Technologia i rozwiązania

Bezpieczeństwo urządzeń mobilnych
Receptury

Prashant Verma
Akshay Dixit

Helion

[PACKT] PUBLISHING

Tytuł oryginału: Mobile Device Exploitation Cookbook

Tłumaczenie: Andrzej Watrak

ISBN: 978-83-283-3216-4

First published in the English language under the title ‘Mobile Device Exploitation Cookbook (9781783558728)’

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/bezumo.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/bezumo
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Wprowadzenie do bezpieczeństwa urządzeń przenośnych

W tym rozdziale opiszemy następujące receptury:

- Instalacja i konfiguracja pakietu SDK oraz programu ADB dla systemu Android
- Utworzenie prostej aplikacji dla systemu Android i uruchomienie jej w emulatorze
- Analiza modelu uprawnień w systemie Android za pomocą programu ADB
- Omijanie blokady ekranu w systemie Android
- Przygotowanie środowiska programistycznego Xcode i symulatora w systemie iOS
- Utworzenie prostej aplikacji w systemie iOS i uruchomienie jej w symulatorze
- Przygotowanie środowiska do testów penetracyjnych systemu Android
- Przygotowanie środowiska do testów penetracyjnych systemu iOS
- Uzyskiwanie dostępu administracyjnego do urządzeń przenośnych

Wprowadzenie

Użytkowanie smartfonów jest w dzisiejszym świecie gorącym tematem. Przybywa użytkowników smartfonów, a nie tradycyjnych telefonów komórkowych. Według różnych analiz i badań w przyszłości będziemy coraz więcej korzystać ze smartfonów i tabletów, ponieważ za pomocą tych urządzeń można robić wiele różnych ciekawych rzeczy.

Wraz z rosnącą popularnością urządzeń przenośnych coraz większe jest ryzyko ich użytkowania. Hakerzy i przestępcy informatyczni szukają wszelkich sposobów na zaatakowanie urządzeń użytkowników, pozyskanie ich danych osobistych, numerów kart kredytowych, haseł i innych poufnych informacji. Producenci systemów bezpieczeństwa publikują raporty o rosnącej liczbie ataków towarzyszących coraz większemu wykorzystaniu urządzeń przenośnych. Dziś wiele firm obawia się ujawnienia swoich wewnętrznych danych, a co za tym idzie strat finansowych i utraty reputacji.

W tej książce zapoznamy Cię z kilkoma metodami włamywania się do urządzeń przenośnych, abyś miał wyobrażenie o możliwych atakach. Ich znajomość pozwoli Ci być lepiej przygotowanym do przeciwdziałania im i do chronienia swoich danych.

Ten rozdział opisuje podstawowe modele bezpieczeństwa dwóch najbardziej popularnych systemów operacyjnych stosowanych w urządzeniach przenośnych: Android oraz iOS. Wprowadzimy Cię również w wykorzystywane w tych systemach środowiska programistyczne. Opiszemy środowiska do przeprowadzania testów penetracyjnych oraz uzyskiwania dostępu administracyjnego do systemów operacyjnych. Rozdział ten stanowi podstawę dla tematów opisanych w następnych rozdziałach i zawiera informacje niezbędne do poznania metod wykorzystywania luk w bezpieczeństwie urządzeń.

Instalacja i konfiguracja pakietu SDK oraz programu ADB w systemie Android

Pierwszym krokiem do tworzenia i testowania bezpieczeństwa aplikacji w systemie Android jest instalacja i konfiguracja pakietu SDK oraz programu ADB. Pakiet SDK jest dostępny w dwóch wersjach: jako składnik środowiska Android Studio i jako samodzielny produkt. Poniższy opis dotyczy przygotowania środowiska Android Studio wraz z pakietem SDK.

Program ADB (ang. *Android Debug Bridge*) jest bardzo przydatnym narzędziem umożliwiającym komunikację komputera z urządzeniem Android lub jego symulatorem. Jest wykorzystywany do diagnozowania problemów i testowania bezpieczeństwa aplikacji dla urządzeń przenośnych.

> Używane w książce określenie „urządzenia Android" oznacza smartfony i tablety.

Przygotuj się

Otwórz w przeglądarce stronę *https://developer.android.com* i pobierz oprogramowanie Android Studio lub sam pakiet SDK. Będziesz potrzebował również oprogramowania JDK w wersji 7 lub nowszej.

Jak to zrobić?

Zainstaluj środowisko Android Studio:

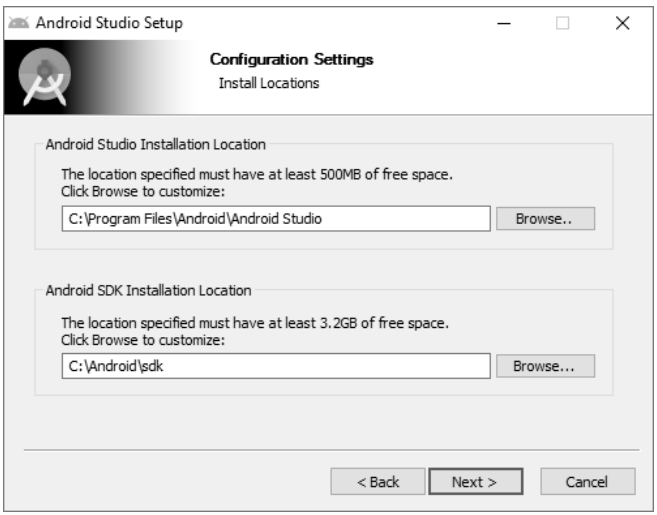
1. Otwórz stronę *http://developer.android.com/studio/index.html* i pobierz najnowszą wersję środowiska.
2. Po pobraniu pliku instalacyjnego uruchom go i postępuj według pojawiających się wskazówek.

> W chwili powstawania tej książki plik instalacyjny miał nazwę *android-studio-bundle-145.3360264-windows.exe*.

Wraz z Android Studio instalowane jest domyślnie oprogramowanie SDK i ADB, o ile nie wykluczysz go z instalacji.

> Skrót *AVD* oznacza wirtualne urządzenie Android (ang. *Android Virtual Device*), czyli emulator prawdziwego urządzenia. Emulator służy do uruchamiania, testowania i diagnozowania aplikacji. Szczególnie przydaje się w sytuacjach, gdy odpowiednie rzeczywiste urządzenia nie są dostępne. Większość testów aplikacji wykonuje się za pomocą emulatora. Również Ty wykorzystasz go w następnej recepturze.

Zwróć uwagę na ścieżki instalacyjne środowiska Android Studio i pakietu SDK. Będziesz je wielokrotnie wykorzystywał podczas konfigurowania środowiska:

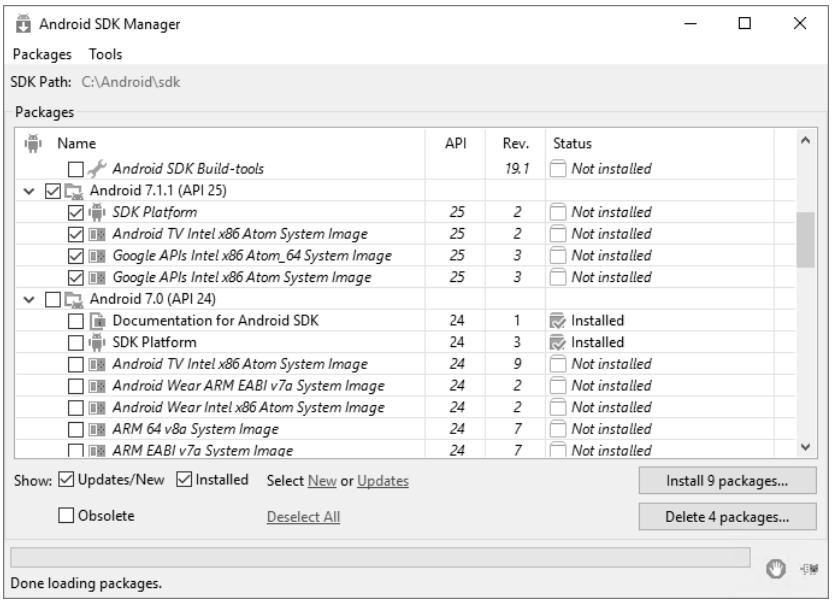


Po zainstalowaniu środowiska Android Studio uruchom je. Pojawi się kilka kolejnych wskazówek. Będziesz musiał pobrać pakiet SDK, co w zależności od szybkości połączenia z internetem może zająć do czterech godzin.

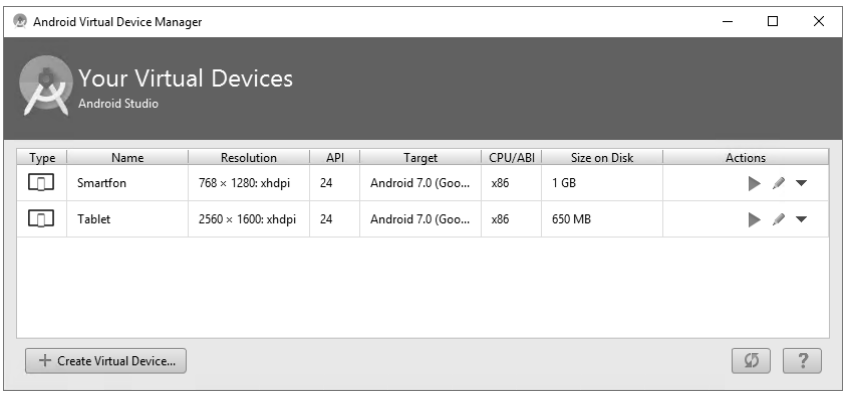
Jak to działa?

Środowisko programistyczne jest gotowe. Poświęć chwilę na zapoznanie się z zawartością folderu z pakietem SDK (którego adres widoczny jest na poprzednim rysunku). Musisz znać kilka plików i folderów:

- ***SDK Manager.exe***: oprogramowanie do zarządzania pakietami SDK, usuwania ich starych wersji oraz instalowania nowych.



- ***AVD Manager.exe***: oprogramowanie do zarządzania urządzeniami AVD. W odpowiednim czasie użyjesz go do utworzenia kilku emulatorów.



