Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

ЗВОРОТНА РОЗРОБКА ТА АНАЛІЗ ШКІДЛИВОГО ЗАБЕЗПЕЧЕННЯ

Лабораторна робота №8
Мобільні застосування

Виконала:
студентка 3 курсу
гр. ФБ-92
Шатковська Діана

Перевірив:
Якобчук Д.І.

Київ - 2021

Мобільні застосування

Мета роботи:

Отримати навички зворотного проектування та аналізу мобільних застосунків.

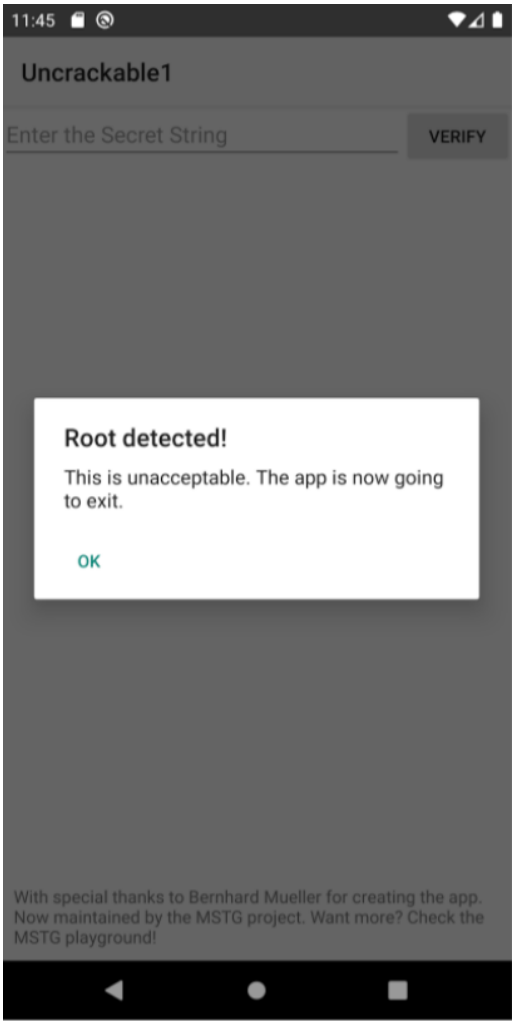
Хід роботи

Завдання 1

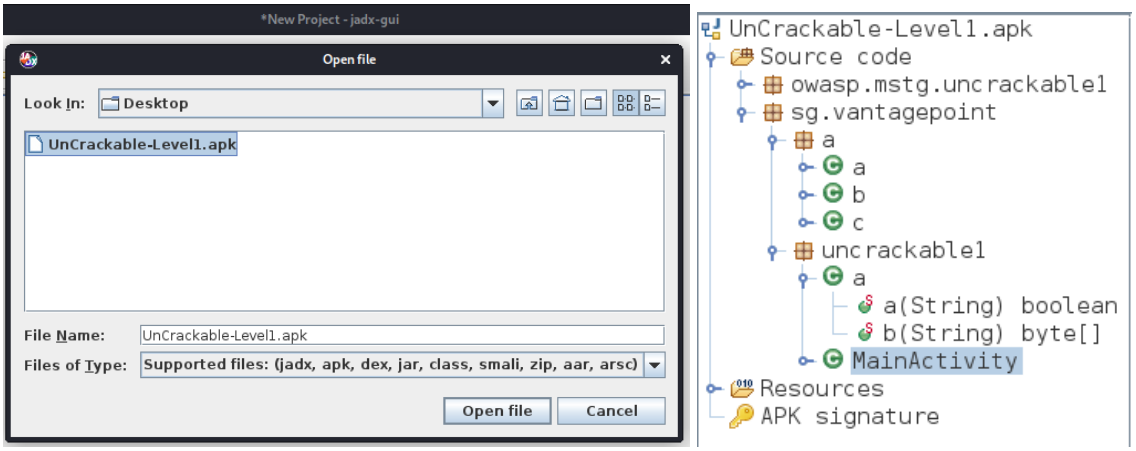
Проаналізуйте UnCrackable Mobile Apps, Hacking Playground з OWASP MSTG.

