# SplitDroid: Isolated Execution of Sensitive Components for Mobile Applications

Lin Yan, Yao Guo(✉), and Xiangqun Chen

Key Laboratory of High Confidence Software Technologies (Ministry of Education), Institute of Software, School of EECS, Peking University, Beijing, China
{yanlin10,yaoguo,cherry}@sei.pku.edu.cn

**Abstract.** Although many approaches have been proposed to protect mobile privacy through techniques such as isolated execution, existing mechanisms typically work at the app-level. As many apps themselves might contain vulnerability, it is desirable to split the execution of an app into normal components and sensitive components, such that the execution of sensitive components of an app can be isolated and their private data are protected from accesses by the normal components.

This paper proposes SplitDroid, an OS-level virtualization technique to support the split-execution of an app in order to isolate the execution of sensitive components and protect its private data. SplitDroid is enabled by porting the Linux Container to the Android environment and the ability to split Android apps through programming and runtime support. We also introduce a secure network channel to allow communication between the isolated component and normal Android apps, such that non-privacy-related information can be interchanged to ensure its correct execution. Finally, we demonstrate the feasibility and effectiveness of SplitDroid through a case study.

**Keywords:** Mobile security · Isolated execution · Privacy protection · OS-level virtualization

## 1 Introduction

As the development of mobile Internet and smartphones, more and more mobile applications (*apps* for short) have been developed to help people with their work, entertainment and daily life. Currently, both Google Play and App Store have over one million apps available for mobile users [2] to download.

As the number of mobile apps grows, people are storing more and more sensitive information on smartphones, such as passwords, credit card numbers, geo-locations, contacts information and even biometric information like fingerprints. Unfortunately, these sensitive data are vulnerable to various attacks from different malicious apps such as malware. For example, there are already a huge number of malware aiming at stealing user privacy on the Android OS [30].

As a result, many approaches have been proposed to protect sensitive data on smartphones based on various techniques, such as data encryption [9], data isolation [8,13,15,17–19,27] and isolated execution [14,16]. In this paper, we focus


B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 78–96, 2015.
DOI: 10.1007/978-3-319-28865-9_5

on approaches based on isolated execution because it is able to separate the execution of attackers and target apps into isolated environments, thus preventing sensitive data from being stolen from target apps.

Researchers have proposed several isolated execution approaches to protect mobile privacy. Solutions like L4Android [16] and Xen on ARM [14] support multiple virtual machines (VMs) containing Android OS running simultaneously on the same hardware. With bare-metal virtualization, these solutions provide strong isolation guarantee between VMs. However, they are too heavyweight to be used in smartphone environments considering the impact on memory usage, performance, and energy consumption. Other solutions aim at isolating confidential data at the app-level. For example, TrustDroid [6], MOSES [22] and AppCage [31] extend the Android framework to group apps into different domains and enforce access control among domains to protect user confidential data. These app-level solutions are lightweight in essence, but they assume that the middleware layer can be trusted, which is not always true in reality.

Recent approaches like Cells [4] and Airbag [25] achieve isolated execution by leveraging OS-level virtualization technologies. Both of them support multiple Android user spaces running simultaneously on the same Linux kernel. OS-level isolation provides strong albeit lightweight isolation guarantee between user spaces. However, these solutions typically protect each app as a whole, which might not be enough in many cases.

Based on our observation, privacy leakage often occurs within one app, where private data in one component may be leaked through another component in the same app. For example, many apps employ "social login" capabilities, which allow users to log in using popular third-party accounts such as Facebook or Weibo. This technique is similar to *single sign-on (SSO)*, and it allows users to log into different apps with one account such as Facebook. However, one of the potential vulnerabilities here is that, when a user logs into her Facebook account within a different app, the account and password information might be leaked through this app. In order to protect user information being leaked in these situations, we need an environment where sensitive components in an app can be executed in an isolated environment, for example, when a user enters her passwords. After successful login, the app will receive a "log in successful" message or an authentication token, but it cannot access the actual login credentials such as passwords.

In order to achieve this kind of fine-grained protection on sensitive data, this paper proposes SplitDroid, an OS-level virtualization technique to support the split-execution of a mobile app. We introduce the concept of separating privacy-related sensitive components from the rest of a mobile app and isolating the execution of them in a secure environment. By porting the Linux Container to Android, we build a trusted container with a separate Android runtime that runs sensitive components alone. We also provide developers the ability to split Android apps with dedicated programming and runtime support. We introduce a secure communication channel across containers for exchanging non-sensitive information with sensitive components from the normal Android environment.

As a result, our approach is lightweight since both containers share the same OS kernel, but it also provides strong protection with the Linux Container.

As a case study, we implement SplitDroid on a Nexus 5 smartphone and demonstrate its feasibility by enabling the split-execution of an app using social login. We also evaluate the performance overhead of SplitDroid to show its effectiveness.

This paper makes the following main contributions:

- We propose SplitDroid, a fine-grained privacy protection technique based on OS-level virtualization, which enables isolated execution of sensitive components in an app.
- We successfully port the Linux Container to Android and demonstrate that it is feasible to achieve execution isolation through OS-level virtualization.
- We introduce a mechanism to split the execution of an Android app through programming and runtime support. Based on the mechanism, we implement SplitDroid on Android and demonstrate its feasibility through a case study.