Uruchom teraz jeden z emulatorów, aby sprawdzić, czy działa poprawnie. Uruchomienie zajmuje 2 – 3 minuty, bądź więc cierpliwy. Jeżeli instalacja środowiska przebiegnie pomyślnie, emulator uruchomi się bez problemów. Przykładowy wygląd emulatora jest pokazany w opisie następnej receptury.

- ***platform-tools***: folder zawierający kilka przydatnych narzędzi, m.in. program ADB i bazę danych SQLite3. Będziesz z nich korzystał podczas wykonywania receptur opisanych w tej książce.
- ***tools***: folder zawierający pliki wsadowe i aplikacje. Najczęściej będziesz korzystał z pliku *emulator.exe*, czasami również z innych plików z rozszerzeniem *.exe*.

Co dalej?

Ponieważ wielu programistów woli korzystać z innych środowisk niż Android Studio, dlatego możliwe są inne metody tworzenia aplikacji dla tego systemu. W takich sytuacjach należy pobrać osobny pakiet SDK. Zawiera on niezbędne narzędzia programistyczne, które są uruchamiane za pomocą wiersza poleceń.

Narzędzia te przydają się również do wykonywania testów penetracyjnych, przeprowadzania ataków oraz szybkiej analizy kodu i danych aplikacji. Bardzo często kodowanie nie jest wtedy konieczne, za to niezbędna jest diagnostyka kodu. W takich przypadkach można korzystać z samodzielnych narzędzi pakietu SDK.

Zobacz też

- Receptura „Analiza modelu uprawnień w systemie Android za pomocą programu ADB”

Utworzenie prostej aplikacji dla systemu Android i uruchomienie jej w emulatorze

Teraz, gdy pakiet SDK jest już gotowy, napisz swoją pierwszą aplikację dla systemu Android. Aby zacząć, potrzebne będą pewne umiejętności programowania. Niech Cię jednak nie przerazi kod źródłowy. W internecie dostępnych jest mnóstwo przykładów umożliwiających wykonanie pierwszego kroku.

Przygotuj się

Aby móc rozpocząć tworzenie aplikacji dla systemu Android, niezbędny jest poprawnie zainstalowany pakiet SDK. Jeżeli wykonałeś poprzednią recepturę i wiesz co nieco na temat programowania w języku Java, to cała reszta okaże się prosta i będziesz w stanie napisać swoją pierwszą aplikację.

Jak to zrobić?

Napisz prosty program dodający dwie liczby. My nadaliśmy programowi nazwę *Dodawanie*.

1. Utwórz graficzny interfejs aplikacji. W tym celu przeciągnij do okna projektowego trzy pola tekstowe `EditText` (dwa do wprowadzenia liczb i trzecie do wyświetlenia ich sumy), dwie kontrolki `TextView` na teksty informujące użytkownika, że musi wpisać dwie liczby, oraz przycisk do wywoływania metody wykonującej obliczenia.
   - Plik *activity_main.xml* jest tworzony automatycznie. Otwórz go w edytorze i sprawdź, czy wygląda podobnie jak kod przedstawiony poniżej:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.administrator.dodawanie.MainActivity">
```

   - Tekst z opisem pola do wpisania pierwszej liczby:

```
<TextView
    android:text="Pierwsza liczba:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true" />
```

   - Pole tekstowe z identyfikatorem `liczba1`, do którego odwołuje się kod Java:

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:ems="10"
    android:id="@+id/liczba1"
    android:layout_alignBaseline="@+id/textView"
    android:layout_alignBottom="@+id/textView"
    android:layout_toEndOf="@+id/textView"
    android:layout_alignStart="@+id/liczba2" />
```

   - Tekst z opisem pola do wpisania drugiej liczby:

```
<TextView
    android:text="Druga liczba:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:layout_below="@+id/liczba1"
        android:layout_alignParentStart="true"
        android:layout_marginTop="60dp"
        android:id="@+id/textView2" />
```

- Pole tekstowe z identyfikatorem `liczba2`, do którego odwołuje się kod Java:

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="number"
    android:ems="10"
    android:id="@+id/liczba2"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_alignStart="@+id/dodaj" />
```

- Przycisk dodawania:

```
<Button
    android:text="Dodaj"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/liczba2"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="32dp"
    android:id="@+id/dodaj" />
```

- Pole tekstowe z wynikiem dodawania:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/dodaj"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="39dp"
    android:id="@+id/wynik" />
```

```
</RelativeLayout>
```

2. Teraz napisz kod odczytujący i dodający do siebie liczby wprowadzone przez użytkownika oraz wyświetlający ich sumę. Na razie nie przejmuj się terminami takimi jak *Activity*, *Intent* itp. Skup się jedynie na napisaniu poprawnego kodu. Środowisko Android Studio będzie pilnowało każdego Twojego kroku. Zacznij od zakodowania klasy `MainActivity`, jak poniżej:

```
package com.example.administrator.dodawanie;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Button;
```

```
import android.view.View;

public class MainActivity extends AppCompatActivity {
  EditText liczba1;
  EditText liczba2;
  TextView wynik;
  Button dodaj;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    liczba1 = (EditText)findViewById(R.id.liczba1);
    liczba2 = (EditText)findViewById(R.id.liczba2);
    dodaj = (Button)findViewById(R.id.dodaj);
    wynik = (TextView)findViewById(R.id.wynik);

    liczba1 = (EditText)findViewById(R.id.liczba1);
    dodaj.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View v) {
        Suma();
      }
    });
  }

  private void Suma(){
    int l1 = Integer.parseInt(liczba1.getText().toString());
    int l2 = Integer.parseInt(liczba2.getText().toString());
    int l3 = l1 + l2;
    wynik.setText(Integer.toString(l3));
  }
}
```

Zwróć uwagę, jak prosty jest ten kod. Odczytuje jedynie dwie liczby, dodaje je do siebie i wyświetla wynik.

3. Uruchom kod. Pojawi się emulator, a w nim uruchomiony program (patrz rysunek na następnej stronie).

Zobacz też

- Frank Ableson, Robi Sen, Chris King, *Android w akcji. Wydanie II*, Helion 2011.

Analiza modelu uprawnień w systemie Android za pomocą programu ADB

Po przygotowaniu środowiska programistycznego i utworzeniu pierwszej aplikacji czas zapoznać się z modelem uprawnień w systemie Android. System ten wywodzi się z systemu Linux, w którym aplikacje są uruchamiane na określonych kontach użytkowników, należących do określonych grup. W systemie Android stosowany jest ten sam model uprawnień dla aplikacji jak w systemie Linux. Dzięki temu modelowi aplikacje są chronione przed wzajemną ingerencją.

Przygotuj się

Sprawdź, czy zainstalowałeś program ADB. Potrzebny Ci będzie również emulator lub urządzenie z systemem Android, na którym będziesz wykonywał operacje za pomocą programu ADB.

Do opisanych zastosowań najlepiej nadaje się rzeczywiste urządzenie lub emulator używane od jakiegoś czasu (świeżo utworzony emulator lub nowe urządzenie nie zawierają wystarczającej ilości danych, które można by było przeglądać za pomocą programu ADB), dlatego do celów poznawczych lepiej użyć urządzenia z otwartym dostępem administracyjnym.

Jak to zrobić?

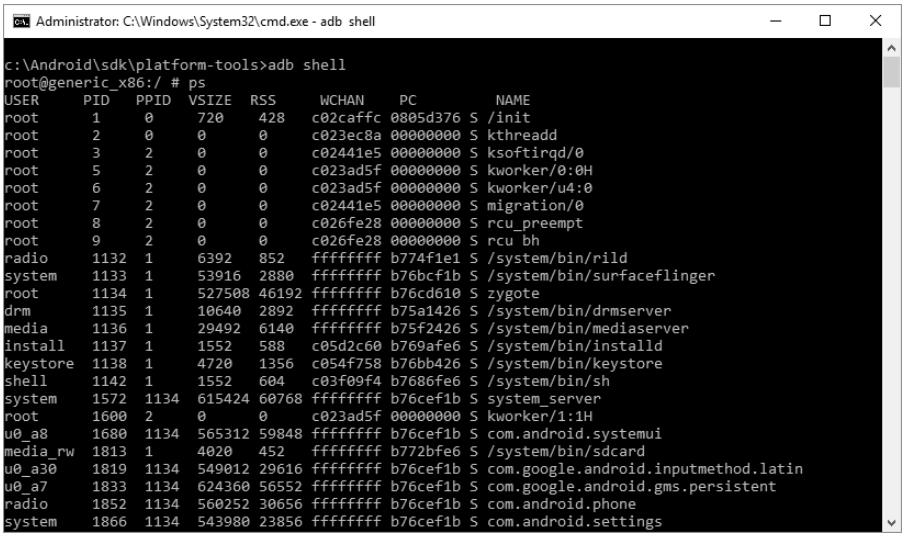
Aby przeanalizować model uprawnień w systemie Android, wykonaj poniższe kroki:

1. Włącz w urządzeniu Android tryb debugowania USB i połącz je z komputerem, na którym uruchomiony jest program ADB. Program ADB jest bardzo przydatnym narzędziem umożliwiającym wykonywanie wielu użytecznych poleceń realizujących m.in. następujące operacje:
   - wysyłanie danych do urządzenia/emulatora,
   - pobieranie danych z urządzenia/emulatora,
   - dostęp do wiersza poleceń urządzenia/emulatora,
   - instalowanie i usuwanie aplikacji,
   - poruszanie się po systemie plików,
   - wykradanie najważniejszych plików systemowych,
   - wykradanie plików aplikacyjnych (na przykład z ustawieniami lub danymi bazy SQLite),
   - przeglądanie dzienników urządzenia.
2. Teraz wykorzystasz program ADB do analizy uprawnień aplikacji. W tym celu musisz najpierw otworzyć wiersz poleceń urządzenia, wpisując polecenie `adb shell`. Następnie wpisz polecenie **`ps`**, tak aby wyświetlić szczegółowe informacje o procesach działających na urządzeniu.

   Rysunek na następnej stronie przedstawia listę procesów działających na telefonie dołączonym do komputera z systemem Windows, na którym został uruchomiony program ADB.

Jak to działa?

