A Brief Tutorial for Using EviCheck

Mohamed Nassim Seghir

University of Edinburgh

Abstract. EviCheck is a tool for the verification, certification and generation of lightweight fine-grained policies for Android. In this tutorial we try to cover the different aspects of EviCheck with illustrative examples and commands that can be directly used as they are to reproduce some experiments.

1 Obtaining EviCheck

EviCheck can be downloaded here:

http://homepages.inf.ed.ac.uk/mseghir/EviCheck

Instructions on the installation and usage are included in file README that comes with. All the examples used in this tutorial are provided with EviCheck, so if you are in EviCheck's main directory you can test any command shown here by a simple copy and paste.

2 Verification, Certification and Digital Evidence

Let us consider as a running example the audio recording app Recorder whose code snippets and the associated graphical interface are illustrated in Figure 1. The access to the recording device is carried out via object recorder (line 2). At the creation phase (onCreate), a callback for a click event is associated with the button Start (line 5). Within the callback onClick, the method startRecording is invoked (line 24) which in turns calls recorder.setAudioSource and recorder.start to set the (on-device) microphone as a source and trigger the recording process. This app requires the permission RECORD_AUDIO which is associated with the API method setAudioSource.

2.1 Verification

We consider the policy recorder.pol composed of the single rule:

```
EVICHECK_ENTRY_POINT ~EVICHECK_ONCLICK_HANDLER : ~RECORD_AUDIO
```

such that ∼ represents the negation operator. To verify if the policy is respected by the app, we call EviCheck as follows:

```
public class Recorder extends Activity {
  private MediaRecorder recorder = null;
  ....
  public void onCreate(....) {
        ((Button) findViewById(Start))
          .setOnClickListener(startClick);
              ....
        // startRecording();
  }

  private void startRecording() {
        recorder = new MediaRecorder();
        recorder.setAudioSource
        (MediaRecorder.AudioSource.MIC);
        recorder.setOutputFile(/* file name */);
        ....
        recorder.start();
  }

  private View.OnClickListener startClick
          = new View.OnClickListener() {
  public void onClick(View v) {
        ....
        startRecording();
  }};
  ....
}
```

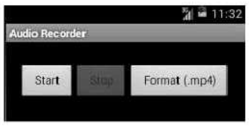

Fig. 1. Code snippets and graphical interface of the Recorder app

```
python2.7 EviCheck.py -f examples/apps/Recorder.apk -g
          -p examples/policies/recorder.pol -t recorder.cert -m
```

The option -f is for the APK file to be analysed, -g means certificate generation (verification) mode, -p is followed by the policy file, -t is for the certificate which is stored in file recorder.cert and -m tells EviCheck to use the API-to-permissions map as initial map. The user has also the option to provide his own initial map using the option -i. As a result, EviCheck returns:

```
Policy valid!
=============================
APK file: example_app/Recorder.apk
Analysis time: 0.00019907951355
Policy checking time: 0.000382900238037
Number of rules: 1
Number of valid rules: 1
Number of violated rules: 0
Number of methods: 73
=============================
```

Which indicates that the policy is valid, followed by some additional information.

2.2 Certification

Let us now try to check the validity of the generated certificate. This is done as follows:

```
python2.7 EviCheck.py -f examples/apps/Recorder.apk -c
          -p examples/policies/recorder.pol -t recorder.cert -m
```

As you can see, this time we use -c instead of -g to indicate that we are in the certificate checking mode. Hence the certificate is read from the file recorder.cert. As a result we obtain:

```
Certificate valid!
Policy valid!
============================
APK file: example_app/Recorder.apk
Analysis time: 0.000128030776978
Policy checking time: 0.000263214111328
Number of rules: 1
Number of valid rules: 1
Number of violated rules: 0
Number of methods: 73
============================
```

Meaning that both the certificate and the policy are valid.