## 2 Preliminaries

In this section, we introduce some preliminaries to define the scope of our work. We start by presenting a running example to demonstrate the motivation for split execution of sensitive app components. Then we describe our design goals, assumptions and the adversary model.

### 2.1 A Running Example

Figure 1 depicts our motivation for fine-grained privacy protection by isolating the execution of sensitive app components. The execution overflow on the left side of Figure 1 shows a normal execution overflow of some mobile app $A$. App $A$ requires user login before accessing its services. This feature is reflected in the execution of the "login" component in the normal execution overflow, which means a user needs to provide her account and password to the remote authentication server via the login UI in app $A$.

As app $A$ runs in an open environment together with many other apps, malware may coexist with app $A$ and steal the account and password combination during the login procedure. Researchers have proposed many solutions to prevent this kind of privacy leakage. An app can be easily isolated into a stand-alone environment of many types, such as bare-metal VM, OS-level containers and app-level sandboxes.

However, isolating the mobile app as a whole may not be enough to prevent privacy leakage. As depicted in Figure 1, login credentials may be leaked through another component in app $A$ itself. For example, if app $A$ provides the login feature by integrating social login components from popular social networking services such as Facebook, user's Facebook password may be leaked through vulnerabilities of components in app $A$.

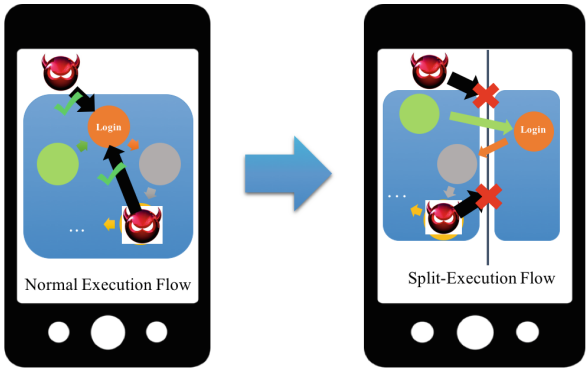


**Fig. 1.** A running example. *Left: an app containing a login component, in which the login credential might be stolen by adversaries; Right: the login component can be split and executed in an isolated environment, the original app has no access to login credentials.*

In order to prevent such kind of privacy leakage, we are motivated to build an isolated execution environment to confine the sensitive component that contains the private information. As depicted in the right side of Figure 1, software components with sensitive data (e.g., the "login" component in app $A$) can be split and executed in a fully-isolated environment. Threats on user privacy from both inside and outside app $A$ will be blocked by the isolation mechanism.

### 2.2 Goals

In order to provide fine-grained protection on user privacy, we propose SplitDroid, which provides isolated execution of sensitive app components. Our design of SplitDroid aims to meet the following goals:

- **Privacy confinement.** As more and more sensitive data congregate on mobile devices, our work aims to confine user privacy by isolating the execution of software components that are related to the collection and transformation of these sensitive data. Isolated execution can be achieved by leveraging virtualization technologies, which could support multiple execution environments running simultaneously and provide strong isolation among them.
- **Ease of programming.** We want to minimize developer efforts to utilize the proposed mechanism to enable the spilt-execution of an app. The underlying mechanism that facilitates the isolated execution should not be exposed to developers. In other words, developers do not have to know how to construct and manage the isolated execution environment. Building a new app based on the SplitDroid should be as simple as building native Android apps with Java.
- **User transparency.** App users should not notice the existence of the split-execution in SplitDroid. For example, when a user clicks the "login" button in

an app and then types in "username" and "password", a user should not feel the action or delay of switching environments, although the login credentials collection UI and account authentication process actually happens in another isolated execution environment.
- **Low performance overhead.** The influence of isolated execution on system performance must be low. In order to meet this requirement, the performance overhead of the isolated execution environment must be low compared to the normal execution environment and the switch overhead between these two environments must also be kept low. Since the isolated execution environment is based on virtualization technologies and different virtualization technologies bring different performance overheads and trusted computing base(TCB) sizes, we must find a proper trade-off between them.

### 2.3 Assumptions

We make the following assumptions in our work.

- We assume that the isolated execution environment created by SplitDroid is fully trusted as sensitive user data are confined in it. The whole software stack within the isolated execution environment including the OS kernel, middlewares and applications are all assumed to be trustworthy.
- We assume that the OS kernel inside the normal execution environment is also trusted since we choose OS-level isolation to implement the isolated execution. In fact, the OS kernel is shared between the normal execution environment and the isolated execution environment.
- We assume that there exists a trusted communication channel between the normal and isolated execution environments. The communication channel is mainly used to exchange non-sensitive information such as "login successful" notifications and tokens which do not contain user credentials.
- We assume that the external parties communicating securely with the isolated execution environment can be trusted, such as login authentication servers, mobile banking services, etc.