Poświęć chwilę na przyjrzenie się powyższemu rysunkowi. Zwróć uwagę na pierwszą, drugą i ostatnią kolumnę z nagłówkami `USER`, `PID` i `NAME`. Każda aplikacja ma unikatowy identyfikator PID i jest uruchamiana na określonym koncie użytkownika. Tylko kilka najważniejszych procesów działa na koncie użytkownika *root*, inne aplikacje są uruchamiane na innych kontach. Na przykład aplikacja `com.android.systemui` o identyfikatorze PID 1680 działa na koncie użytkownika *u0_a8*, natomiast aplikacja `com.android.settings` na koncie użytkownika *system*.



W systemie Android każda aplikacja działa we własnym izolowanym obszarze (ang. *sandbox*). Izolowany obszar jest to wirtualne środowisko, w którym aplikacja działa z ograniczonymi uprawnieniami dostępu do innych aplikacji, jak również inne aplikacje mają do niej ograniczony dostęp. Model uprawnień w systemie Android (czyli zasada uruchamiania aplikacji na określonych kontach) ułatwia tworzenie izolowanych obszarów, dzięki czemu można ograniczać zakres działania (kontekst) aplikacji i udzielać jej pełnego lub ograniczonego dostępu do innych aplikacji. W ten sposób zabezpiecza się również dane aplikacji przed kradzieżą i atakami złośliwego oprogramowania.

Co dalej?

Model uprawnień w systemie Android oraz izolowane obszary są elementami standardowego systemu bezpieczeństwa, będącego celem ataków hakerów. Ataki omijające izolowane obszary lub wykonywane za pomocą niebezpiecznego kodu to przykłady łamania zabezpieczeń systemu Android. Standardowe zabezpieczenia są w systemie zaimplementowane w formie modelu uprawnień.

Zobacz też

- Więcej informacji na temat programu ADB znajdziesz na stronie *http://developer.android.com/tools/help/adb.html*.

Omijanie blokady ekranu w systemie Android

Użytkownikom urządzeń przenośnych zaleca się ich zabezpieczanie za pomocą hasła, kodu pin i blokady ekranu (graficznego wzoru). Termin „omijanie blokady" kojarzy się użytkownikom raczej z odzyskiwaniem zapomnianego wzoru blokującego telefon, a nie z ominięciem zabezpieczeń i dostaniem się do urządzenia. Dla nas to określenie opisuje tę drugą, bardziej agresywną operację, ponieważ książka poświęcona jest lukom w bezpieczeństwie urządzeń. Jak możesz jako haker ominąć blokadę ekranu w urządzeniu ofiary? Ten temat jest szeroko dyskutowany i obecnie znanych jest kilka sztuczek umożliwiających osiągnięcie tego celu. Powstało wiele wirusów, które w niektórych wersjach systemu Android skutecznie realizują to zadanie, a w innych nie działają wcale.

Przygotuj się

Przeanalizujmy przykład omijania blokady ekranu telefonu za pomocą programu ADB. W tym celu będzie Ci potrzebny program ADB, który poznałeś w opisie poprzedniej receptury. Teraz wykorzystaj nabytą wiedzę do włamania się do telefonu. Ponadto będzie Ci potrzebne urządzenie z włączoną opcją debugowania USB i zdefiniowaną blokadą, którą usuniesz.

Jak to zrobić?

Aby ominąć blokadę ekranu, wykonaj poniższe kroki:

1. Podłącz urządzenie do komputera za pomocą przewodu USB. Jeżeli w urządzeniu jest włączona opcja debugowania USB i otwarty jest dostęp administracyjny, wtedy zadanie jest o wiele łatwiejsze. Jeżeli dostęp administracyjny jest zamknięty, i tak będziesz mógł się włamać. Jednak na razie w tej recepturze zajmijmy się pierwszym przypadkiem.
2. Po podłączeniu urządzenia wpisz w terminalu poniższe polecenie:

   ```
   adb shell
   ```

3. Pojawi się wiersz poleceń podłączonego urządzenia Android.
4. Teraz przejdź do katalogu */data/system*, w którym znajdują się pliki kluczy. Wpisz następujące polecenie:

   ```
   cd /data/system
   ```

5. Na koniec musisz usunąć określony klucz wykorzystywany do blokowania ekranu. W tym celu możesz po prostu użyć następującego polecenia:

   ```
   rm *.key
   ```

6. Jeżeli pojawi się komunikat `Permission denied` informujący, że do wykonania powyższego polecenia potrzebne są uprawnienia superużytkownika, wpisz polecenie **su** i powtórz operację. Zostaną w ten sposób usunięte pliki kluczy zawierające informacje wykorzystywane do blokowania ekranu.
7. Uruchom ponownie urządzenie. Blokada ekranu powinna zostać teraz wyłączona.

Jak to działa?

Pliki kluczy zapisane w katalogu */data/system* zawierają różne informacje systemowe, między innymi wzór blokady ekranu. Jeżeli pliki te zostaną usunięte, system po restarcie nie odnajdzie informacji potrzebnych do zablokowania ekranu, więc po prostu otworzy dostęp do urządzenia.

> Wykonanie tej receptury na urządzeniu z włączoną opcją debugowania USB i otwartym dostępem administracyjnym jest bardzo proste.

Co dalej?

Należy przede wszystkim pamiętać, że powyższa receptura nie jest jedynym sposobem ominięcia blokady ekranu, nie zawsze też jest skuteczna. Hakerzy stosują wiele różnych metod omijania blokady ekranu w urządzeniu Android. Zadanie jest tym bardziej skomplikowane, że nie wszystkie metody działają na wszystkich wersjach systemu. Dlatego czasami będziesz musiał poświęcić więcej czasu na znalezienie sposobu ominięcia blokady ekranu.

Przygotowanie środowiska programistycznego Xcode i symulatora w systemie iOS

Wiesz już, jak wygląda środowisko programistyczne dla systemu Android, czas więc poznać środowisko dla systemu iOS. Ten system jest stosowany w telefonach iPhone i tabletach iPad. Aplikacje dla tego systemu tworzy się w środowisku Xcode, które działa tylko w systemie iOS. Środowisko to (wraz z symulatorem urządzeń) jest wykorzystywane do tworzenia aplikacji dla systemu iOS.

> Zwróć uwagę, że piszemy o *emulatorze* urządzenia Android i *symulatorze* urządzenia iOS. Terminy te są do siebie podobne, jednak jest między nimi jedna istotna różnica. W emulatorze można wykorzystywać pewne funkcjonalności systemu operacyjnego do testowania określonych aplikacji.

Na przykład emulator może korzystać z kamery laptopa, jeżeli wymaga tego jej aplikacja. W przypadku symulatora urządzenia iOS testy takiej aplikacji będą ograniczone. Ponadto emulator może wysyłać wiadomości SMS do innego emulatora. Niektórzy programiści uważają, że emulator jest lepszy od symulatora, jednak takie uogólnienia nie zawsze są słuszne, ponieważ oba rodzaje programów wykonują te zadania, do których są przeznaczone.

Przygotuj się

Xcode jest środowiskiem przeznaczonym do tworzenia aplikacji dla systemu iOS. Środowisko to działa tylko w tym systemie, więc będzie Ci potrzebny komputer MacBook. Zainstaluj na nim środowisko Xcode oraz pakiet SDK i zacznij kodować.

Informacje o tym, jak zacząć pracę ze środowiskiem Xcode, znajdziesz na stronie *https://developer.apple.com/programs/how-it-works*.

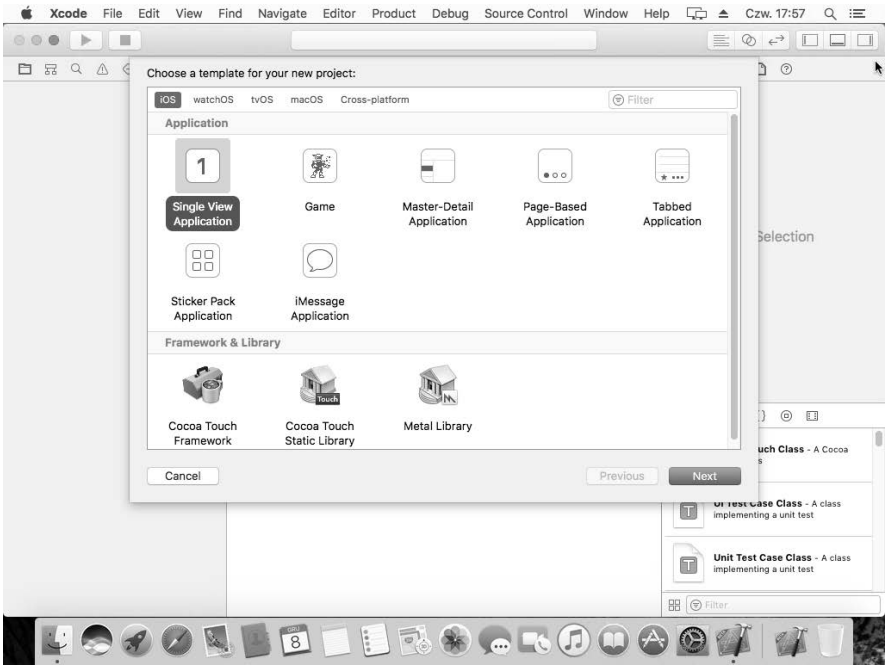
Jak to zrobić?