2.3 Tempering with the certificate

Let us have a look inside the certificate (recorder.cert), for the sake of the presentation we omit the prefixes representing package names.

```
....
File-><init>(Ljava/lang/String; Ljava/lang/String;)V:
Recorder->startRecording()V: RECORD_AUDIO
Toast->show()V:
Recorder->stopRecording()V:
Recorder->getString(I)Ljava/lang/String:
....
```

As we can see, the permission RECORD_AUDIO is associated with method startRecording. Let us try to trick EviCheck by removing RECORD_AUDIO from that entry. If we check the certificate now, we obtain

```
Invalid certificate!
Problem located in method: Recorder->startRecording()V
```

So EviCheck is able to detect that the certificate is corrupted and gives the location of the inconsistency. Let us try another trick by removing the entire entry. EviCheck answers:

```
Invalid certificate!
Entry: Recorder->startRecording()V is not present
```

So it is also able to distinguish this case.

2.4 Tempering with the app

Let u consider the new app Recorder_bad corresponding to the previous example Figure 1 where line 8 is uncommented. We call EviCheck with the verification option (-g):

```
python2.7 EviCheck.py -f examples/apps/Recorder_bad.apk -g
          -p examples/policies/recorder.pol -t recorder.cert -m
```

and we obtain:

```
Rule 1 ==> ~EVICHECK_ONCLICK_HANDLER EVICHECK_ENTRY_POINT : ~RECORD_AUDIO
Policy violated!
Tag RECORD_AUDIO is in Recorder->onCreate(Landroid/os/Bundle;)V
```

Here EviCheck points to the violated rule and the cause of its violation, which is the presence of the tag (permission) RECORD_AUDIO in the entry corresponding to the method onCreate which represents an entry point.

3 Policy Generation

In what follows we will show how to generate anti-malware policies using EviCheck. This requires a training set of malware and benign applications whose pecifications will be extracted.

3.1 Specification extraction

A specification is a summary of the usage of different permissions in various contexts within an app. They somehow represent a condensed (compressed) form of the generated certificate. For illustration, we consider again the app Recorder and we call EviCheck as follows:

```
python2.7 EviCheck.py -f examples/apps/Recorder.apk -g -m -r recorder.spec
```

The option `-r` indicates that the specification of the app must be generated and stored in the file recorder.spec which looks like:

```
//------------------------------------------------------------
//--------------------- examples/apps/Recorder.apk
//------------------------------------------------------------
EVICHECK_ACTIVITY_METHOD : RECORD_AUDIO
EVICHECK_ONCREATE_METHOD :
EVICHECK_ONCLICK_HANDLER : RECORD_AUDIO
```

The specification simply says that the permission RECORD_AUDIO is used in a click handler and in an activity.

3.2 Inferring a policy

As mentioned previously, to infer a policy we require a training set of benign and malware applications. The goal is to find a policy under which a maximum of malware is excluded and a maximum of benign applications is allowed. We formulate this as an optimisation problem and use the SMT solver **Z3** as back-end to solve it. For this, EviCheck takes as input two files containing a batch of specifications, one for benign apps and the other one for malware. We consider the two files representing the training set we got from McAfee and Drebin [1], 1000 malware and 1000 benign. The two files are spec_perm_train.ben and spec_perm_train.mal for benign and malware respectively. We call EviCheck as follows:

```
python2.7 EviCheck.py -s examples/specs/train/spec_perm_train -p policy_perm.pol
-z3
```

Below is a snippet from the generated policy policy_perm.pol:

```
 ........
EVICHECK_PAUSE_METHOD : ~BLUETOOTH
EVICHECK_RECEIVER_METHOD : ~SEND_SMS
EVICHECK_SERVICE_METHOD : ~CHANGE_WIFI_STATE
EVICHECK_START_METHOD : ~BLUETOOTH
EVICHECK_DESTROY_METHOD : ~READ_LOGS
EVICHECK_DO_INBACKGROUND : ~DISABLE_KEYGUARD
EVICHECK_ONCREATE_METHOD : ~RESTART_PACKAGES
EVICHECK_DO_INBACKGROUND : ~VIBRATE
EVICHECK_ONTOUCH_HANDLER : ~SEND_SMS
EVICHECK_ONCLICK_HANDLER : ~WRITE_HISTORY_BOOKMARKS
EVICHECK_ONCLICK_HANDLER : ~READ_LOGS
EVICHECK_ONCLICK_HANDLER : ~SEND_SMS
EVICHECK_ACTIVITY_METHOD : ~WRITE_HISTORY_BOOKMARKS
EVICHECK_RECEIVER_METHOD : ~ACCESS_COARSE_LOCATION
EVICHECK_ONCREATE_METHOD : ~SEND_SMS
EVICHECK_SERVICE_METHOD : ~READ_LOGS
EVICHECK_SERVICE_METHOD : ~READ_SMS
EVICHECK_ONTOUCH_HANDLER : ~BLUETOOTH
EVICHECK_SERVICE_METHOD : ~CHANGE_WIFI_MULTICAST_STATE
EVICHECK_ONCLICK_HANDLER : ~ACCESS_COARSE_LOCATION
EVICHECK_RESTART_METHOD : ~VIBRATE
EVICHECK_ACTIVITY_METHOD : ~EXPAND_STATUS_BAR
EVICHECK_ONCREATE_METHOD : ~USE_CREDENTIALS
.......
```