### 2.4 Adversary Model

SplitDroid aims to protect user privacy against the following adversaries. In order to steal user's sensitive data, attackers can compromise any part of the user space in the normal execution environment and even gain access to the interface with APIs we propose to support isolated execution. The attacker may also have access to the persistent storage in the normal execution environment. Attackers can be any app in the normal execution environment or even the rest components of the same app, while the execution of its sensitive components can be isolated leveraging SplitDroid. However, we do not consider side-channel attacks or physical attacks in this paper.

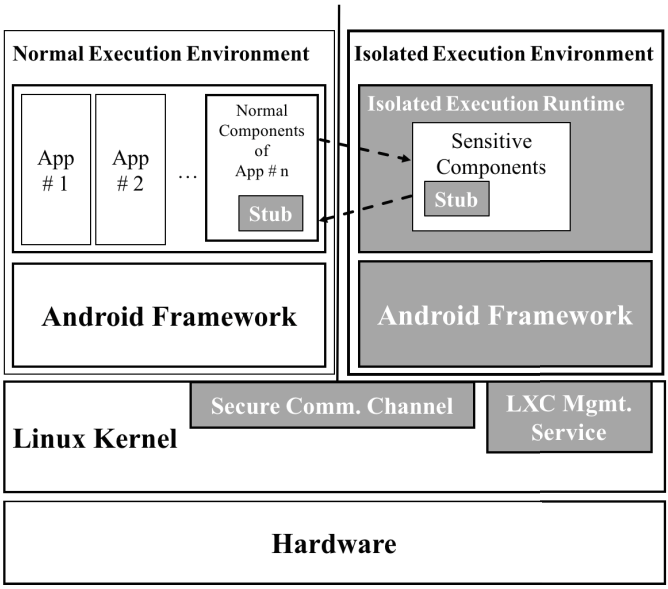


**Fig. 2.** Architecture of SplitDroid.

## 3 SplitDroid Design

In this section, we present the design of SplitDroid. We first give an overview of SplitDroid. Then we elaborate on its key functionalities and detailed design considerations.

### 3.1 Overview of SplitDroid

As depicted in Figure 2, SplitDroid separates the user space into two execution environments: a normal execution environment where an untrusted stack of software (middleware and most apps) runs, and an isolated execution environment created by SplitDroid where the trusted middleware and privacy-related app components run. By leveraging OS-level virtualization technology (Linux Container in our case), sensitive components running in the isolated execution environment is isolated from the untrusted code running in the normal execution environment. Besides, SplitDroid provides a secure communication channel between the two execution environments.

In order to leverage SplitDroid to protect privacy, a mobile app needs to be partitioned into two parts: one part consists of privacy-related component, and the other part consists of non-sensitive components. SplitDroid provides programming and runtime support for developers to develop and deploy the split-execution of mobile apps.

SplitDroid includes the following major components:

– **Isolated Execution Runtime.** First, the *isolated execution runtime* component in SplitDroid ensures the stand-alone execution of sensitive compo-

nents in the isolated execution environment. Its tasks include managing the execution lifecycle and providing isolated storage for sensitive components.

- **LXC Management Service.** The isolated execution environment is implemented as a container enabled by porting the Linux Container (LXC) to Android. The LXC management service is mainly responsible for the lifecycle management of the isolated execution environment, including the initialization, creation, suspension and termination of it. Most importantly, the LXC management service performs the environment switches when split-execution happens in SplitDroid.
- **App Stubs.** We introduce *app stubs* as the proxies of the sensitive components to the normal components of an app during split execution, and vice versa. During the split execution of an app, two stubs are running in the normal execution environment and the isolated execution environment, respectively. For example, if the "login" components in a mobile app are considered to be the sensitive part that are split and executed the isolated execution environment, the stub in the normal execution environment will serve as an agent of the "login" component, which offers exactly the same interface as "login" components.
- **Secure Communication Channel.** A secure communication channel between two execution environments is constructed through the shared OS kernel. The communication channel is responsible to fulfill synchronization between the two parts of the mobile app running in different execution environments through trusted stubs. End-to-end security of the trusted communication channel can be achieved through encrypted communication between the app stubs.

### 3.2 The Isolated Execution Environment

SplitDroid creates an isolated execution environment to run sensitive app components separately. We introduce the isolated execution environment by adopting OS-level virtualization. Compared to bare-metal virtualization, OS-level virtualization is lightweight since the OS kernel can be shared by VMs.

**Overview of the Linux Container.** SplitDroid adopts OS-level virtualization to create the isolated execution environment. In particular, we use a container-based lightweight virtualization framework in mainstream Linux kernel called Linux Container [3], which enables multiple isolated user-space instances running on a shared Linux kernel, thus offering OS-level virtualization. LXC relies on several Linux kernel features, in which *Namespaces* and *Control Groups* are the key enablers.

**Porting LXC to Android.** LXC was originally targeted at the X86 architecture for desktop Linux systems, so we need to port it to the ARM architecture in order to support Android. Since the Linux kernel used in Android has been optimized to support mobile environment, some kernel capabilities needed to

run LXC have been turned off and the standard GNU C library `libc` has been replaced by `bionic libc`. Thus, we first turn on those missing kernel capabilities and recompile the Android kernel. Missing kernel capabilities can be found by running the `lxc-checkconfig` script in the standard LXC-tools against the kernel config file of Android. After recompiling the Linux kernel, we cross compile and statically link all GNU `libc` independent libraries of LXC.