- UnCrackable-Level1

Запустимо додаток:



Не можемо його використовувати, бо маємо рут-права. Задекомпілюємо додаток за допомогою jadx-gui :



Переглянемо код MainActivity. У методі onCreate можна помітити рядок “Root detected!”, що нам й вивела програма при запуску. Як бачимо цей вивід виконується коли **або** c.a() **або** c.b() **або** c.c() повертають True (цей код ймовірно був деобфускований, адже назви методів спрощені до одного символу, що утруднює читання).

```
29    /* access modifiers changed from: protected */
30    public void onCreate(Bundle bundle) {
31        if (c.a() || c.b() || c.c()) {
32            a("Root detected!");
33        }
34        if (b.a(getApplicationContext())) {
35            a("App is debuggable!");
36        }
37        super.onCreate(bundle);
38        setContentView(R.layout.activity_main);
39    }
```

Переглянемо клас c і його методи:

```
6  public class c {
7      public static boolean a() {
8          for (String str : System.getenv("PATH").split(":")) {
9              if (new File(str, "su").exists()) {
10                 return true;
11             }
12         }
13         return false;
14     }
15
16     public static boolean b() {
17         String str = Build.TAGS;
18         return str != null && str.contains("test-keys");
19     }
20
21     public static boolean c() {
22         for (String str : new String[]{"/system/app/Superuser.apk", "/system/xbin/daemonsu", "/system/etc/init.d/99SuperSUDaem
23             if (new File(str).exists()) {
24                 return true;
25             }
26         }
27         return false;
28     }
29 }
```

Якщо розібрати детальніше, то у методі a() програма намагається у кожній директорії, що записана у $PATH, перевірити існування файлу-утиліти під назвою su, що, як відомо, використовується для переходу на рут-юзера.
Метод b() намагається дістати тег-підпис ядра під час компіляції програми, значення ‘test keys’ записується якщо програма була скомпільована стороннім розробником.
Метод c() як і метод a() перевіряє наявність певних файлів(перерахованих у масиві), які судячи з назв можуть дозволити отримати права рут-юзера.

Необхідно обійти цю перевірку на рутовану систему: можемо зробити так, щоб всі ці перевірки повертали false.
Однак, якщо поглянути на метод вище MainActivity.a(), можемо також помітити, що саме цей метод викидає нам помилку та зупиняє програму за допомогою system.exit() - спробуємо замінити функціонал цього методу, щоб повідомлення про помилку не перешкоджало нам - це зробити буде трохи легше.

```
13 public class MainActivity extends Activity {
14     private void a(String str) {
15         AlertDialog create = new AlertDialog.Builder(this).create();
16         create.setTitle(str);
17         create.setMessage("This is unacceptable. The app is now going to exit.");
18         create.setButton(-3, "OK", new DialogInterface.OnClickListener() {
19             /* class sg.vantagepoint.uncrackable1.MainActivity.AnonymousClass1 */
20
21             public void onClick(DialogInterface dialogInterface, int i) {
22                 System.exit(0);
23             }
24         });
25         create.setCancelable(false);
26         create.show();
27     }
28
```

Використаємо adb та Frida, виведемо список процесів, щоб ідентифікувати додаток:

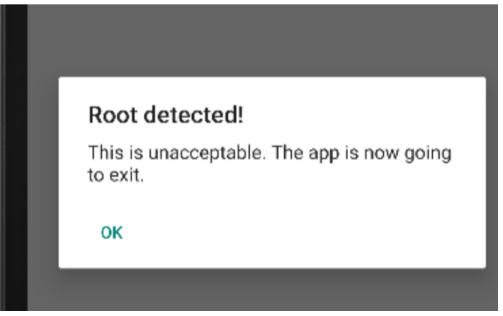
```
H:\platform-tools>adb push frida-server /data/local/tmp
frida-server: 1 file pushed, 0 skipped. 396.8 MB/s (46416812 bytes in 0.112s)

H:\platform-tools>adb shell
generic_x86:/ $ cd /data/local/tmp
generic_x86:/data/local/tmp $ ls
frida-server
generic_x86:/data/local/tmp $ chmod 755 ./frida-server
generic_x86:/data/local/tmp $ ./frida-server
Unable to load SELinux policy from the kernel: Failed to open file ?/sys/fs/selinux/policy?: Permission denied
```

```
127|generic_x86:/data/local/tmp $ su root
generic_x86:/data/local/tmp # ./frida-server
```

```
Microsoft Windows [Version 10.0.19043.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HOME>frida-ps -Ua
 PID  Name          Identifier
----  ------------  ------------------------------------
6861  Chrome        com.android.chrome
7485  Gmail         com.google.android.gm
6365  Google        com.google.android.googlequicksearchbox
6365  Google        com.google.android.googlequicksearchbox
3696  Messages      com.google.android.apps.messaging
3933  Phone         com.android.dialer
7467  Photos        com.google.android.apps.photos
7700  Uncrackable1  owasp.mstg.uncrackable1
```



Виконаємо JS-ін’єкцію та перевизначимо необхідний метод у додатку:

```
Java.perform(function() {
    Java.use("sg.vantagepoint.uncrackable1.MainActivity").a.implementation =
function(s) {
      console.log("Tamper detection suppressed, message was: " + s);
    }
  });
```

```
H:\WINAPI2>frida -U --no-pause -l deploy.js -f owasp.mstg.uncrackable1
    ____
   / _  |   Frida 15.1.14 - A world-class dynamic instrumentation toolkit
  | (_| |
   > _  |   Commands:
  /_/ |_|       help      -> Displays the help system
  . . . .       object?   -> Display information about 'object'
  . . . .       exit/quit -> Exit
  . . . .
  . . . .   More info at https://frida.re/docs/home/
Spawned `owasp.mstg.uncrackable1`. Resuming main thread!
[Android Emulator 5554::owasp.mstg.uncrackable1]-> Tamper detection suppressed, message was: Root detected!
```

Ми можемо використовувати додаток!! Тепер нам потрібно знайти секрет.

```
41    public void verify(View view) {
42        String str;
43        String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();
44        AlertDialog create = new AlertDialog.Builder(this).create();
45        if (a.a(obj)) {
46            create.setTitle("Success!");
47            str = "This is the correct secret.";
48        } else {
49            create.setTitle("Nope...");
50            str = "That's not it. Try again.";
51        }
52        create.setMessage(str);
53        create.setButton(-3, "OK", new DialogInterface.OnClickListener() {
54            /* class sg.vantagepoint.uncrackable1.MainActivity.AnonymousClass2 */
```

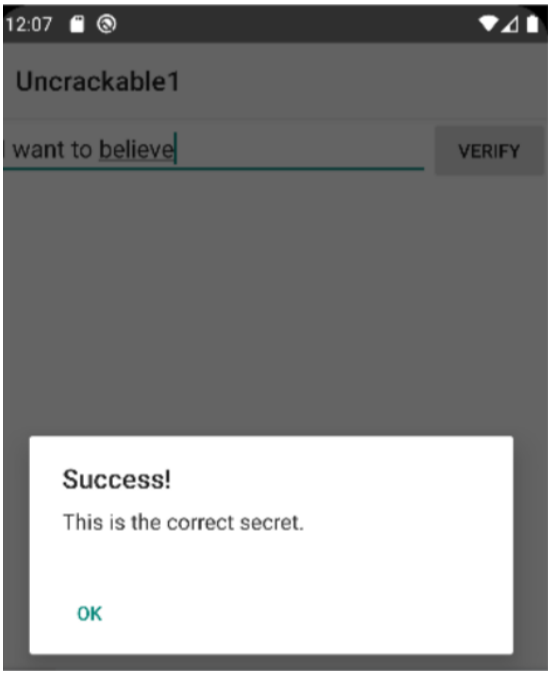
Метод verify() відповідальний за перевірку введеного секрету, що ми можемо зрозуміти з рядків-повідомлень. Бачимо, що ми отримуємо введений рядок в змінну obj, яка верифікується в функції a.a(obj) - переглянемо її.

```
6  public class a {
7      public static boolean a(String str) {
8          byte[] bArr;
9          byte[] bArr2 = new byte[0];
10         try {
11             bArr = sg.vantagepoint.a.a.a(b("8d127684cbc37c17616d806cf50473cc"), Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv
12         } catch (Exception e) {
13             Log.d("CodeCheck", "AES error:" + e.getMessage());
14             bArr = bArr2;
15         }
16         return str.equals(new String(bArr));
17     }
```