Aby skonfigurować środowisko Xcode i symulator urządzenia iOS, wykonaj poniższe kroki:

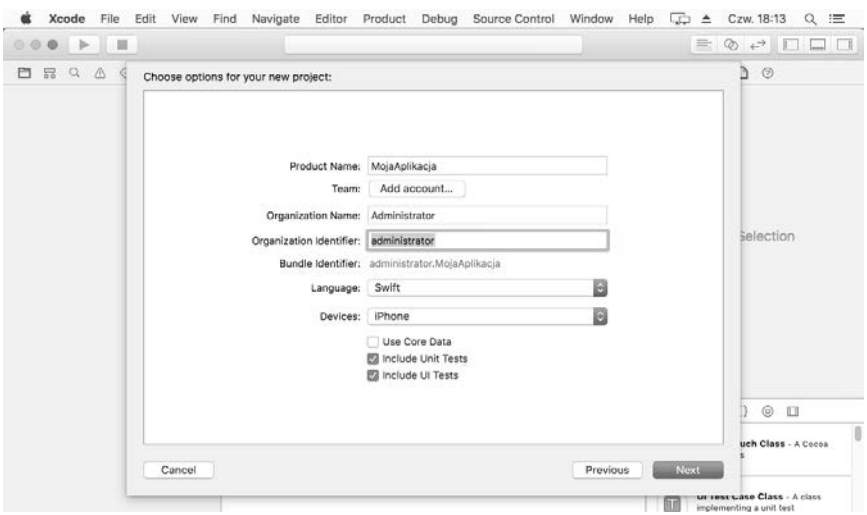
1. Otwórz aplikację App Store i pobierz oprogramowanie Xcode (w taki sam sposób, jak pobiera się każdą inną aplikację). Będzie Ci do tego potrzebny identyfikator Apple ID. Środowisko Xcode można pobrać bezpłatnie ze sklepu App Store.
2. Po zainstalowaniu środowiska możesz zacząć z niego korzystać. Środowisko to jest powszechnie stosowane do tworzenia aplikacji dla różnych systemów operacyjnych firmy Apple. Aby tworzyć aplikacje, będziesz potrzebował również pakietu SDK. Najnowsza wersja środowiska Xcode zawiera już ten pakiet oraz symulator i dodatkowe narzędzia (ang. *Instruments*). Na szczęście instalacja wszystkich powyższych komponentów nie jest skomplikowana, ponieważ wszystko jest zawarte w pliku instalacyjnym środowiska Xcode.
3. Po zakończonej instalacji będziesz mógł utworzyć nowy projekt. Po uruchomieniu środowiska powinno pojawić się okno, tak jak na poniższym rysunku, umożliwiające wybranie systemu operacyjnego, dla którego będziesz tworzył aplikację (patrz rysunek na następnej stronie).

Jak to działa?

Zapoznaj się teraz ze środowiskiem Xcode.



W oknie pokazanym na powyższym rysunku utwórz nowy projekt. Na początek wybierz prosty szablon *Single View Application* (aplikacja z jednym widokiem) i kliknij przycisk *Next* (dalej). Otworzy się okno *Choose options for your new project* (wybierz opcje nowego projektu). W polu *Product Name* (nazwa produktu) wpisz nazwę projektu. Nazwa ta zostanie dołączona do nazwy firmy (pole *Organization Identifier*) i w ten sposób zostanie utworzona nazwa pakietu aplikacji.

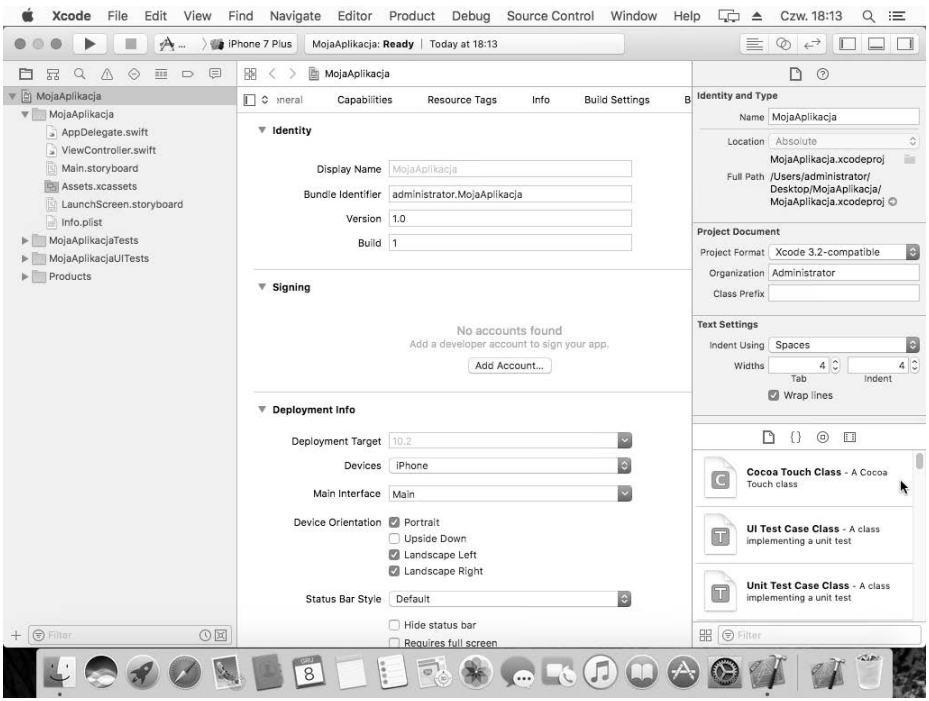


Zwróć uwagę, że w powyższym przypadku w polu *Language* (język) wybraliśmy opcję *Swift*. Jest to nowy język programowania, wprowadzony w systemie iOS 8. Można również wybrać opcję *Objective-C* oznaczającą tradycyjny język C.

> Swift jest nowym językiem programowania aplikacji dla systemu iOS. Jest to interaktywny język, w którym kodowanie jest przyjemnością. Tworzenie aplikacji jest dzięki niemu łatwiejsze, a ponadto można stosować w nim składnię tradycyjnego języka C.

Ważne jest również wybranie z listy *Devices* odpowiedniego urządzenia: *Universal*, *iPhone* lub *iPad*. W tym przykładzie wybierz opcję *iPhone*.

Po wybraniu opcji kliknij przycisk *Next*, a następnie *Create* (utwórz). Pojawi się okno projektu.

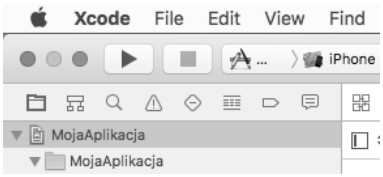


Po lewej stronie okna znajduje się panel nawigacyjny zawierający wszystkie pliki wchodzące w skład projektu. W środkowej części widoczny jest edytor kodu. Jego wygląd może być różny w zależności od rodzaju edytowanego pliku.

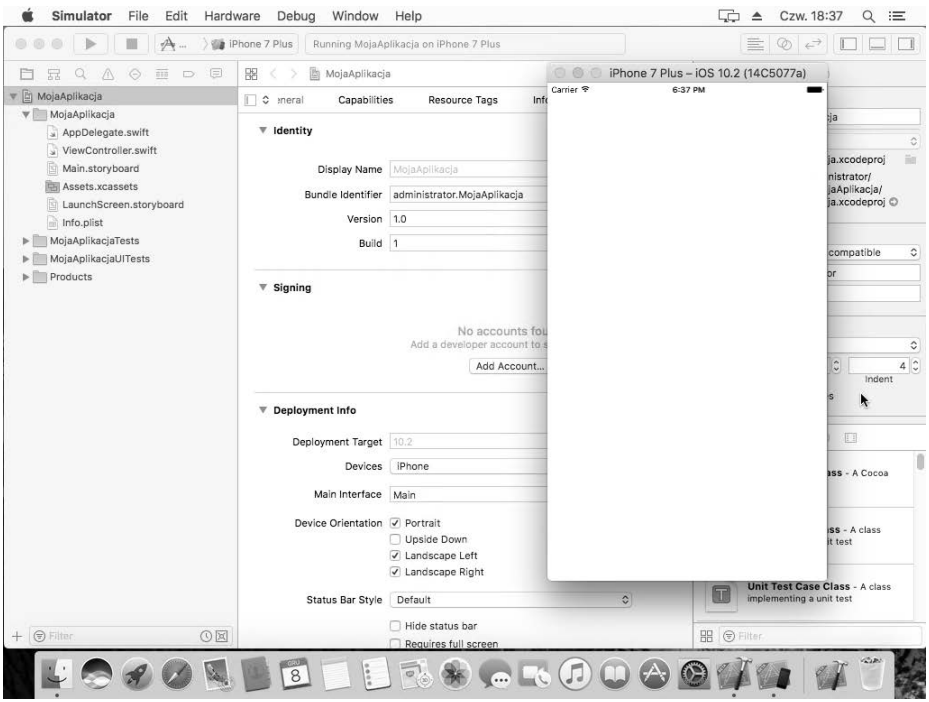
Po prawej stronie znajduje się panel pomocniczy, zawierający właściwości projektu oraz umożliwiający otwarcie systemu pomocy.

Co dalej?

Chociaż nie napisałeś jeszcze ani jednego wiersza kodu, możesz już uruchomić symulator. Służy do tego celu przycisk znajdujący się po lewej stronie paska narzędzi i przypominający przycisk do odtwarzania muzyki:



Gdy klikniesz ten przycisk, środowisko Xcode automatycznie skompiluje projekt aplikacji i uruchomi ją w domyślnym symulatorze telefonu iPhone 7. Ponieważ nie wpisałeś jeszcze żadnego kodu, w symulatorze zobaczysz jedynie biały, pusty ekran:



Do zatrzymywania aplikacji służy przycisk znajdujący się obok przycisku uruchamiającego.

Zobacz też

- Receptura „Przygotowanie środowiska do testów penetracyjnych systemu iOS”

Utworzenie prostej aplikacji w systemie iOS i uruchomienie jej w symulatorze

Po zapoznaniu się ze środowiskiem Xcode oraz symulatorem możesz zacząć kodować swoją pierwszą aplikację.

Przygotuj się

Do utworzenia aplikacji będzie Ci potrzebny MacBook z zainstalowanym i uruchomionym środowiskiem Xcode i symulatorem. Jeżeli wykonałeś poprzednią recepturę i znasz trochę język Swift, jesteś w pełni gotów do napisania swojej pierwszej aplikacji dla systemu iOS.

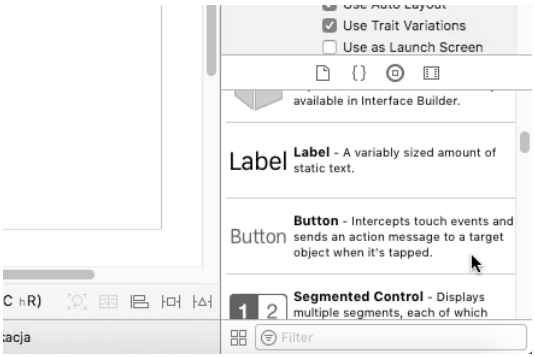
Jak to zrobić?

