawQuery(query, selectionArgs: null);
    result.moveToFirst();
    return result.getString( columnIndex: 0);
}

public boolean checkPassword(String password, String userName){
    SQLiteDatabase sqLiteDatabase = getReadableDatabase();
    String query = "SELECT _password FROM " + TABLE_USERS + " WHERE " + COLUMN_USERNAME + " = " + "'" + userName + "'";
    Cursor result = sqLiteDatabase.rawQuery(query, selectionArgs: null);
    result.moveToFirst();
    if(result.getCount()>0 && BCrypt.verifier().verify(password.toCharArray(), result.getString( columnIndex: 0)).verified){
        return true;
    }else{
        return false;
    }
}
```

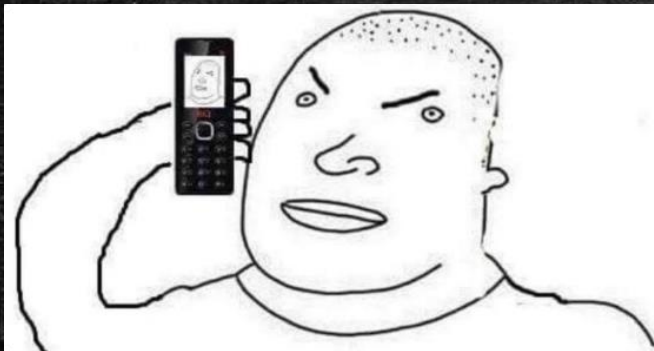
Zawsze należy hashować
hasło!

Zadanie 1 – Błąd zmęczonego programisty



- Ważne jest, aby w aplikacji, która korzysta z loginów użytkowników nie dało się zarejestrować dwóch osób o takiej samej nazwie użytkownika. Chociaż w przypadku MyVulnerableApp użytkowników rozróżnia się po unikalnym id, warto dodać funkcjonalność, która zablokuje zarejestrowanie dwóch takich samych nazw użytkownika. Osoba pisząca MyVulnerableApp o to zadbała, jednak w trakcie implementacji nieco się zmęczyła i popełniła błąd. Proszę znaleźć go i poprawić tak, aby działało rozróżnianie użytkowników także po ich nazwie.

Zadanie 2 – Ale to ty dzwonisz



- W polu na numer telefonu powinien się znajdować wyłącznie numer telefonu, przecież można potem na niego nawet zadzwonić. Atrybut `inputType` w pliku `xml` zadbał o to, aby można było wpisywać w pole 'Phone number' tylko niektóre znaki. To jednak za mało, żeby w pełni zabezpieczyć to pole przed niewłaściwymi danymi wejściowymi. Proszę odkryć podatność i ją zabezpieczyć. Ponadto, informacja o nieodpowiednich danych wejściowych powinna wyświetlić się użytkownikowi na ekranie.

Zadanie 3 – Ciemność, widzę ciemność



- Czasem już tak bywa, że człowiekowi się nic nie chce - nawet wpisać danych w pole z danymi wejściowymi. Trochę ciężko zadzwonić pod żaden numer telefonu, czy skontaktować się z nikim. Proszę o załatwienie tej luki, należy szczególnie zwrócić uwagę na pole z numerem telefonu. Ponadto, informacja o braku danych wejściowych powinna wyświetlić się użytkownikowi na ekranie.

Bibliografia

- <https://www.nowsecure.com/resource/infographic-surprising-stats-exposing-mobile-data-dangers>
- <https://developer.android.com/training/articles/security-tips>
- <https://developer.android.com/training/keyboard-input/style>
- <https://developer.android.com/topic/security/best-practices>
- <https://www.tripwire.com/state-of-security/security-awareness/top-mobile-app-security-best-practices-developers>

Dziękujemy za uwagę!