The most challenging part of constructing an isolated execution container is device virtualization which includes multiplexing the framebuffer and input devices.

In Android, all graphical contents shown on screen are updated by the screen updater to the *framebuffer* memory, which is mapped from kernel to user space. Since we have only one physical screen with two screen updaters from two VMs, we modify the framebuffer driver such that it will receive update requests from each container but will only allow the foreground VM to actually update the framebuffer. The other background container can still update its display data in a backend buffer, which will not be displayed until it switches to the foreground.

For input devices such as touch screen and physical buttons, we modify the device drivers only to respond to the requests from the foreground VM while input requests from the background container are discarded.

**Resulting Environment.** Based on LXC, SplitDroid creates a new container at system boot time. The new container is initialized with a clean copy of the same Android framework as the original Android running on the smartphone and relevant SplitDroid components. The new container is designed to serve as the isolated execution environment to confine the execution of sensitive components. LXC-tools are provided to enable on-demand switching between containers when the split execution of mobile apps starts. Two containers are configured to locate in the same virtual local network provided by LXC. SplitDroid also provides a secure communication channel between containers based on encrypted socket connection.

### 3.3 Split-Execution of Android Apps

SplitDroid introduces the concept of split execution of Android apps to protect user privacy. Specifically, an app can be split into sensitive components and normal components. By isolating the execution of sensitive components in a trusted environment, sensitive data can be protected from being leaked. Thus the goal of app split is to identify software components inside an app containing some specific sensitive data. In our current design, we provide app developer the ability to split the Android apps through programming support, which enables them to execute in a split manner with the provided runtime support. In the future, we plan to leverage static analysis techniques such as taint analysis to identify sensitive components automatically to help split Android app binaries.

**App Split.** Split execution of an Android app has been widely explored for various purposes such as computation offloading [7], where the computation-

intensive components of an app can be executed on a remote environment such as a server or a cloud.

An Android app can be viewed as a state machine in which the states represent Android *Activities* and state transitions represent *Activity* switches. Each *Activity* corresponds to a Java class that inherits from the `ApplicationContext` class. Besides, an *Activity* may be related to other Java classes and string/img resources depending on its actual business logic. Throughout the generation, process and storage of sensitive data (e.g. passwords), one specific item of sensitive data relates to at least one *Activity* (e.g. "Login" *Activity*).

In our design, we split an Android app on the granularity of *Activities*. Given the sensitive data to be protected, all *Activities* inside an app will be analyzed to check if they are related to the sensitive data. After the app is analyzed, all privacy-related *Activities* (including the related classes and resources) will be extracted from the original app and packaged as sensitive components. A stub will be inserted in to the rest of the app to work as the interface proxy of the extracted sensitive components.

**Runtime Support.** The *isolated execution runtime* component inside SplitDroid provides runtime support for the split execution of Android apps. The first task of the isolated execution runtime is managing the code of sensitive components. In our design, there is a one-time configuration step to install the code of sensitive components into the isolated execution environment in parallel with the installation of normal components of an app into the normal execution environment. As there may exist more than one app needing split execution, the isolated execution runtime should not confuse among different sensitive components from different apps. To keep track of sensitive components from different apps, the isolated execution runtime maintains an identity table to enforce a signature-based check before isolated execution.

SplitDroid also provides runtime support to manage the lifecycle of isolated execution. Once receiving a request to run sensitive components from the normal execution environment, the isolated execution runtime starts a new service process to run the code. When sensitive components finish executing, the isolated execution runtime will terminate the service process and issue a notification to the normal execution environment through a callback function. To ensure isolation, SplitDroid only supports sensitive components from one app to run in the isolate execution environment at a given time.

**Programming Support.** SplitDroid provides programming support for developers to enable the split execution of Android apps. In order to enable isolated execution of the sensitive components, a developer should go through the following procedure.

1. **Define the interface for accessing the isolated components from the normal execution environment.** The first step towards developing an app with SplitDroid is to specify which part of the app is privacy-related. After figuring out privacy-related components, the developer must specify

an interface to access these isolated components from the normal execution environment. In the normal execution environment, the interface to access isolated components should inherit from an `ITrustedStub` interface.

2. **Implement the actual business logic running in the isolated execution environment.** The developer should provide the actual code to be executed in the isolated execution environment. There must be a core class in the business logic code that inherits the `IsolatedExecution` class since the *trusted runtime service* component in SplitDroid will search the core class to start isolated execution. Meanwhile, the core class has to implement the interface defined in the first step, which wires the interface in the *trusted stub* to the actual business logic.
3. **Interact with isolated components.** After defining the interface and implementation of isolated components, the developer can write code to start to run these software components in the isolated execution environment and interact with them at runtime. The first thing to start running isolated components is to create an instance of the `IsolatedExeEnv` class and pass an instance of the core class of the isolated components as parameter to the `initialize` function of the instance. Then the developer can write code to interact with isolated components using pre-defined interfaces from the normal execution environment.

