Practical Out-of-Band Authentication for Mobile Applications

Kapil Singh
IBM T.J. Watson Research Center
Yorktown Heights, NY
kapil@us.ibm.com

Larry Koved
IBM T.J. Watson Research Center
Yorktown Heights, NY
koved@us.ibm.com

ABSTRACT

Mobile devices create new opportunities and challenges for authentication. On one hand, the readily-available sensors provide new opportunities for authentication credentials, such as biometrics and context of the device. On the other hand, mobile applications rely on network services to create rich functionality that often require protection of their sensitive data. The ability for the mobile application developer to adopt a wide range of authentication protocols and techniques is an intractable challenge for adopting new authentication technologies.

In this paper, we propose a flexible framework that enables an out-of-band authentication channel for mobile applications. The framework allows applications to delegate authentication to an independent security service on the client that, in turn, supports an extensible range of authentication protocols. Importantly, the approach presented in this paper does not require any modification of the underlying system, thus not requiring support from the operating system or hardware vendor. Our server-driven approach supports administration and enablement of new authentication techniques and security policies with minimal to no client application modifications. We show the viability of our design by means of a framework prototype and integrating it with a representative authentication system built in-house. We also discuss security and non-security challenges of realizing this approach.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*

General Terms

Security, Design

Keywords

Authentication and Authorization, Mobile Application Security, Out-of-band framework

Middleware 2013 Industry Track, December 11–13, Beijing, China

1. INTRODUCTION

Mobile client applications (or *apps*) request data and services from network-based or server-based systems (hereafter referred to as *network services*). Often these network services contain sensitive data or provide sensitive services requiring the user of the mobile device to provide security credentials to the network services. All applications that expect to receive authentication challenges from the network services need to implement all of the possible authentication challenge-response protocols. The simplest, and most prevalent authentication challenge-response, is that the user is prompted for userid / password entry.

The ability for a mobile application to adopt a wide range of authentication challenge-responses is intractable, especially for the adoption of emerging technologies in mobile contexts, such as biometric or geolocation authentication (e.g., geofencing), where the authentication protocols are not yet standardized. The problem for the application developers is that many of the authentication challenges are not known when the mobile application is developed. More importantly, it is very common in enterprise environments for a security proxy technology to sit between the mobile device and the network service (see Figure 1). This proxy provides enterprise-wide authentication and authorization, and may also provide single sign-on services. As enterprise authentication requirements evolve over time, the challenges the proxy generates are likely not to have been implemented in the already deployed mobile applications.

Application development on modern mobile platforms is also dominated by third parties that often do not have any formal relationship with the network services. These applications typically require users' credentials to enable them to consume the interfaces and functionality exposed by the network services. For example, FriendCaster [2] is a popular third-party mobile application that consumes Facebook data by communicating with Facebook servers using users' credentials. Unfortunately, sharing credentials with third-party applications creates a security risk given the untrusted landscape of mobile applications [17].

To address the aforementioned challenges, we believe that application development should be kept independent of the authentication protocols. This consequently motivates the development of a pluggable framework to which applications can readily delegate the handling of the challenge-response protocols for authentication. In such a design, users' credentials would only be provided to this *trusted* framework that,

