Progressive Authentication on Android

Jeffrey Warren, Varun Ganesan, Vedha Sayyaparaju, Vikas Velagapudi

Demo link: http://www.youtube.com/watch?v=SVxdFII1IGw
Code: https://github.com/jtwarren/progressive_auth_android

Motivation

Imagine the last time that you shared your phone with a friend to show him a picture, video, or online article. After taking your phone, and while you turned your back, he decided to open your mail app and send an embarrassing email to your entire company. While this may seem like a contrived example, there are many similar situations that could be much more harmful or damaging. The "friend" could have opened your banking application, or snooped through your private and confidential messages. Motivated by the Microsoft Research paper "Progressive authentication: deciding when to authenticate on mobile phones"[1], we set out to solve this problem, by building a progressive authentication system for Android devices. We aimed to develop a system that allows users to maintain security without giving up the convenience of quickly opening their phone and accessing applications.

Design †

The current design of our progressive authentication system is focused on providing a secure environment that leverages existing android security techniques.[2] This ensures that malicious third party applications cannot gain control of the authentication flow. Our threat model is an attacker who has physical access to the phone and wants to access protected apps at a higher authentication level than the current one. Our design currently makes changes at the operating system level, as opposed to application level, and protects against different types of intents and actions sent to the OS.

This system is designed around multiple authentication levels that a user can assign to an app. For the purposes of this project, the two we implemented are passcode authentication and password authentication. The authentication methods are ordered in such a manner that to unlock a higher authentication level you must unlock all the lower levels. For our setup, level 0 was no authentication, level 1 was passcode authentication and level 2 was password authentication. After a user assigns an authentication level, our system intercepts calls to start that activity and enforces an authorization check. The activity will be authorized to launch only if

[1] O. Riva, C. Qin, K. Strauss, and D. Lymberopoulos, "Progressive authentication: deciding when to authenticate on mobile phones, in Proceedings of the 21st USENIX conference on Security symposium, Security'12, (Berkeley, CA, USA), pp. 15-15, USENIX Association, 2012.

[2] Enck, W., Ongtang, M., and McDaniel, P. "Understanding Android Security". IEEE Security and Privacy 7, 1 (2009).

the user has authenticated to the level specified for the app. We interpreted every screen off event as the end of a session and reset the user to an unauthenticated state.

At the core of our progressive authentication is a new system service (ProgressiveAuthenticationService) responsible for maintaining the authentication level of the current session and deciding if applications are authorized to launch. This service exposes a restricted interface that is protected by signature level permissions. Other system services and system level applications can use this interface to initiate authentication checks.

The next important piece of the progressive authentication system is a system level application used to authenticate the user. The main responsibilities of the application are to prompt the user for authentication and update the current authentication level, through the ProgressiveAuthenticationService, depending on the users input. Depending on the current authentication level the user has unlocked and the level required by the application to be launched, this application prompts the user with the next authentication steps.

In order to compromise our system, the attacker must gain system level permissions to read or write data. However, if the user were to gain such permissions on the device, then they have compromised Android. This is a case we do not cover as we are not attempting to improve the Android OS security model, but providing users with a better tool for securing more of their phone.

Implementation

ProgressiveAuthenticationService

The ProgressiveAuthenticationService is a system service, running at the system level. The service exposes three methods as part of its interface:

```
// Determine if the package is authorized to launch.
boolean authorized(packageName)

// Get the required authentication level of the package
int getRequiredLevel(packageName)

// Update the authentication level
void updateAuthenticationLevel(authLevel)
```

Most importantly, the service is protected by a signature level permission. This prevents any application not signed with the same key as the OS image from interacting with the service. This security is achieved by the use of `enforceCallingPermission(permission, message)` in each of the high-risk functions exposed by the interface. A security exception will be thrown if the signature permission is not enforced.

The authentication level (`authLevel`) is stored as part of the service. The `authLevel` starts at `0` and can be incremented only through the `updateAuthenticationLevel(authLevel)` method. Further, it is cleared (set back to `0`) when the screen turns off (and on) through the use of a broadcastReceiver which listens for the `ACTION_SCREEN_OFF` and `ACTION_SCREEN_ON` broadcasts. If, for any reason, the service is killed or restarted, the authLevel will be reset to `0`.

startActivity()

An authorization check is made when any application attempts to launch. This check is done in the `startActivityForResult` method in the base Activity class of the Android framework. The check is made by calling the `ProgressiveAuthenticationService` to check if the attempted application is authorized to launch. If the app is authorized to launch, it will be launched as normal. If the application is not authorized to launch yet, it will be sent to the authentication application and launched when the appropriate authentication is completed.