### 3.4 Usage Scenarios

To illustrate the applicability of SplitDroid, we present two real-world usage scenarios: social login and mobile payment. The *confidentiality* and the *integrity* of user privacy can be guaranteed by adopting SplitDroid in both cases.

**Social Login.** Social login within mobile apps is a form of single sign-on (SSO), which enables a user logging into a third-party app with existing login information from social networking services such as Facebook and Weibo. Social login is beneficial to all parties involved.

However, malware in an untrusted environment or even malicious app components inside a third-party app that integrates Facebook social login service could steal user's login credentials. Once the login credentials of a popular social networking account get stolen, all related third-parties are in danger. As described in Section 2.1, SplitDroid can be used to eliminate this kind of privacy leakage by isolating the social login components in an isolated execution environment.

**Mobile Payment.** With the development of e-commerce and e-banking, more and more e-commerce services are moving to the mobile platform. While building one's own mobile payment system is expensive and insecure, many e-commerce apps choose to integrate third-party payment plugins such as PayPal and Alipay.

For example, when users place orders in an e-commerce app that integrates PayPal services. After adding desired products into her shopping-cart, a user can choose to check out with PayPal. By logging into PayPal and choosing a

proper bank account, the user can successfully place her order. Convenient on one hand, this procedure contains potential risk of privacy leakage since PayPal login happens in an untrusted environment. The user's PayPal login credentials and related bank account information could be potentially leaked. With SplitDroid, we can prevent this from happening by isolating the software components of PayPal login and payment service.

## 4 Case Study

### 4.1 Goal

As a case study, we implement a prototype of SplitDroid to demonstrate its feasibility. Based on the prototype, we build a mobile app to show the effectiveness on privacy protection of SplitDroid. We also evaluate the performance of SplitDroid to show its practicality.

### 4.2 Implementation

Our implementation includes two parts: the prototype of SplitDroid and a mobile app based on the split-execution mechanism provided by SplitDroid.

**SplitDroid Prototype.** We implemented a prototype of SplitDroid based on CyanogenMod 11 (corresponds to Android 4.4) on a Nexus 5 smartphone. Then we port LXC 1.0 to Android and modify device drivers as we have described in Section 3.2. We use the same Android version for the runtime in both the normal execution environment (the host) and the isolate execution environment (the container created by LXC). We implement components in SplitDroid based on our design discussed in Section 3. We implement the programming support in SplitDroid by providing an SDK to app developers .

**Mobile App Implementation.** In a proof-of-concept implementation, we have implemented a simple app integrating social login services provided by Weibo. This usage scenario has been previously described in Section 2.1 and Section 3.4.

We implement the app by following the programming steps described in Section 3.3 . As shown in Code Example 1, the isolated sensitive components of Weibo social login function only has one interface, which is used to activate the `authorize` action.

**Code Example 1. Declaring the interface for an SSO service.**

```
public interface ISSOService extends ITrustedStub
{
  public void authorize(AuthListener authListener);
}
```

Code Example 2 presents the core class from the implementation of the actual business logic of Weibo social login service. The code example only shows some key steps to conduct the SSO authorization based on the Weibo SDK since the whole code base is relatively large and mostly irrelevant.

**Code Example 2. Implementing the SSO service.**

```
public class SSOService extends IsolatedExecution implements
    ISSOService
{
  public authorize(AuthListener authListener)
  {
    // Implements the authorize() function using the SSO SDK
    ......
    mAuthInfo = new AuthInfo(this, Constants.APP_KEY,
        Constants.REDIRECT_URL, Constants.SCOPE);
    mSsoHandler = new SsoHandler(WBAuthActivity.this,
        mAuthInfo);
    mSsoHandler.authorizeWeb(authListener);
    ......
  }
}
```

**Code Example 3. The `AuthListener` Class.**

```
class AuthListener implements WeiboAuthListener
{
  @Override
  public void onComplete(Bundle values)
  {
    // Parse login Token from Bundle
    mAccessToken = Oauth2AccessToken.parseAccessToken(values)
        ;
    if (mAccessToken.isSessionValid())
    {
      // Handle login information
      .........
    } else
    {
      // Handle error
      String code = values.getString("code", "");
      .........
    }
  }
}
```

**Code Example 4. Calling the SSO service in Main Class.**

```
IsolatedExeEnv isolatedExeEnv = IsolatedExeEnv.initialize(new
    SSOService);
ISSOService sSOService = (ISSOService) isolatedExeEnv.
    getTrustedStub();
sSOService.authorize(new AuthListener());
```

Code Example 4 shows how we interact with sensitive components running in the isolated execution environment. After initialization, the code running in the normal execution environment receives a reference of isolated components by calling the `getTrustedStub` function of the `IsolatedExeEnv` class instance. Then the `authorize` function of isolated Weibo social login components can be invoked directly through the definition of the `ISSOService` interface. Code Example 3 shows the implementation of a callback handler of SSO authorization, which is required as a parameter to call the Weibo social login Service.

### 4.3 Evaluation

To evaluate the effectiveness of our approach, we run the the mobile app we build in the SplitDroid environment. Figure 3 shows the screenshots of the whole split-execution process. The Weibo social login part of the app can be successfully isolated in a trusted environment.