Бачимо що тут даний рядок порівнюється з деякою змінною. При її створенні використовується бейс-64 рядок. Спробуємо його розшифрувати:

```
┌──(root㉿kali)-[~/lab8]
└─# echo 5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2GOc= | openssl enc -aes-128-ecb -base64 -d -K 8d127684cbc37c17616d806cf50473cc
I want to believe
```

Отримали рядок, що потенційно і є нашим секретом, введемо його у додаток:

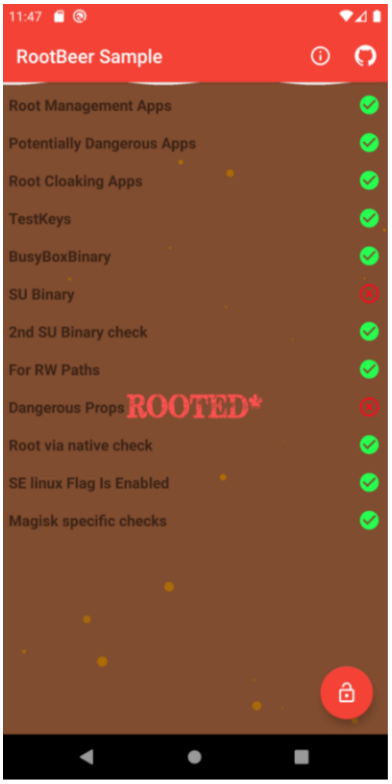


У нас вийшло.

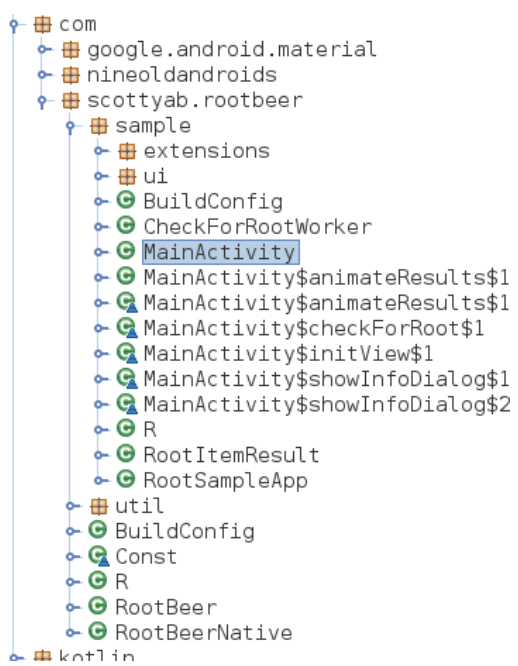
Завдання 2

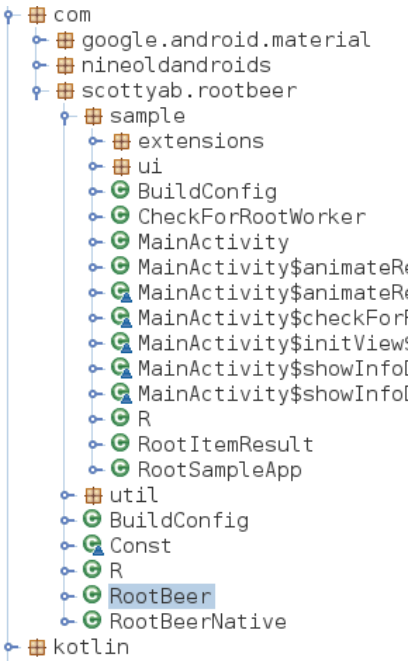
Реалізуйте обхід перевірок RootBeer за допомогою Frida.