To handle implicit intents, we used the existing `resolveActivity` function to prompt the user to pick an application and then extract the package name. We make this call from within `startActivityForResult` so we have access to the chosen package name and can perform the appropriate checks. When a default application has been chosen for an implicit intent, `resolveActivity` returns the information of that default application, so we have access to the necessary package info in either case.

ProgressiveAuthenticationApplication

The `ProgressiveAuthenticationApplication` includes the logic for deciding which authentication method to run and when to start the launching application. The launching application is passed in and its required authentication level is checked. The appropriate authentication methods are then launched to prompt the user to authenticate up to the required level. Once this is done, the launching application is launched. If anything fails, the launching application is not launched and focus will be given back to the previous application, usually the launcher.

Two authentication methods were built as proof of concept of the progressive authentication system -- passcode and password. The passcode authentication method simply checks the user input against a stored pin. This is meant to simulate a low authentication method that focuses on convenience rather than security. The password method provides a more robust, fully salted and hashed, authentication method. This is meant to simulate a higher authentication method that puts security over convenience. It was a conscious decision to make these authentication methods active, requiring user input. We did not want to get distracted from building a secure system by the machine learning challenges that accompany many forms of passive authentication. The authentication methods are integrated into the system application in private activities preventing outside processes from spoofing the authentication.

We also implemented a basic wifi authentication service but did not integrate it into the app because we felt it did not fit into the current authentication structure. The wifi authentication service would allow users to access apps only when connected to certain user defined wifi networks. Since higher authentication levels rely user having passed previous authentication methods and the wifi authentication can only be satisfied in certain limited contexts, it makes it difficult to insert the wifi method in our authentication flow. Wifi authentication would restrict all higher authentication levels to be satisfied only in the context of the correct wifi networks. In the future, rather than mapping the authentication methods to an absolute authentication level, we would allow them to contribute to the service's confidence score of whether the current user is the owner. This change would result in a tradeoff between a clear understanding of each security level and a more versatile form of progressive authentication.

Conclusion

Android is a secure mobile operating system but application and data security relies on the user being careful with their devices. Progressive authentication gives users the flexibility of quick access to commonly used apps while still securing high-risk applications. Further, it allows users to separate authentication based on the importance of the application. Now, a user can share his or her phone to let a friend look something up on the internet without worrying about data within the email or banking applications. Use of android security mechanisms ensures that the progressive authentication system itself is secure and not vulnerable to side loaded apps and intents sent to the device via adb by an attacker.

Future Work

As noted in the design section, our current authentication methods supported by our system rely primarily on active checks of the user's identity. Another way to increase convenience for the owner is to add authentication schemes that learn the characteristics of the device owner's usage patterns and monitors this data to alter the authentication level passively. Adapting the system to support passive authentication methods should be minimally invasive. It would primarily involve updating the ProgressiveAuthenticationService. However, we may want to consider redesigning our system completely based on how much we want to emphasize passive authentication methods.

In addition to adding more authentication methods, we would like to support more user control in customizing their progressive authentication experience. This includes allowing multiple forms of authentication to reach the same auth level, recommending authentication levels for apps based on permissions, and choosing the order of the authentication methods. When we consider allowing more user control, we will have to weigh the user's knowledge of Android security. Even the best authentication scheme is bottlenecked by user negligence.

†Discussion of Design Evolution

Coming into this project with minimal Android experience, our team was required to learn the Android operating system. As such, our understanding of the operating system matured as we developed solutions. Our original approach to solving this problem was to develop a custom launcher that would do everything from maintaining state, to checking and prompting for authentication, to actually launching the applications. It was a exceptionally high level solution that did not handle critical cases such as implicit intents, and proved to be too easy to bypass.

We decided that we needed to modify the android source code after taking a step back and looking more closely how intents were resolved. After many experiments with system services, system applications, and permissions we were able to build a solution that we could reason is secure -- through the use of system and signature permissions and other security mechanisms.