[1] http://user.informatik.uni-goettingen.de/ darp/drebin/

The batch files for our training set as well as the inferred policy are provided with EviCheck.

3.3 Testing the policy

We also provide two testing sets (benign and malware) as batch files, namely spec_perm_test.ben and spec_perm_test.mal. If we want to test the policy on the set of malware, we use the command below with option option -ts for testing:

```
python2.7 EviCheck.py -s examples/specs/test/spec_perm_mal.ben
-p examples/policies/policy_perm.pol -z3 -ts
```

We obtain:

```
 Policy violated by 889 apps from 1000
```

3.4 API-base Policies

We can also infer policies which use APIs instead of permissions. For this, we are providing two training sets (batch files) of API-based specs which are spec_api_train.ben and spec_api_train.mal. To generate the policy, we use the option -api in the command below:

```
python2.7 EviCheck.py -s examples/specs/train/spec_api_train
-p examples/policies/policy_api.pol -z3 -api
```

We also have two testing batch files, spec_api_test.ben and spec_api_test.mal, of API-based specs generated from the same sets of apps used to generate permission-based specs. For example, to test the policy policy_api.pol on the set of malware we use the command:

```
python2.7 EviCheck.py -s examples/specs/test/spec_api_test.mal
-p examples/policies/policy_api.pol -z3 -api -ts
```

We obtain:

```
 Policy violated by 828 apps from 1000
```

3.5 Malware detection and explanation

We consider two malware families: FakeInstaller and Plankton. During the verification of most of the instances of FakeInstaller with respect to the permission-based policy, we obtain

```
Policy violated!
Rule 1 ==> EVICHECK_RECEIVER_METHOD : ~SEND_SMS
```

```
Tag SEND_SMS is in Checker->onReceive(Landroid/content/Context; ...)V

Rule 13 ==> EVICHECK_ACTIVITY_METHOD : ~SEND_SMS
Tag SEND_SMS is in Main->start()V

Rule 21 ==> EVICHECK_ONCLICK_HANDLER : ~SEND_SMS
Tag SEND_SMS is in Main$1->onClick(Landroid/view/View;)V
```

We contrast this with descriptions provided by anti-virus companies[2]:*"The user is forced to click an Agree or Next button, which sends the premium SMS messages. We have also seen versions that send the messages before the victim clicks a button"*. This is quite in accordance with rules 1, 13 and 21. Hence the policy is targeting the relevant aspect of the malicious behaviour, thus providing a good explanation on why the app is bad.

Similarly for the familly Plankton, we have found that among the relevant rules are: 9 and 17 which respectively correspond to *"forward information to a remote server"* and *"collect or modify the browser's bookmarks"* in the description[3].

```
Policy violated!
Rule 9 ==> EVICHECK_SERVICE_METHOD : ~ACCESS_WIFI_STATE
Tag ACCESS_WIFI_STATE is in AndroidSDKProvider->i()...

Rule 17 ==> EVICHECK_SERVICE_METHOD : ~READ_HISTORY_BOOKMARKS
Tag READ_HISTORY_BOOKMARKS is in PushService->onStart(...)V

Rule 22 ==> EVICHECK_RECEIVER_METHOD : ~ACCESS_FINE_LOCATION
Tag ACCESS_FINE_LOCATION is in AlarmReceiver->onReceive(...)V
```

[2] http://blogs.mcafee.com/mcafee-labs/fakeinstaller-leads-the-attack-on-android-phones

[3] http://www.f-secure.com/v-descs/trojan_android_plankton.shtml