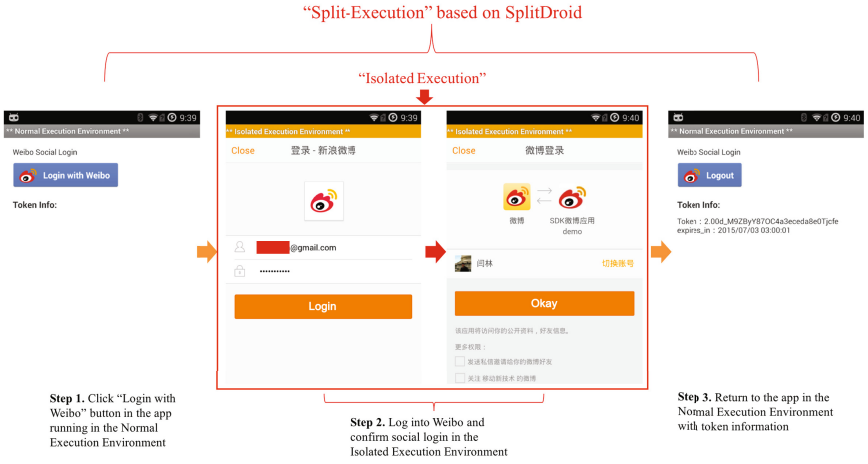


**Fig. 3.** Case Study: an Android app using the Weibo account login service.

To evaluate the performance impact of OS-level virtualization in SplitDroid, we first run the AnTuTu benchmark [1] with three different setups. "Baseline" means running AnTuTu in standard Android without SplitDroid. "NEE" means running AnTuTu in the normal execution environment of SplitDroid. "IEE" means running AnTuTu in the isolated execution environment created by SplitDroid. We can see from the results shown in Figure 4, the impact of the OS-level virtualization in SplitDroid on system performance is relatively low.

We then evaluated the performance of the mobile app running on SplitDroid by comparing the execution time of split-execution with the execution time of normal execution (normal app in standard Android environment) on the same Nexus 5 smartphone. The execution time is measured between clicking the "login

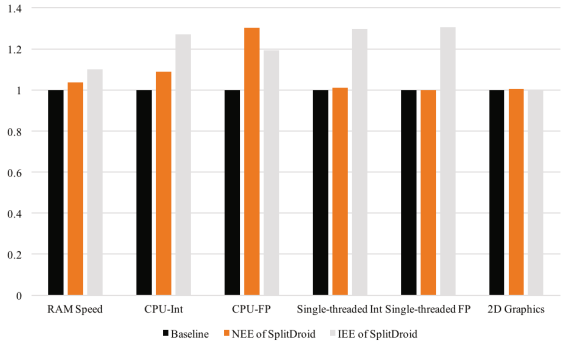


**Fig. 4.** Normalized performance impact of SplitDroid.

**Table 1.** Comparison of execution time (normal execution vs. split-execution).

| | Execution time (ms) | | |
|---|---|---|---|
| | Max. | Min. | Avg. |
| w/o SplitDroid | 75 | 29 | 45 |
| w/ SplitDroid | 223 | 179 | 190 |

with weibo" button, which is shown as the first step in Figure 3, and switching back to the UI containing token information, which is shown as the last step in Figure 3. The input time of user login credentials and network communication time is subtracted to reflect only the overhead brought by SplitDroid. Results are shown in Table 1. We can see that although the average execution time of SplitDroid is more than three times of the normal Android, the worst execution time is roughly one fifth of a second, which should be acceptable to most mobile users.

## 5 Discussions

### 5.1 Limitations

**Size of the Trusted Computing Base (TCB).** The TCB size of a privacy protection solution has long been a major concern in the research community. People believe that smaller TCB will narrow the attack surface of the trusted software components and ease the formal verification efforts. Compared to solutions based bare-metal virtualization, SplitDroid has a larger TCB size due to the inclusion of the OS kernel and a trusted container. However, LXC is more lightweight than bare-metal virtualization to fit in the mobile environment. So it is a trade-off we make between practical security protections and the TCB size.

**Lack of Support for Legacy Mobile Apps.** Although SplitDroid can be used to isolate privacy-related sensitive components inside a mobile app, it still

requires a source-code based approach, which means developers have to split the app manually in advance. However, support of legacy apps can be implemented with the help of program analysis in further studies.

**Heavy Deployment Process.** Our current implementation requires users to reinstall a new ROM on their smartphones in order to use SplitDroid. The reason is that the Android kernel has to be recompiled to support LXC by turning on several kernel capabilities. However, if SplitDroid is employed as a standard process in the future, the users do not to worry this any more.

### 5.2 Future Work

**Automated App Split.** The current way to support split-execution in SplitDroid relies on the efforts of developers to leverage our programming support. This would undermine the wide adoption of SplitDroid since it offers no support for existing mobile apps. One potential future direction is to find an automated way to split existing apps. This can be achieved by applying taint analysis on the data flow of sensitive data, thus identifying sensitive software components automatically.