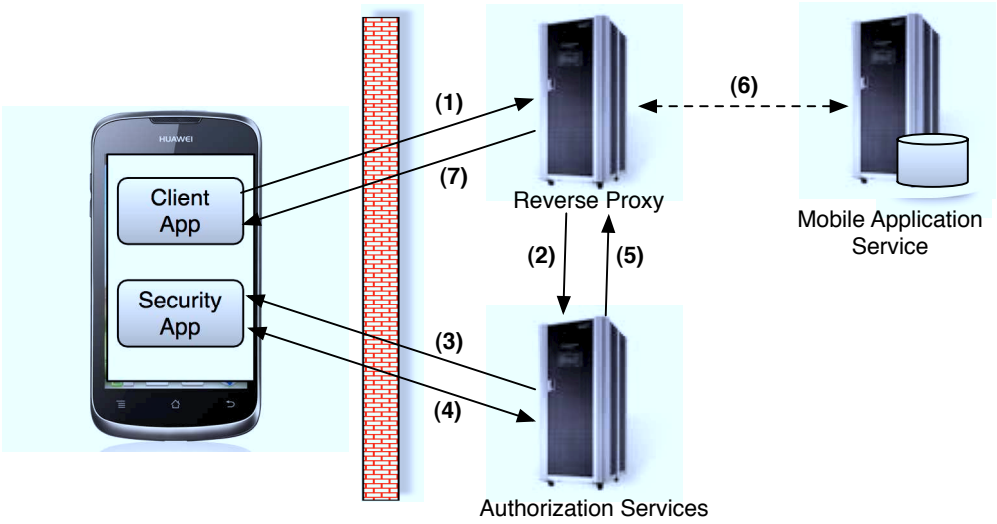

Figure 1: High-level architecture of our framework with the corresponding workflows as follows: (1) The Client App makes a service request to the Application Server, (2) the Reverse Proxy intercepts the request, determines if additional authorization is needed and calls the Authorization Services, (3) the Authorization Service invokes the Security App by sending authentication challenges to the mobile client, (4) the challenge-response protocol ensues between the Security App and the Authorization Services, (5) the Authorization Service communicates its decision to the Reverse Proxy, (6) if the request is authorized, it falls through to the Application Server for further processing, (7) the response is sent back to the Client App.

in turn, can proxy authentication services for any mobile application.

In this paper, we develop a novel architecture of such a framework and a high-level protocol (together named as *Out of Band Authentication Channel*, or *OOBAC*) that enables the mobile application to remain unaware of the network-based authentication services. We separate the authentication challenge-response handling from the mobile application and offer this security functionality as a separate mobile application or service on the mobile device. The authentication challenge-response protocol between the network service and the mobile device is handled by this specialized security application or service. As new network services authentication-challenge techniques and protocols are implemented and deployed, only the mobile security application needs to be updated to handle the new protocol, not the multitude of client applications using network services that are being mediated by the reverse proxy server. This provides significant improvement over the current state of the art where the application developer needs to include handling of all possible authentication challenge-response protocols. Additionally, separating the authentication channel from the mobile application prevents a malicious or vulnerable application from leaking users' authentication credentials. As we will describe later, the OOBAC application can be given a greater number of privileges, such as geolocation, network access, and access to other sensors that should not necessarily be afforded to the mobile applications [10, 15]. Thus, by completely separating the authentication (OOBAC) from the mobile applications, we are better able to enforce the Principal of Least Privilege [14].

Our protocol employs an *out-of-band* channel between the authentication and authorization services and the security application. The protocol is initiated by the network-based authentication and authorization service that makes a push notification call to the security service on the mobile client. This notification to the mobile client initiates the communication between the two participating parties. Once authentication processing over this secondary channel is complete, communication between the mobile application and the network service resumes in accordance with the successful or unsuccessful authorization of the request.

By leveraging an an out-of-band channel for authentication, our framework enables dynamic runtime authentication decisions based on the context of the request. One such example is to utilize the user's location to determine what authentication credentials are requested from the user for a particular transaction. For example, a user can be requested to provide multiple biometric credentials at a public location while no credentials are needed at work location. Note that our framework only enables specification and enforcement of such user- (or organization-) specific policies without making any judgement on what such policies should be.