Po ogólnym zapoznaniu się ze środowiskiem Xcode zacznij tworzyć interfejs graficzny aplikacji:

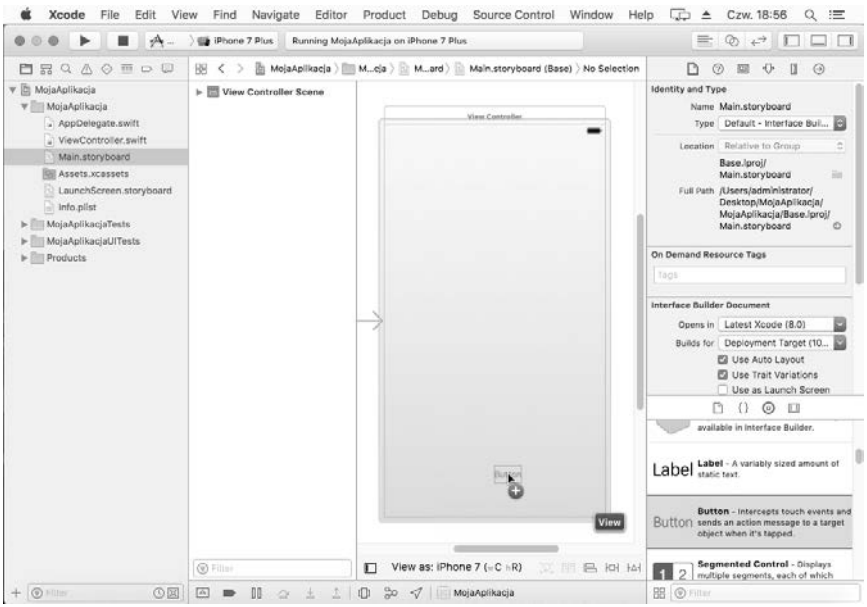
1. W panelu nawigacyjnym kliknij pozycję *Main.storyboard*. Pojawi się edytor interfejsu aplikacji:

Edytor interfejsu służy do tworzenia widoków aplikacji i definiowania przejść pomiędzy nimi. Ponieważ Twoja aplikacja składa się tylko z jednego widoku, w interfejsie został umieszczony już jego kontroler (*View Controller*).

2. Teraz dodaj przycisk do widoku. W dolnej części panelu po prawej stronie okna znajduje się biblioteka obiektów (*Object Library*), pokazana na poniższym rysunku:

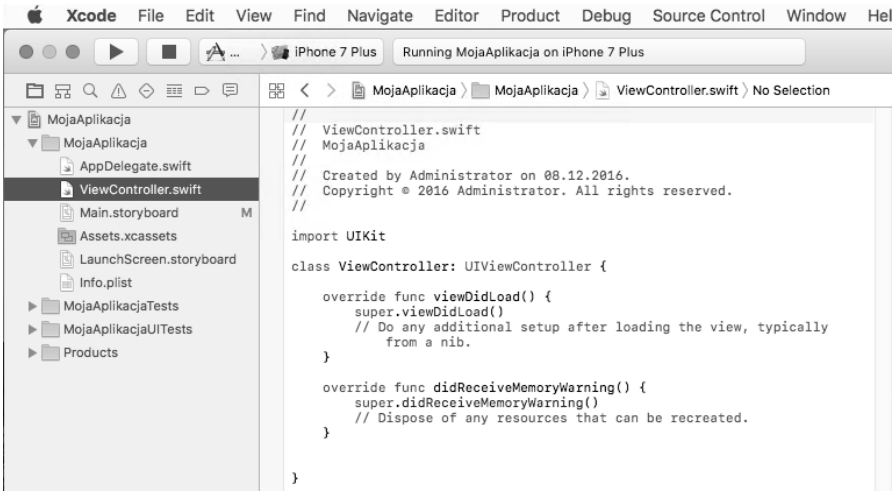


3. Przeciągnij z biblioteki do widoku element *Button* (przycisk):



4. Po przeciągnięciu przycisku umieść go w dowolnym miejscu widoku, a następnie dwukrotnie kliknij i zmień jego nazwę na *Kliknij mnie*.

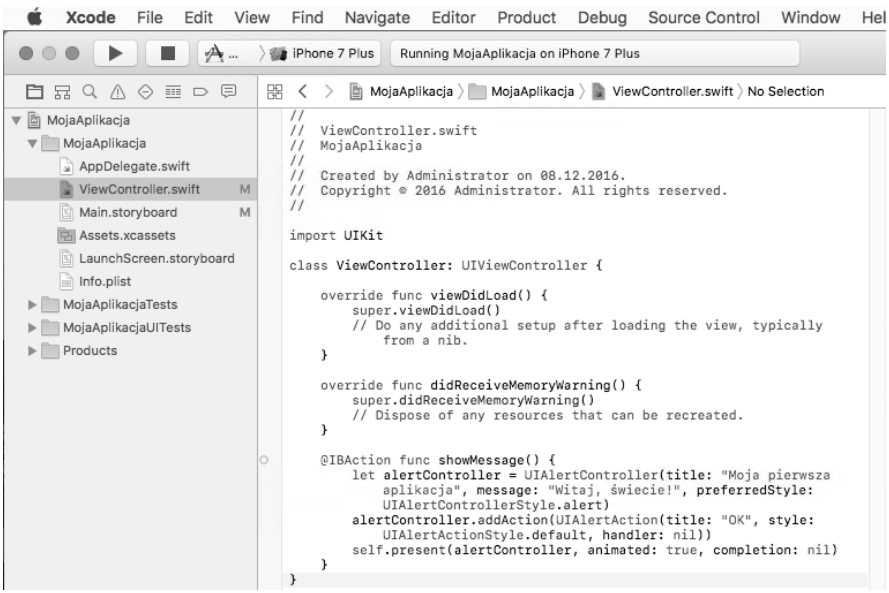
5. Teraz wpisz kilka wierszy kodu wyświetlającego komunikat. W panelu nawigacyjnym znajduje się plik o nazwie *ViewController.swift*. Kliknij go i do zdefiniowanej w nim klasy `ViewController` dodaj nową metodę, która będzie wyświetlała komunikat.



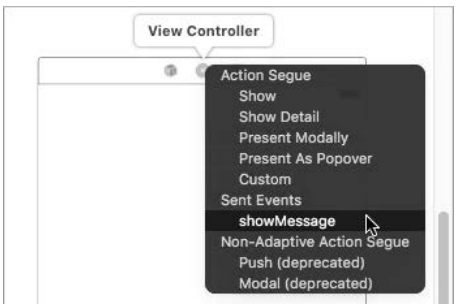
6. Teraz wpisz poniższy kod metody:

```
@IBAction func showMessage(){
  let alertController = UIAlertController(title: "Moja pierwsza aplikacja", message:
    "Witaj, świecie!", preferredStyle: UIAlertControllerStyle.Alert)
  alertController.addAction(UIAlertAction(title: "OK",
    style: UIAlertActionStyle.Default, handler:nil))
  self.present (alertController, animated: true, completion: nil)
}
```

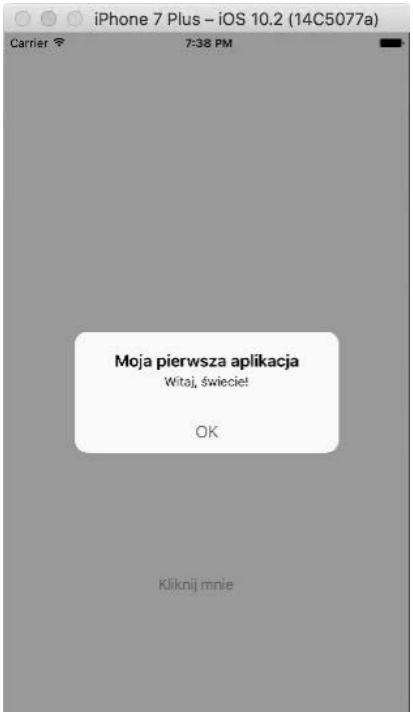
7. Gotowy kod powinien wyglądać następująco:



8. Teraz musisz połączyć przycisk *Kliknij mnie* w interfejsie z metodą `showMessage`. Jest to prosta operacja. Najpierw w panelu nawigacyjnym kliknij pozycję *Main.storyboard*, która zawiera definicję zawartości ekranu.
9. Naciśnij klawisz *Ctrl*, a następnie kliknij przycisk *Kliknij mnie* i przeciągnij go do ikony *View Controller*.
10. Zwolnij oba klawisze. Pojawi się menu z opcją `showMessage`. Kliknij ją, aby połączyć przycisk z metodą:



11. Gotowe! Jeżeli wszystkie operacje wykonałeś poprawnie, możesz uruchomić aplikację. Gdy klikniesz przycisk *Kliknij mnie*, powinien pojawić się następujący widok:



Jak to działa?

Atrybut `@IBAction`, wprowadzony w języku Swift, służy do łączenia elementów interfejsu użytkownika z kodem aplikacji. W tym przypadku został on użyty do połączenia przycisku z metodą wyświetlającą komunikat. Aby można było wykonać takie połączenie, określiłeś typ `func` metody `showMessage`.

Gdy przycisk *Kliknij mnie* przeciągnąłeś do ikony *View Controller* i wybrałeś w menu opcję `showMessage`, utworzyłeś połączenie pomiędzy przyciskiem a metodą.

> Począwszy od wersji systemu iOS 8 klasy `UIActionSheet` i `UIAlertView` zostały zastąpione nową klasą `UIAlertController`.

W kodzie metody wywoływana jest funkcja `UIAlertController` wyświetlająca komunikat z tytułem *Moja pierwsza aplikacja* oraz treścią *Witaj, świecie!* Zdefiniowana jest również następująca akcja:

```
alertController.addAction(UIAlertAction(title: "OK",
  style: UIAlertActionStyle.Default, handler:nil))
```

Jest to kod, który po kliknięciu przycisku *OK* zamyka okno komunikatu.

Co dalej?

Możesz poeksperymentować z różnymi stylami przycisku, umieścić w interfejsie tabele, odnośniki itp. Zastosuj inne funkcjonalności, aby nauczyć się kodować aplikacje dla systemu iOS.