**SplitDroid Based on Cloud Services.** Although SplitDroid can protect mobile privacy information from being leaked in a fine-grained level, sensitive data can still be retrieved by physical attacks of dumping memory or storage contents. As future work, we can investigate on building isolated execution environment with both container technology and cloud service such as storage service, which is similar to the idea of TinMan [26]. Sensitive data will never be leaked on the mobile devices by not appearing in the local environment. However, this requires a dedicated design on human computer interaction as the sensitive data can not be provided directly on mobile devices.

## 6 Related Work

### 6.1 Privacy Protection on Smartphones

Many approaches have been proposed to protect sensitive data on smartphones. Popular techniques include data encryption [9], data isolation [8,13,15,17–19,27] and isolated execution. Besides, TaintDroid [10] extends the Android platform to track the privacy data that flowing through third party applications. TaintDroid can be applied to monitor the system behaviors related to sensitive data on smartphones.

Our work focuses on isolated execution based privacy protection because it can separate the execution of attackers and target apps into isolated environments.

### 6.2 Isolated Execution Based Privacy Protection

**Bare-Metal-Level Virtual Machines.** Bare-metal virtualization provides a strong isolation guarantee to put different applications into separated VMs. Some efforts tried to improve the security of the Android platform by introducing platform virtualization [5,12,14,16]. However, platform virtualization is a heavyweight mechanism, which runs multiple software stacks in different virtual machines. It is usually neither necessary nor affordable in the current battery-powered mobile devices. Full platform virtualization requires the support of device virtualization to multiplex hardware to guest domains.

**Application-Level Sandboxes.** TrustDroid [6] introduces a lightweight isolation framework to protect apps in separate domains of different trust levels. TrustDroid can support the isolation between corporate applications and private applications. TrustDroid relies on the MAC mechanism to enforce the isolation policy of each domain. Meanwhile, TrustDroid also depends on the Android middleware to confine the inter-domain communication and data access. MOSES [22] also targets a similar usage scenario: company smartphones used by employees. MOSES introduces a policy-based framework to isolate apps with different security profiles on Android. AppCage [31] proposes two user-level sandboxes: dex sandbox and native sandbox to interpose and regulate an app's access to sensitive APIs. These app-level solutions all assume that the Android middleware is trusted, which is often not true in real cases.

**OS-Level Containers.** Cells [4] is a virtual mobile smartphone architecture by leveraging OS-level virtualization. It introduces a new device namespace mechanism and novel device proxies to multiplex a single set of phone hardware into multiple virtual phones (VPs). Airbag [25] adopts a similar container-based method to isolate suspicious apps. However, these work mainly focus on isolating the execution of a mobile app as a whole, which is not enough for real usage scenarios where software components from the same app can also steal sensitive data. SplitDroid adopts similar OS level virtualization technology while achieving more fine-grained privacy protection.

### 6.3 Split Execution of Mobile Apps

There are several attempts focusing on splitting the execution of some certain components inside mobile apps such as advertisement components [20,21,24]. These approaches mostly focus on isolating specific categories of untrusted components. However, our work aims to separate and protect trusted components with more strict isolation guarantee based on Linux Container.

Other previous work have investigated in split execution of applications such as Java or Android apps in order to provide features such as computation offloading [7]. These can be implemented by either splitting app binaries [7,11] or source code redevelopment [28,29]. TLR [23] proposes the split execution of .Net apps. Although the current design of SplitDroid requires developers to redesign the

app to support split execution, it can also be implemented using an automated approach based on program analysis to split Android binaries.

## 7 Conclusion

Although isolated execution has been studied in smartphone environments such as Android, they typically isolate the apps as a whole. Since an app cannot always be trusted to handled all private information such as credentials in third-party login services, we introduce the concept of splitting the execution of an app into normal components and sensitive components, such that the execution of sensitive components of an app can be isolated and their private data is can be protected from being accessed by the normal components.

We have presented SplitDroid, an OS-level virtualization technique that supports the split-execution of an app. SplitDroid creates an isolated execution environment enabled by porting the Linux Container to the Android environment. SplitDroid also provides programming and runtime support for developer to fulfill the split-execution of mobile apps. We have demonstrated the feasibility and effectiveness of SplitDroid by building a prototype of SplitDroid and evaluation through a case study.

**Acknowledgement.** This work is supported in part by the High-Tech Research and Development Program of China under Grant No. 2013AA01A605 and the National Natural Science Foundation of China under Grant No. 61421091, No. 91118004, No. 61103026.