2. DESIGN

In modern mobile platforms, application development has become much more distributed with a growing number of applications being developed by third parties. These third-party application developers often do not have a formal relationship with the network services. The application typically consumes the interfaces exposed by these network services. For example, FriendCaster [2] is a popular third-party mobile application that consumes Facebook data by communicating with Facebook servers using the user's credentials. This scenario presents several concerns. First, the third-party application, FriendCaster, must be trusted with the user's Facebook credentials. This can be quite risky given the untrusted landscape of mobile applications. Second, any

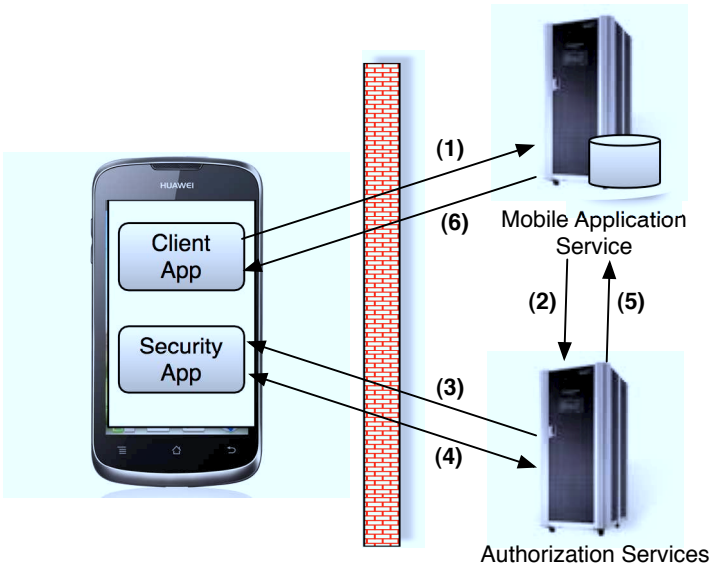

Figure 2: Proxy-less architecture with the corresponding workflows as follows: (1) The Client App makes a service request to the Mobile Application Service, (2) the Mobile Application Service determines if additional authorization is needed and calls the Authorization Services, (3) the Authorization Service invokes the Security App by sending authentication challenges to the mobile client, (4) the challenge-response protocol resumes between the Security App and the Authorization Services, (5) the Authorization Service communicates its decision to the Mobile Application Service, (6) if the request is authorized, the Mobile Application Service continues for further processing and the response is sent back to the Client App.

modifications or enhancements to the Facebook's authentication mechanism, such as inclusion of biometric credentials, requires modifications to all applications that use the Facebook APIs.

To address these limitations, flexibility and security are the primary goals of our framework design. Our design is driven by the network services. The network services define the authentication requirements for any request for a resource. Such requirements are imposed in the form of authentication challenges sent to the client by the network services. For example, let us assume that Facebook were to require a new form of knowledge-based authentication when the service deems that there is higher risk, such as authenticating from more then one location at the same time. If authentication were to be embedded into every application calling Facebook APIs, all such applications would need to update their code to handle this new API or face the wrath of unhappy users who are no longer able to log into their Facebook accounts from these applications.

Figure 1 depicts a representative high-level architecture of our framework when configured with a reverse proxy service that enforces authentication and authorization. When an application makes a network services request, the reverse proxy intercepts the request and invokes the authentication and authorization service, which, in turn, sends an authentication challenge to the user (Step 2). Instead of sending the challenge over the primary communication channel, we create a secondary out-of-band channel of communication between the authorization service and a specialized security application on the client. The security application stays dormant on the client device and is only triggered by a security token sent over the push notification channel to the device (Step 3). The security application on the mobile client uses this token as a unique identifier when performing the appropriate authentication services as requested by the service. All authentication services are subsequently performed over this secondary channel (Step 4).

The primary communication channel is being blocked by the reverse proxy while the authentication is performed over this secondary channel. When the authentication protocol completes on the secondary channel, the primary communication channel resumes the communication. If authentication fails and authorization is denied, then the primary communication reports the authorization failure. If authorization is successful, the request initiated by the mobile application is forwarded to the network service for processing as usual. Aside from the reporting of an authentication failure (access denied), the client application is unaware of the authentication being performed on the secondary channel.

It should be noted that while Figure 1 shows the communication in steps (3) and (4) as bypassing the Reverse Proxy, in practice the actual communication may flow through the Reverse Proxy, but still remain on a secondary communication channel since the primary communication channel remains blocked by the Reverse Proxy.