Dobrym punktem wyjścia będzie zapoznanie się z dokumentacją napisaną przez twórców systemu iOS i dostępną pod adresem *https://developer.apple.com/library/ios/documentation/Swift/Conceptual/BuildingCocoaApps/index.html*.

Zobacz też

- Mnóstwo materiału ułatwiającego naukę kodowania aplikacji dla systemu iOS, w tym filmów, podręczników i przykładowych kodów, znajdziesz na stronie *https://developer.apple.com/swift/resources*.

Przygotowanie środowiska do testów penetracyjnych systemu Android

Poznałeś już środowisko programistyczne Android Studio, program ADB i emulator. Utworzyłeś również swoją pierwszą aplikację. Teraz zajmijmy się testami penetracyjnymi. Testy penetracyjne urządzeń przenośnych można ogólnie podzielić na cztery kategorie:

- ataki na dane przesyłane przez aplikacje,
- ataki na pamięć urządzenia,
- ataki na kod źródłowy aplikacji,
- ataki na funkcjonalności systemu operacyjnego urządzenia.

Ostatnia kategoria obejmuje ataki najbardziej skomplikowane. Aplikacje wykorzystują bardzie wiele funkcjonalności, np. komunikację Bluetooth, NFC lub radio, które należy sprawdzić za pomocą wszechstronnych testów penetracyjnych.

Przygotuj się

Musisz przygotować odpowiednio wyposażone środowisko, które umożliwi przeprowadzanie testów penetracyjnych systemu Android z czterech wymienionych wyżej kategorii.

Będą Ci potrzebne następujące komponenty:

- pakiet SDK, emulator, program ADB;
- emulator ze skonfigurowanymi różnymi wersjami systemu Android;
- jeden lub dwa telefony lub tablety (z otwartym dostępem administracyjnym);
- oprogramowanie pośredniczące (proxy), np. Charles, Burp Suite lub Fiddler;
- sieć Wi-Fi;
- przeglądarka bazy SQLite, edytor tekstu, przeglądarka kodu XML;
- przewód USB;
- konwerter plików DEX na JAR, dekompilator kodu Java;
- oprogramowanie DroidProxy lub Autoproxy.

Jak to zrobić?

Przyjrzyjmy się teraz poszczególnym narzędziom:

- Pakiet SDK, emulator, program ADB.

  Narzędzia te poznałeś w recepturach opisanych wcześniej w tym rozdziale.

- Emulator ze skonfigurowanymi różnymi wersjami systemu Android.

  Przyjrzyj się rysunkowi z pierwszej receptury, przedstawiającemu program AVD Manager. Został on użyty do utworzenia wirtualnych urządzeń z wersją systemu Android 7.0 i interfejsem API 24. Klikając przycisk *Create Virtual Device* (utwórz urządzenie wirtualne), możesz utworzyć więcej urządzeń z różnymi wersjami systemu oraz interfejsami API.

  Różne wersje urządzeń wirtualnych przydają się w sytuacjach, gdy trzeba przeprowadzić testy penetracyjne aplikacji utworzonej dla określonego systemu operacyjnego i wykorzystującej jego specyficzne funkcjonalności.

- Jeden lub dwa telefony lub tablety (z otwartym dostępem administracyjnym).

  Posiadanie rzeczywistych urządzeń nie jest konieczne, ale bardzo wskazane. Czasami zdarza się, że aplikacje uruchomione w emulatorze ulegają awarii lub działają bardzo wolno. Ponadto oprogramowanie pośredniczące uruchomione razem z emulatorem również może działać powoli lub ulegać awarii, przez co testy są utrudnione. W takich przypadkach przydają się rzeczywiste urządzenia.

- Oprogramowanie pośredniczące (proxy), np. Charles, Burp Suite lub Fiddler.

  Oprogramowanie pośredniczące można pobrać z odpowiednich stron internetowych. Programy te są dość proste w obsłudze, a ponadto istnieją poświęcone im podręczniki oraz fora użytkowników. Instalacja tego typu narzędzi wykracza poza zakres tej książki. Opiszemy jednak, jak przygotować je do współpracy z aplikacjami dla urządzeń przenośnych.

  Poniżej wymienione są strony z najczęściej stosowanymi narzędziami:

  - *http://portswigger.net/burp/download.html*
  - *http://www.charlesproxy.com/download*
  - *http://www.telerik.com/download/fiddler*

- Sieć Wi-Fi.

  Sieć Wi-Fi będzie potrzebna do ingerowania w przesyłane dane. W dalszej części książki skonfigurujesz oprogramowanie pośredniczące, zainstalowane na komputerze podłączonym do tej samej sieci Wi-Fi co urządzenie Android.

  Możesz wykorzystać istniejącą sieć i router albo utworzyć nową za pomocą laptopa i jednego z bezpłatnych narzędzi. Z naszego doświadczenia wynika, że w drugim przypadku czasami pojawiają się problemy, dlatego preferujemy pierwszy sposób.

- Przeglądarka bazy SQLite, edytor tekstu, przeglądarka kodu XML.

  Powyższe narzędzia będą potrzebne do odczytywania danych pozyskanych z urządzenia. Być może już je posiadasz, a jeżeli nie, możesz je bezpłatnie pobrać z internetu.

- Przewód USB.

  Bardzo ważny jest przewód, którym później połączysz urządzenie z komputerem w celu odczytania danych i przeprowadzenia kontrolowanych ataków.

- Konwerter plików DEX na JAR, dekompilator kodu Java.

  Powyższe niewielkie narzędzia będą bardzo potrzebne w Twoim laboratorium. Za ich pomocą będziesz dekompilował kod aplikacji.

- Oprogramowanie DroidProxy lub Autoproxy.

  Ponieważ starsze wersje systemu Android nie pozwalają na współpracę z oprogramowaniem pośredniczącym, dlatego musisz pobrać ze sklepu Play powyższe narzędzia.

Jak to działa?

Po wyposażeniu laboratorium w wymienione wyżej komponenty sprawdźmy, jak można je wykorzystać do wykonywania testów penetracyjnych z poszczególnych kategorii.

- Ataki na dane przesyłane przez aplikacje

  W tych testach przyda się sieć Wi-Fi i oprogramowanie pośredniczące. Laptopa z zainstalowanym oprogramowaniem Charles lub Burp należy podłączyć do sieci Wi-Fi, a urządzenie przenośne skonfigurować tak, aby przesyłało dane do tego oprogramowania. W razie potrzeby można do tego celu wykorzystać specjalne narzędzia, np. DroidProxy lub Autoproxy. Ponieważ laptop i urządzenie przenośne będą podłączone do tej samej sieci, więc wszystkie dane będą przepływały przez oprogramowanie pośredniczące.

  Za pomocą powyższych narzędzi i oprogramowania pośredniczącego będziesz mógł przechwytywać i modyfikować wszystkie dane przesyłane przez aplikację, o czym będzie mowa w rozdziale 4.

- Ataki na pamięć urządzenia

  Masz przewód USB do połączenia urządzenia z komputerem, masz również emulator. Zarówno na urządzeniu, jak i na emulatorze możesz uruchamiać aplikacje. Masz również potężne narzędzie ADB, za pomocą którego możesz zalogować się do urządzenia lub emulatora, wykraść z niego dane, a także przeprowadzić wiele różnych ataków.

- Ataki na kod źródłowy aplikacji

  Dekompilację kodu można podzielić na dwa etapy: konwersję pliku APK do DEX oraz konwersję DEX na JAR.

  APK jest formatem pakietów aplikacyjnych. Po utworzeniu i skompilowaniu aplikacji tworzony jest plik z rozszerzeniem *.apk*.

  Konwersja pliku APK na DEX jest dość prosta, polega jedynie na zmianie jego nazwy i rozpakowaniu archiwum. Natomiast konwersję pliku DEX na JAR wykonuje się za pomocą odpowiednich narzędzi.

Przygotowanie środowiska do testów penetracyjnych systemu iOS

Poznałeś już środowisko programistyczne Xcode i symulator urządzenia z systemem iOS. Utworzyłeś również swoją pierwszą aplikację. Teraz zajmijmy się testami penetracyjnymi. Podobnie jak w poprzedniej recepturze, testy penetracyjne urządzeń przenośnych można ogólnie podzielić na cztery kategorie:

- ataki na dane przesyłane przez aplikacje przenośne,
- ataki na pamięć urządzenia,
- ataki na kod źródłowy aplikacji,
- ataki na funkcjonalności systemu operacyjnego urządzenia.

Przygotuj się

Musisz przygotować odpowiednio wyposażone środowisko, które umożliwi przeprowadzanie testów penetracyjnych systemu iOS z czterech wymienionych wyżej kategorii.

Będą Ci potrzebne przynajmniej podane niżej komponenty. Ich lista nie różni się zbytnio od wyposażenia laboratorium do testów systemu Android, zawiera jednak kilka specjalistycznych narzędzi.

- symulator;
- środowisko Xcode;
- program iExplorer;
- jeden lub dwa telefony iPhone lub tablety iPad (z otwartym dostępem administracyjnym);
- oprogramowanie pośredniczące (proxy), np. Charles, Burp Suite lub Fiddler;
- sieć Wi-Fi;
- przeglądarka bazy SQLite, edytor tekstu, przeglądarka kodu XML, edytor plików *.plist*;
- przewód USB;
- narzędzia otool i classdump.

Jak to zrobić?

Przyjrzyjmy się teraz poszczególnym narzędziom:

- Symulator

  Symulatora będziesz używał do uruchamiania aplikacji. Testy będziesz mógł przeprowadzić, używając jedynie MacBooka z zainstalowanymi odpowiednimi narzędziami (nie będzie potrzebna sieć Wi-Fi ani rzeczywiste urządzenie przenośne).

- Środowisko Xcode

  Środowisko Xcode służy do tworzenia aplikacji dla systemu iOS. Bardzo przydaje się do przeglądania nie tylko kodu źródłowego, ale również innych plików.

- Program iExplorer

  Program iExplorer jest odpowiednikiem eksploratora plików w systemie Windows i służy do poruszania się w systemie plików urządzeń iPhone i iPad po podłączeniu ich do komputera za pomocą przewodu USB. Za jego pomocą można również odczytywać pliki i wykradać dane. Program można pobrać ze sklepu App Store. Jest również dostępna jego wersja dla systemu Windows.