Запустимо додаток:



Знову спробуємо задекомпілювати додаток за допомогою jadx-gui та переглянемо основну функцію:





У класі RootBeer можемо помітити метод перевірки на рутованість:

```
43    public boolean isRooted() {
          return detectRootManagementApps() || detectPotentiallyDangerousApps() || checkForBinary("su") || checkForDange
      }


|| checkForDangerousProps() || checkForRWPaths() || detectTestKeys() || checkSuExists() || checkForRootNative(

tive() || checkForMagiskBinary();
```

Нижче можна переглянути імплементацію кожного з них, наприклад:

```
public boolean detectRootManagementApps() {
    return detectRootManagementApps(null);
}

public boolean detectRootManagementApps(String[] strArr) {
    ArrayList arrayList = new ArrayList(Arrays.asList(Const.knownRootAppsPackages));
    if (strArr != null && strArr.length > 0) {
        arrayList.addAll(Arrays.asList(strArr));
    }
    return isAnyPackageFromListInstalled(arrayList);
}
```

Логічно, що isRooted() поверне False, якщо кожен метод в умові буде повертати False, можемо перевизначити ці методи(або необхідні з них, або й сам метод isRooted()):

Визначимо ідентифікатор процесу:

```
H:\WINAPI2>frida-ps -Ua
 PID  Name             Identifier
----  ---------------  ------------------------------------
6861  Chrome           com.android.chrome
7485  Gmail            com.google.android.gm
6365  Google           com.google.android.googlequicksearchbox
6365  Google           com.google.android.googlequicksearchbox
3933  Phone            com.android.dialer
7467  Photos           com.google.android.apps.photos
8043  RootBeer Sample  com.scottyab.rootbeer.sample
```

Як і в попередньому завданні, виконаємо JS-ін’єкцію для перевизначення необхідних методів додатку:

```
Java.perform(function(){
    var RootBeer = Java.use("com.scottyab.rootbeer.RootBeer");

    RootBeer.detectTestKeys.overload().implementation = function(){
        return false;
    };

    RootBeer.checkForBusyBoxBinary.overload().implementation = function(){
        return false;
    };

    RootBeer.checkForSuBinary.overload().implementation = function(){
        return false;
    };

    RootBeer.checkSuExists.overload().implementation = function(){
        return false;
    };

    RootBeer.checkForDangerousProps.overload().implementation = function(){
        return false;
    };

    RootBeer.checkForRootNative.overload().implementation = function(){
        return false;
    };

    RootBeer.isRooted.overload().implementation = function(){
        return false;
    };
});
```

```
H:\WINAPI2>frida -U --no-pause -l deploy1.js -f com.scottyab.rootbeer.sample
     ____
    / _  |   Frida 15.1.14 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
Spawned `com.scottyab.rootbeer.sample`. Resuming main thread!
[Android Emulator 5554::com.scottyab.rootbeer.sample]->
```

Після ін’єкції програма показує, що наша система не рутована:

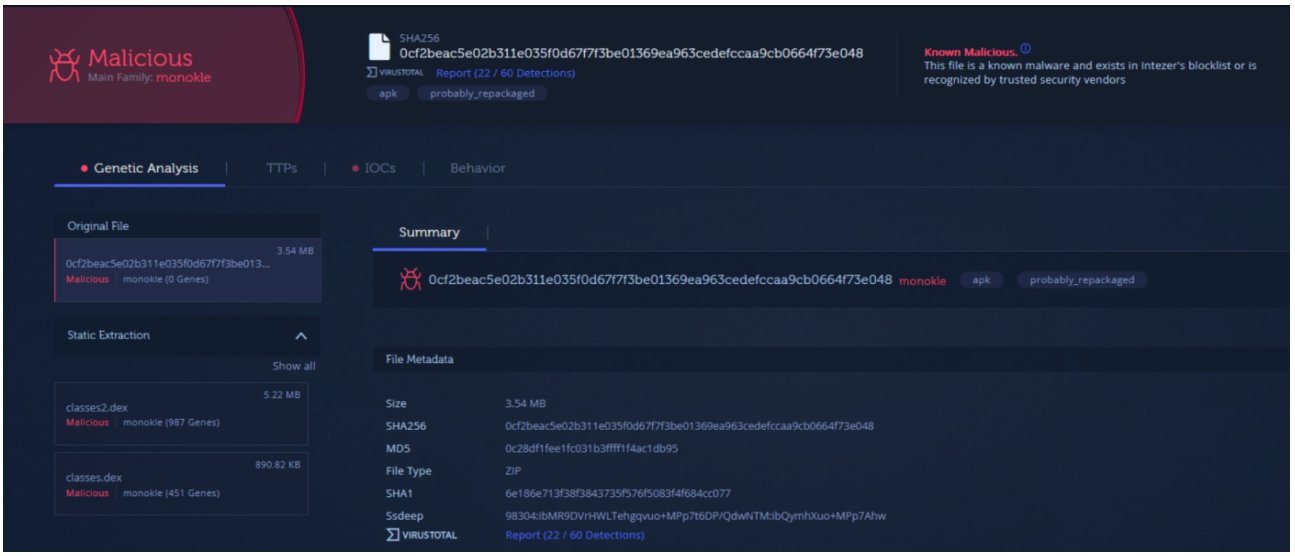
Завдання 3

Проаналізуйте зразки Monokle:

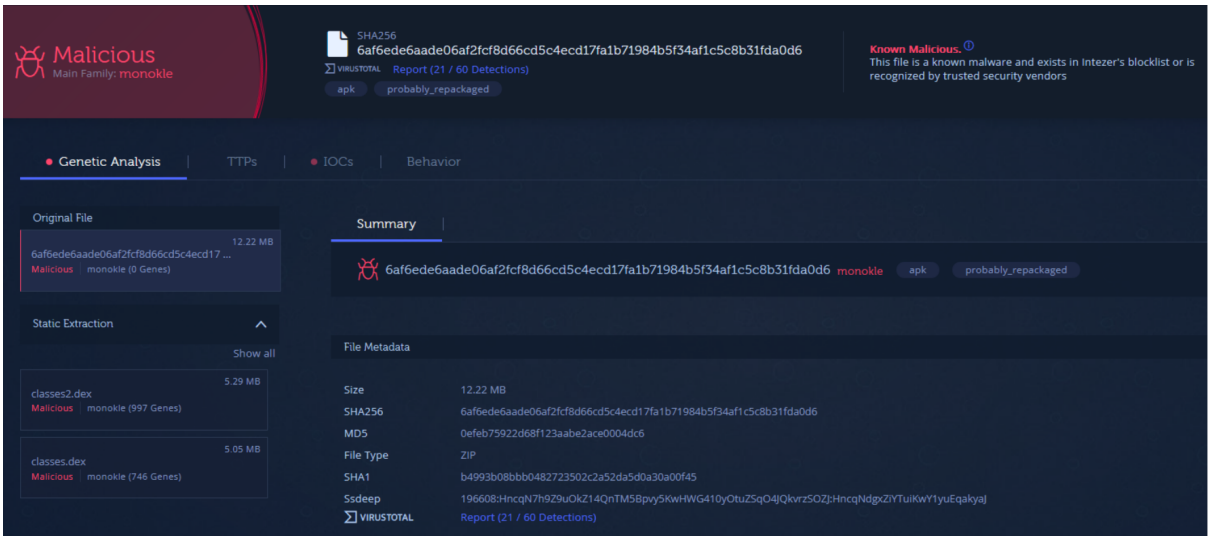
0c28df1fee1fc031b3ffff1f4ac1db95
0efeb75922d68f123aabe2ace0004dc6
1464cd00ab0a1a4137b17976bf507311

Використавши [Open-Source Threat Intelligence Community] було віднайдено файли за їх хешами.

0c28df1fee1fc031b3ffff1f4ac1db95



0efeb75922d68f123aabe2ace0004dc6



1464cd00ab0a1a4137b17976bf507311