While the use of a Reverse Proxy is a typical deployment pattern for supporting common authentication / authorization services for multiple applications, there are many mobile application services that directly perform the authentication and authorization. Instead of the Reverse Proxy initiating the challenge-response protocol, the Mobile Application Service invokes the Authentication Services in the same way that the Reverse Proxy invoked these services (Figure 2).

Finally, we note that it is sometimes desirable to communicate the user identity from the Client App to the Security App. From a usability perspective, if the user enters their

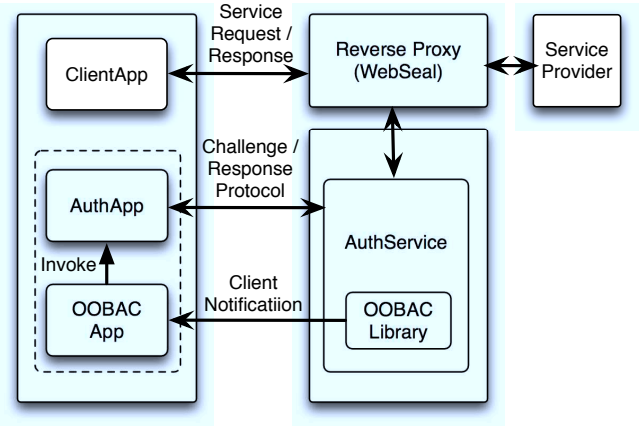

Figure 3: Prototype implementation of OOBAC.

identity into the Client App then it becomes frustrating and annoying to reenter the same information as part of the security credentials. The communication of the user identity, as well as any other relevant application state, can be passed to the Security App via existing interprocess communication mechanisms. Aside from this usability enhancement of passing along the userid to the Security App, there is no other communication between the mobile applications and the Security App, thus minimizing the integration and deployment.

3. IMPLEMENTATION

One goal of our implementation is to enable easy integration of our OOBAC framework with a variety of authentication and authorization systems and services. The implication of this goal is that multiple authentication and authorization protocols can be supported by the OOBAC framework. Thus, we implemented our framework as an entity independent of the authentication and authorization system, allowing us to seamlessly integrate with any such system.

For our prototype implementation, we developed our own authentication and authorization system in-house, with the client side application (denoted by *AuthApp*) capable of collecting both biometric credentials (e.g., face and voice) and non-biometric credentials (e.g., geolocation context) on demand. The server side (denoted as *AuthServer*) is responsible for verifying such credentials before providing appropriate access. *AuthApp* and *AuthServer* communicate with each other using REST-based API calls [12]. It is feasible to offer the server-side OOBAC library functionality as a service, thus providing further separation of the authentication and authorization services from the OOBAC services.

In our OOBAC implementation, the client side of the framework is developed as an application independent of *AuthApp* (Figure 3). We use Android as our mobile platform of choice for the prototype implementation, however, the client side of the framework can be similarly developed for other platforms. The server side is developed as a library that abstracts all communication with the client side and exposes a set of simple APIs that can be used by the *AuthServer* to invoke *AuthApp* on demand and pass appropriate invocation parameters (such as session identifier). We use Google Cloud Messaging (GCM) service [3] to serve notification triggers from OOBAC's server library to the Android-based OOBAC client.

On the mobile device, the communication from the OOBAC client to the *AuthApp* is in the form of an invocation trigger. In Android, one application can invoke another application by starting an appropriate *activity* of the target application [1]. The Android platform exposes a set of APIs for such invocation that takes the target application's package and activity name as input parameters. These parameters are passed by the *AuthServer* using the API calls to the OOBAC server and then communicated from the OOBAC server to its client.

In essence, our OOBAC framework acts as a bridge between the target application (denoted by *ClientApp*) and the *AuthApp* providing the authentication/authorization functionality. When a Reverse Proxy (or Mobile Application Service in a proxy-less design) determines that additional credentials are needed to allow a transaction, a request is made to *AuthServer* (or proxy-less equivalent) that, in turn, triggers the OOBAC App via the server-side OOBAC library and push notification service.