- Jeden lub dwa telefony iPhone lub tablety iPad (z otwartym dostępem administracyjnym)

  Do testów przydadzą się urządzenia z otwartym dostępem administracyjnym. Będziesz mógł na nich instalować aplikacje do przetestowania, dzięki czemu nie będziesz musiał korzystać z symulatora.

- Oprogramowanie pośredniczące (proxy), np. Charles, Burp Suite lub Fiddler

  Oprogramowanie pośredniczące można pobrać z odpowiednich stron internetowych. Programy te są dość proste w obsłudze, a ponadto istnieją poświęcone im podręczniki i fora użytkowników. Instalacja tego typu narzędzi wykracza poza zakres tej książki. Opiszemy jednak, jak przygotować je do współpracy z aplikacjami dla urządzeń przenośnych.

  Poniżej wymienione są strony z najczęściej stosowanymi narzędziami:

  - *http://portswigger.net/burp/download.html*
  - *http://www.charlesproxy.com/download*
  - *http://www.telerik.com/download/fiddler*

- Sieć Wi-Fi

  Sieć Wi-Fi będzie potrzebna do ingerowania w przesyłane dane. W dalszej części książki skonfigurujesz oprogramowanie pośredniczące, zainstalowane na komputerze podłączonym do tej samej sieci Wi-Fi co urządzenie przenośne.

  Możesz wykorzystać istniejącą sieć i router albo utworzyć nową za pomocą laptopa i jednego z bezpłatnych narzędzi. Z naszego doświadczenia wynika, że w drugim przypadku czasami pojawiają się problemy, dlatego preferujemy pierwszy sposób.

- Przeglądarka bazy SQLite, edytor tekstu, przeglądarka kodu XML, edytor plików *.plist*

  Powyższe narzędzia będą potrzebne do odczytywania danych pozyskanych z urządzenia. Być może już je posiadasz, a jeżeli nie, możesz je bezpłatnie pobrać z internetu.

  W plikach *.plist* aplikacje zapisują swoje dane, dlatego do ich odczytywania będzie potrzebny odpowiedni edytor.

- Przewód USB

  Bardzo ważny jest przewód, którym później połączysz urządzenie z komputerem w celu odczytania danych i przeprowadzenia kontrolowanych ataków.

- Narzędzia otool i classdump-z

  Powyższe narzędzia służą do dekompilowania kodu aplikacji dla systemu iOS.

Jak to działa?

Po wyposażeniu laboratorium w wymienione wyżej komponenty sprawdźmy, jak można je wykorzystać do wykonywania testów penetracyjnych z poszczególnych kategorii.

- Ataki na dane przesyłane przez aplikacje

  W tych testach przyda się sieć Wi-Fi oraz oprogramowanie pośredniczące. Laptop z zainstalowanym oprogramowaniem Charles lub Burp należy podłączyć do sieci Wi-Fi, a urządzenie przenośne skonfigurować tak, aby przesyłało dane do tego oprogramowania. Ponieważ laptop i urządzenie będą podłączone do tej samej sieci, więc wszystkie dane będą przepływały przez oprogramowanie pośredniczące. Do testów nie będzie potrzebny MacBook (można użyć dowolnego komputera), natomiast będzie potrzebne urządzenie z systemem iOS.

  Możesz również użyć samego MacBooka, bez urządzenia przenośnego. Będzie wtedy potrzebne środowisko Xcode i symulator. Na laptopie trzeba również zainstalować oprogramowanie pośredniczące Burp lub Charles.

  Za pomocą powyższych narzędzi i oprogramowania pośredniczącego będziesz mógł przechwytywać i modyfikować wszystkie dane przesyłane przez aplikację, o czym będzie mowa w rozdziale 4.

- Ataki na pamięć urządzenia

  Masz przewód USB do połączenia urządzenia z komputerem, masz również program iExplorer, za pomocą którego będziesz mógł odczytywać pliki i wykradać dane.

- Ataki na kod źródłowy aplikacji

  W tej książce opisane są narzędzia otool i classdump-z. Dekompilację kodu aplikacji dla systemu iOS można przeprowadzić w ograniczonym zakresie, więc powyższe narzędzia przydadzą się tylko częściowo. Szczegółowo opiszemy je w dalszej części książki.

Co dalej?

- *Ataki na funkcjonalności systemu operacyjnego urządzenia*

Powyższa kategoria ataków obejmuje szczególne przypadki, które w systemie iOS są wyjątkowo skomplikowane. Aplikacje wykorzystują bardzo dużo funkcjonalności, np. komunikację Bluetooth, NFC lub radio, które należy sprawdzić za pomocą wszechstronnych testów penetracyjnych. Typowym przykładem jest wykonywanie zrzutów zawartości ekranu w tle aplikacji.

Uzyskiwanie dostępu administracyjnego do urządzenia przenośnego

Rooting polega na uzyskaniu do urządzenia dostępu administracyjnego, dającego możliwość wykonywania wszelkich operacji w systemie Android, np. montowania/demontowania systemów plików, uruchamiania usług SSH, HTTP, DHCP, DNS lub proxy, przerywania działania procesów itp.

Dzięki możliwości uruchamiania poleceń z uprawnieniami administratora można w systemie Android wykonywać dowolne operacje.

Jailbreaking (dosł. ucieczka z więzienia) to proces rozszerzania uprawnień i usuwania ograniczeń dostępu do podzespołów urządzenia z systemem iOS. Uzyskuje się wtedy dostęp administracyjny do plików systemowych, można pobierać aplikacje, rozszerzenia i motywy, które standardowo nie są dostępne w sklepie Apple Store.

Przygotuj się

Do uzyskania dostępu administracyjnego do urządzenia Android potrzebny będzie jedynie przewód USB oraz odpowiedni kod, który trzeba uruchomić za pomocą programu ADB lub zapisać w pamięci ROM urządzenia.

W przypadku urządzenia z systemem iOS potrzebny będzie przewód USB i odpowiednie oprogramowanie.

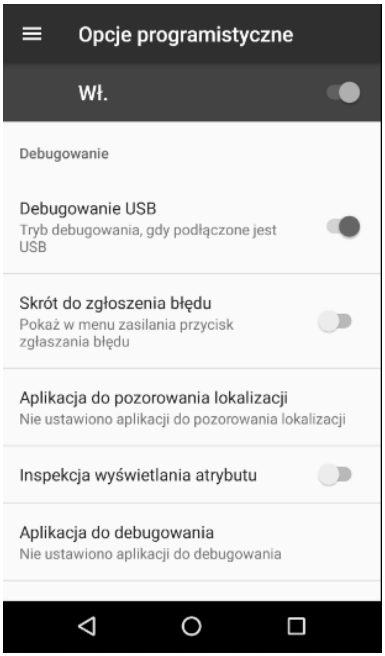
Jak to zrobić?

Poniżej opisane są operacje uzyskiwania dostępu administracyjnego do obu typów urządzeń.

Dostęp do urządzenia Android

Właściwy dostęp uzyskuje się jednym kliknięciem. Wcześniej jednak trzeba wykonać kilka prostych operacji:

1. Pobrać i zainstalować na komputerze oprogramowanie Java JDK, a następnie Android SDK.
2. Włączyć na urządzeniu opcję debugowania USB. W tym celu kliknij ikonę *Ustawienia*, następnie *Opcje programistyczne* i *Debugowanie USB*:



Od tej chwili dalsza część procesu uzyskiwania dostępu administracyjnego polega na znalezieniu w internecie i wykonaniu sprawdzonej, odpowiedniej dla danego urządzenia metody. Dla większości urządzeń jest to jedna z następujących możliwości:

- Użycie odpowiedniej aplikacji na komputerze

  Wykonaj następujące operacje:

  1. Zainstaluj na komputerze aplikację do uzyskiwania dostępu administracyjnego.
  2. Podłącz do komputera urządzenie za pomocą przewodu USB.
  3. Wykonaj instrukcje podawane przez aplikację.

- Użycie aplikacji na urządzeniu

  Wykonaj następujące operacje:

  1. Pobierz plik APK do uzyskiwania dostępu administracyjnego.
  2. Włącz w urządzeniu opcję debugowania USB i w ustawieniach bezpieczeństwa zezwól na instalację aplikacji z nieznanego źródła.
  3. Zainstaluj plik APK za pomocą polecenia `adb install /ścieżka/do/pliku/apk`.
  4. Postępuj według wskazówek pojawiających się na ekranie.

- Zapisanie kodu w pamięci ROM

  Wykonaj następujące operacje:

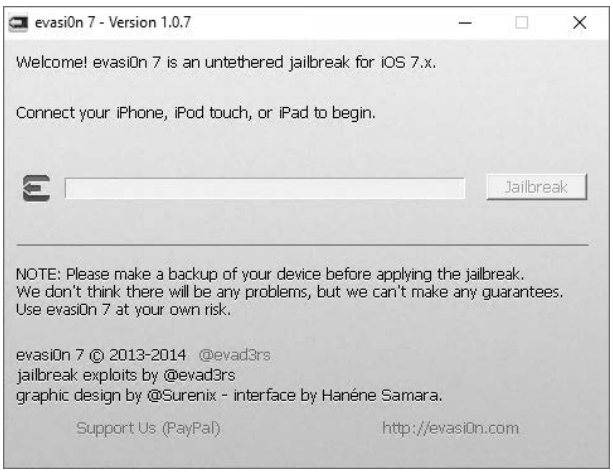
  1. Zapisz na karcie SD zmodyfikowaną zawartość pamięci ROM (w postaci pliku *.zip*).
  2. Uruchom ponownie urządzenie w trybie odzyskiwania.

3. W menu odzyskiwania wybierz opcję instalacji pliku zapisanego na karcie SD.
4. Wskaż plik *.zip* do zapisania w pamięci ROM.

Dostęp do urządzenia iOS

Zanim otworzysz dostęp administracyjny, musisz utworzyć kopię zapasową urządzenia. Dzięki temu będziesz mógł odtworzyć system, jeżeli z jakiegoś powodu operacja się nie powiedzie.

