Freemium SDK configuration

Classes and method overview

The following classes and structures are available (in alphabetical order):

- *RASPDetection*;
- *RASPFacade*;
- *RASPFacadeFactory*;

RASPDetection

`RaspDetection` represents type of tampering which is recognized by SDK. In free version of SDK only ROOT tampering detection is available:

```
public enum RASPDetection {
    ROOT
}
```

- ***ROOT*** – application is running on rooted device;

RASPFacadeFactory

`RASPFacadeFactory` is a factory class which is used to create instance of `RASPFacade`. It has only two methods:

```
public final class RASPFacadeFactory {

    private final Context context;

    public static RASPFacadeFactory create(@NonNull final Context context) {
        return new RASPFacadeFactory(context);
    }

    private RASPFacadeFactory(@NonNull final Context context) {
        this.context = context;
    }

    public RASPFacade createRASPFacade() throws RASPFacadeException {
        return RASPFacade.newInstance(ObjectFactory.create(context));
    }
}
```

- `create` method – initializes `RASPFacadeFactory` object;

- `createRASPFacade` method – initializes `RASPFacade` object from current `RASPFacadeFactory`;

RASPFacade

`RASPFacade` is the main entry point for App Protector. It is initialized with `RASPFacadeFactory` as mentioned above. After we initialize this class an application can detect **ROOT** tampering. `RASPFacade` contains `doDetectOnDemand` method:

```
Set<RASPDetection> doDetectOnDemand(Set<RASPDetection> raspDetections);
```

doDetectOnDemand

`doDetectOnDemand` method is used for a single on demand App Protector detection. `raspDetections` parameter represents types of tampering that will be detected by the SDK in this single on-demand tampering detection. The method will return detected tampering, or an empty `Set` if no tampering is detected.

App Protector SDK sample usage

RASPFacade initialization

Before using any method from RASPFacade instance, it needs to be created and initialized by using RASPFacadeFactory and its two methods:

- `create` and
- `createRASPFacade`.

```
private void initializeRaspFacade() {
    try {
        raspFacade = RASPFacadeFactory.create(getApplicationContext())
                                      .createRASPFacade();
    } catch (RASPFacadeException raspFacadeException) {
        // Handle expected error...
    }
}
```

Initializing `RASPFacade` can throw a `RASPFacadeExpection` if the `RASPFacade` object is created in the wrong way, e.g. if the wrong application context was passed to the `create` method.

detectOnDemand usage

After `RASPFacade` has been initialized, the `detectOnDemand` method can be called at any time. The method will return detected tampering, or an empty `Set` if no tampering is detected.

```
void detectOnDemand() {
    final Set<RASPDetection> detections =
raspFacade.detectOnDemand(EnumSet.of(RASPDetection.ROOT));
    // Handle detected tamperings...
}
```

RASPFacade usage

```
private void dummyRASPFacadeUsage() {
    try {
        initializeRaspFacade();
        detectOnDemand();

    } catch (final RASPFacadeException raspFacadeException) {
        // Handle expected error...
    }
}
```