## References

1. AnTuTu benchmark. http://www.antutu.com/
2. Google Play has more apps than apple now, but it's still behind in one key area. http://www.businessinsider.com/google-play-vs-apple-app-store-2015-2
3. Linux Container. http://lxc.sourceforge.net/
4. Andrus, J., Dall, C., Hof, A.V., Laadan, O., Nieh, J.: Cells: a virtual mobile smartphone architecture. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 173–187. ACM (2011)
5. Barr, K., Bungale, P., Deasy, S., Gyuris, V., Hung, P., Newell, C., Tuch, H., Zoppis, B.: The VMware mobile virtualization platform: is that a hypervisor in your pocket? ACM SIGOPS Operating Systems Review **44**(4), 124–135 (2010)
6. Bugiel, S., Davi, L., Dmitrienko, A., Heuser, S., Sadeghi, A.R., Shastry, B.: Practical and lightweight domain isolation on Android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 51–62. ACM (2011)
7. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: CloneCloud: elastic execution between mobile device and cloud. In: Proceedings of the Sixth Conference on Computer Systems, EuroSys 2011, pp. 301–314 (2011)

8. Dam, M., Le Guernic, G., Lundblad, A.: TreeDroid: a tree automaton based approach to enforcing data processing policies. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 894–905. ACM (2012)
9. Diesburg, S.M., Wang, A.I.A.: A survey of confidential data storage and deletion methods. ACM Computing Surveys (CSUR)**43**(1), 2 (2010)
10. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS)**32**(2), 5 (2014)
11. Gordon, M.S., Jamshidi, D.A., Mahlke, S.A., Mao, Z.M., Chen, X.: COMET: code offload by migrating execution transparently. In: OSDI, pp. 93–106 (2012)
12. Heiser, G., Leslie, B.: The OKL4 microvisor: convergence point of microkernels and hypervisors. In: Proceedings of the First ACM Asia-Pacific Workshop on Workshop on Systems, pp. 19–24. ACM (2010)
13. Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: These aren't the droids you're looking for: retrofitting Android to protect data from imperious applications. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 639–652. ACM (2011)
14. Hwang, J.Y., Suh, S.B., Heo, S.K., Park, C.J., Ryu, J.M., Park, S.Y., Kim, C.R.: Xen on ARM: system virtualization using Xen hypervisor for ARM-based secure mobile phones. In: 5th IEEE Consumer Communications and Networking Conference, CCNC 2008, pp. 257–261. IEEE (2008)
15. Kantola, D., Chin, E., He, W., Wagner, D.: Reducing attack surfaces for intra-application communication in Android. In: Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 69–80. ACM (2012)
16. Lange, M., Liebergeld, S., Lackorzynski, A., Warg, A., Peter, M.: L4Android: a generic operating system framework for secure smartphones. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 39–50. ACM (2011)
17. Li, X., Hu, H., Bai, G., Jia, Y., Liang, Z., Saxena, P.: DroidVault: a trusted data vault for Android devices. In: 2014 19th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 29–38. IEEE (2014)
18. Nauman, M., Khan, S., Zhang, X.: Apex: extending Android permission model and enforcement with user-defined runtime constraints. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, pp. 328–332. ACM (2010)
19. Ongtang, M., McLaughlin, S., Enck, W., McDaniel, P.: Semantically rich application-centric security in Android. Security and Communication Networks**5**(6), 658–673 (2012)
20. Pearce, P., Felt, A.P., Nunez, G., Wagner, D.: AdDroid: privilege separation for applications and advertisers in Android. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2012, Seoul, Korea (2012)
21. Roesner, F., Kohno, T.: Securing embedded user interfaces: Android and beyond. In: Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 2013), Washington, D.C., pp. 97–112 (2013)
22. Russello, G., Conti, M., Crispo, B., Fernandes, E.: MOSES: supporting operation modes on smartphones. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, pp. 3–12. ACM (2012)

23. Santos, N., Raj, H., Saroiu, S., Wolman, A.: Trusted language runtime (TLR): enabling trusted applications on smartphones. In: Proceedings of the 12th Workshop on Mobile Computing Systems and Applications, pp. 21–26. ACM (2011)
24. Shekhar, S., Dietz, M., Wallach, D.S.: AdSplit: separating smartphone advertising from applications. In: Presented as Part of the 21st USENIX Security Symposium (USENIX Security 2012), Bellevue, WA, pp. 553–567 (2012)
25. Wu, C., Zhou, Y., Patel, K., Liang, Z., Jiang, X.: Airbag: boosting smartphone resistance to malware infection. In: Proceedings of the Network and Distributed System Security Symposium (2014)
26. Xia, Y., Liu, Y., Tan, C., Ma, M., Guan, H., Zang, B., Chen, H.: TinMan: eliminating confidential mobile data exposure with security oriented offloading. In: Proceedings of the Tenth European Conference on Computer Systems, pp. 27. ACM (2015)
27. Xu, R., Saïdi, H., Anderson, R.: Aurasium: practical policy enforcement for Android applications. In: USENIX Security Symposium, pp. 539–552 (2012)
28. Yuan, P., Guo, Y., Chen, X.: Uniport: a uniform programming support framework for mobile cloud computing. In: The 3rd IEEE International Conference on Mobile Cloud Computing, Services and Engineering, MobileCloud 2015 (2015)
29. Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., Yang, S.: Refactoring Android java code for on-demand computation offloading. In: OOPSLA 2012, pp. 233–248. ACM (2012)
30. Zhou, Y., Jiang, X.: Dissecting Android malware: characterization and evolution. In: IEEE S&P, pp. 95–109. IEEE (2012)
31. Zhou, Y., Patel, K., Wu, L., Wang, Z., Jiang, X.: Hybrid user-level sandboxing of third-party Android apps. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS 2015, Singapore, Republic of Singapore, pp. 19–30 (2015)