Proces uzyskiwania dostępu administracyjnego polega na pobraniu na komputer odpowiedniego programu dla systemu iOS lub Windows, podłączeniu urządzenia do komputera za pomocą przewodu USB i uruchomieniu programu. Jednym z takich programów jest evasi0n, przedstawiony na poniższym rysunku:



Program wyświetla wskazówki, według których należy postępować. Instaluje również na urządzeniu program Cydia. Służy on do pobierania programów z alternatywnego sklepu zawierającego aplikacje inne niż oficjalnie zatwierdzone przez Apple. Większość dostępnych w tym sklepie aplikacji, np. do stosowania własnych motywów, widżetów, programów itp., została opracowana przez programistów korzystających z dostępu administracyjnego.

Jak to działa?

Przyjrzyjmy się dokładniej procesowi uzyskiwania dostępu administracyjnego.

Dostęp do urządzenia Android

Dzięki możliwości uruchamiania aplikacji z uprawnieniami administratora można w systemie Android wykonywać dowolne operacje.

Proces uzyskiwania dostępu administracyjnego w systemie Android polega zazwyczaj na wykonaniu dwóch kroków:

1. Wyszukanie luki w zabezpieczeniach urządzenia, umożliwiającej uruchomienie specjalnego kodu z uprawnieniami administratora.
2. Wykorzystanie luki do zainstalowania programu *su*.

Jeżeli urządzenie ma niezablokowany program rozruchowy (bootloader), wtedy cały proces jest bardzo prosty. Niezablokowany program rozruchowy pozwala na zapisywanie w pamięci ROM dowolnego kodu, między innymi programu *su*. Wystarczy jedynie, trzymając przyciśnięte odpowiednie klawisze, uruchomić ponownie urządzenie w trybie umożliwiającym zapisywanie pamięci ROM, a następnie za pomocą odpowiedniego oprogramowania zapisać nową zawartość pamięci ROM.

Ale co robić, gdy urządzenie ma zablokowany program rozruchowy? Wtedy nie ma możliwości uruchomienia żadnego kodu z rozszerzonymi uprawnieniami.

Wiele programów, np. usług systemowych, działa z uprawnieniami administracyjnymi, aby mogły mieć dostęp do podzespołów urządzenia.

Wszystkie metody uzyskiwania dostępu administracyjnego polegają na wykorzystaniu luki w bezpieczeństwie któregoś z takich programów, uruchomieniu kodu montującego system plików w trybie odczytu i zapisu, a następnie zainstalowaniu programu *su*.

Dostęp do urządzenia iOS

Proces uzyskiwania dostępu administracyjnego do urządzenia z systemem iOS przebiega różnie w zależności od użytego narzędzia i wersji systemu. Teraz przeanalizujemy działanie jednego z takich narzędzi, wykorzystywanego do uzyskania dostępu do telefonu iPhone 5.

Najpierw należy użyć biblioteki *libimobiledevice*, która wykorzystuje błąd w kodzie tworzącym kopię zapasową systemu iOS, i uzyskać w ten sposób dostęp do niedostępnego standardowo pliku z ustawieniami strefy czasowej.

Następnie należy w odpowiednim miejscu systemu plików umieścić odnośnik pozwalający innym programom na komunikowanie się z procesem *launchd* ładowanym do pamięci podczas uruchamiania urządzenia, który umożliwia uruchamianie aplikacji z uprawnieniami administracyjnymi.

Biblioteka *libimobiledevice* obsługuje protokół komunikacyjny stosowany w urządzeniach iOS i jest dostępna dla różnych systemów operacyjnych. Otwiera ona innym programom dostęp do systemu plików urządzenia, umożliwia odczytywanie informacji o urządzeniu i jego wewnętrznych ustawieniach, tworzenie i odtwarzanie kopii zapasowych, zarządzanie zainstalowanymi aplikacjami, odczytywanie książki telefonicznej, kalendarza, notatek, zakładek oraz synchronizowanie muzyki i filmów. Więcej informacji na temat tej biblioteki znajdziesz na stronie *http://www.libimobiledevice.org*.

Od tej chwili po każdorazowym utworzeniu kopii zapasowej systemu iOS wszystkie programy będą miały dostęp do pliku z ustawieniami strefy czasowej oraz procesu launchd.

Pomysłowe, prawda?

W systemie iOS stosowane jest podpisywanie kodu, uniemożliwiające niezaufanym aplikacjom dostęp do procesu launchd. Aby więc uniknąć konieczności podpisywania kodu, narzędzie do uzyskiwania dostępu administracyjnego uruchamia nową, na pozór niegroźną aplikację. W momencie gdy użytkownik zostanie poproszony o kliknięcie ikony tej aplikacji, stosowana jest technika wykorzystująca ciąg znaków *shebang*, umożliwiających uruchomienie kodu za pomocą innej, podpisanej aplikacji, którą w tym przypadku jest proces launchd. Shebang jest to ciąg znaków umieszczanych na początku skryptu, składający się z kratki i wykrzyknika (`#!`).

W systemie Unix wiersz skryptu zawierający ciąg shebang jest traktowany jako dyrektywa interpretera. Po wywołaniu skryptu uruchamiany jest odpowiedni interpreter z parametrem zawierającym ścieżkę do pliku z programem do uruchomienia.

Jeżeli na przykład jest to *ścieżka/do/pliku*, a skrypt zaczyna się od wiersza `#!/bin/sh`, wtedy uruchamiany jest program */bin/sh* z argumentem *ścieżka/do/pliku*.

Program launchd (po uzyskaniu do niego dostępu) jest wykorzystywany do zmieniania ustawień umożliwiających wprowadzanie zmian w plikach systemowych.

Aby dostęp administracyjny był stale otwarty, modyfikowany jest plik *launchd.conf* zawierający ustawienia konfiguracyjne programu launchd. Dzięki temu po każdym restarcie urządzenia nie trzeba uruchamiać programu otwierającego dostęp administracyjny.

Teraz następuje ostatnia faza operacji polegająca na usunięciu zabezpieczeń jądra systemu. W systemie iOS stosuje się program AMFID (ang. *Apple Mobile File Integrity Daemon*), który uniemożliwia korzystanie z niezatwierdzonych aplikacji. Wywoływany jest ponownie program launchd, który do programu AMFID ładuje bibliotekę funkcji umożliwiających uruchamianie wszystkich aplikacji.

Innym zabezpieczeniem stosowanym w jądrze systemu jest technika ASLR (ang. *Address Space Layout Randomization*), która uniemożliwia modyfikację zawartości pamięci poprzez „ukrywanie" kodu aplikacji po każdorazowym uruchomieniu urządzenia. W ten sposób zabezpiecza się urządzenie przed nadpisaniem fragmentów kodu aplikacji.

Następnie program otwierający dostęp administracyjny stosuje pomysłową sztuczkę do określenia położenia specjalnego obszaru pamięci, tzw. wektora przerwań ARM. W obszarze tym zapisywane są informacje o częściach pamięci, w których miały miejsce awarie aplikacji.

Program symuluje awarię aplikacji, sprawdza zawartość wektora przerwań i odczytuje minimalną ilość informacji wystarczającą do rozpoznania pozostałych części jądra systemu.

W ostatnim kroku program wykorzystuje błąd w kodzie obsługującym interfejs USB. Kod ten przekazuje programowi adres pamięci wykorzystywanej przez jądro systemu i oczekuje, że program zwróci ten adres w niezmienionej postaci.

W ten sposób program może w jądrze systemu zmodyfikować zabezpieczenia, które uniemożliwiają wprowadzanie zmian w kodzie, i uzyskać nad systemem pełną kontrolę, co było jego celem!

Skorowidz

A

B

C

D

E

G

I

J

K

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Bezpieczeństwo urządzeń mobilnych
Receptury

Urządzenia przenośne, takie jak smartfony i tablety, już dawno przestały służyć tylko do prostej komunikacji. Dziś są coraz bardziej zaawansowane technicznie, mają też coraz więcej skomplikowanych aplikacji. Niestety, skutkiem ubocznym tego rozwoju jest pojawianie się luk w ich zabezpieczeniach. Tymczasem konsekwencje skutecznego ataku na urządzenie mobilne bywają bardzo poważne. Nic dziwnego, że ostatnimi czasy temat zabezpieczania aplikacji i urządzeń przed atakami jest bardzo popularny zarówno wśród programistów, jak i samych użytkowników.

Niniejsza książka jest znakomitym kompendium wiedzy o bezpieczeństwie urządzeń przenośnych. Przedstawiono tu różnorodne techniki ingerencji w popularne urządzenia i zasady wykonywania testów penetracyjnych. Dowiesz się stąd, jak wykryć luki w zabezpieczeniach i ochronić urządzenia przed atakami. Autorzy przedstawili także istotne informacje o analizie kodu aplikacji oraz metodach śledzenia ataków i przechwytywania danych przesyłanych przez urządzenia. Sporo miejsca poświęcono poszczególnym rodzajom ataków na urządzenia pracujące na takich platformach jak Android, iOS, BlackBerry i Windows.

Sprawdź, czy właśnie w tej chwili ktoś nie atakuje Twojego smartfona!

W książce między innymi:

- systemy Android i iOS — korzystanie z pakietów SDK i testy bezpieczeństwa
- pakiety SDK dla systemów BlackBerry i Windows
- przygotowanie i prowadzenie testów penetracyjnych
- zabezpieczanie ruchu sieciowego
- ochrona danych przesyłanych bezprzewodowo

Prashant Verma jest CISSP i od wielu lat zajmuje się tematyką bezpieczeństwa urządzeń przenośnych. Zabierał głos na prestiżowych konferencjach OWASP Asia Pacific w 2012 r. w Sydney oraz RSA Conference Asia Pacific w Japonii i Singapurze w 2014 r. Chętnie dzieli się wiedzą i wynikami badań podczas szkoleń, warsztatów i wykładów.

Akshay Dixit jest specjalistą w dziedzinie bezpieczeństwa informatycznego, wykładowcą i badaczem. Świadczy usługi doradztwa w tym zakresie dla wielu instytucji publicznych oraz firm. Obecnie pracuje nad sztuczną inteligencją i podatnością urządzeń przenośnych na ataki. Prowadzi szkolenia, prezentacje i warsztaty.

[PACKT] PUBLISHING

Helion

księgarnia internetowa
http://helion.pl

zamówienia telefoniczne
0 801 339900
0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-3216-4

cena: 39,90 zł