4. RELATED WORK

There have been many proposed solutions for providing application independent authentication services. We describe a few here, including their strengths and weaknesses.

Web browsers support a status code from the server indicating that authentication credentials need to be supplied by the client [7]. By separating out the authentication challenges from the application, the challenge-response logic and corresponding user interface code does not need to be created by every web application developer. In practice, however, there are very few challenge response protocols supported by all of the popular web browsers.

To address the weakness of current browser limitations, as well as to address phishing attacks, desktop browser extensions were created to manage security credentials (e.g., [9, 11]). These technologies allow for the management of multiple security credentials, and allow for supporting multiple challenge-response protocols. These technologies were developed in the context of web authentication. We have not seen a similar technology integrated into mobile devices, or that support non-HTTP requests.

For desktop computing, a number of vendors have created single sign-on solutions that manage a set of security credentials for the user so that they can connect to multiple services, each with different credential requirements (e.g., [4]). The user authenticates with a single set of authentication credentials, which then allows the single sign-on technology to access and pass along the credentials required for the other services. We have not seen a similar technology integrated into mobile devices, and recognize that it would be difficult to integrate this kind of technology into unmodified mobile browsers (e.g. without "jail breaking" the mobile devices).

A mobile device password management framework, GuarDroid [16], provides a secure storage mechanism for userids and passwords on mobile devices. It functions by operating separately from the apps, acting as a proxy, and automatically fills in the users' credentials (userid and password). While this may work for traditional credential management, it does not allow for extensible authentication, and requires modification of the operating system. Additionally, if the

HTTP session were to be encrypted (e.g., SSL), then the usual problems that arise with key management and trust also arise since there is a man-in-the-middle.

There are many forms of two-factor authentication. The typical pattern is for the authentication service to generate a security challenge to which the user needs to respond. In the case of 2-factor authentication tokens (e.g., [6]), the authentication service generates a random value that is sent to the security token and the user then enters this value as the second factor in the response to the authentication challenge. The expense and inconvenience of carrying a separate security token is usually an impediment to its use for mobile applications. A similar 2-factor approach was created to allow for email to be used as the communication channel for SAML-based user authentication. Unfortunately these 2-factor approaches lack the flexibility needed for new challenge-response protocols, such as biometric authentication, that are well integrated into mobile devices in a way that affords ease of use.

An alternative approach would be to create an authentication API that could be used by all applications and configurable on-the-fly. One such approach is the Java Authentication and Authorization Services [5, 13] that is part of the standard Java runtime libraries. The strength of this approach is that the authentication services can be configured after application development and deployment. The practical problem is that there are many thousands of applications already deployed that do not already use such a framework and would not be easily adaptable to the use of new authentication challenge-response protocols without application modification since the deployed mobile applications would need to include the updated authentication challenge-response protocols. As a practical matter, retesting and rereleasing an mobile application can be both time consuming and expensive for development organizations both large and small.

5. DISCUSSION

There are benefits of keeping the authentication and authorization logic independent of the applications. In addition to the simplified application development, it allows same security application to provide authentication and authorization services to multiple client applications. Since the security application does not share any authentication or authorization credentials with the client application, our design also enables the protection of such credentials from malicious applications (such as the fake Netflix application that was found to be stealing Netflix credentials [8]). On the network services side, the proxy acts as an effective control point where certain mandatory policies, such as corporate security policies, can be enforced. Moreover, the security application can provide additional context for the device (e.g. device identifier, its location, etc.) as part of the challenge-response protocol that, in turn, enables the authentication and authorization service to make rich, context-aware decisions.

Similar to AdSplit [15], we are partitioning functionality. In AdSplit, they partitioned the application from the advertising library to achieve the principal of least privilege. This limits the destructive capability of the application and/or ad library. By separating the OOBAC from the application, each can minimize the number of privileges that each requires, thus better achieving the principal of least privilege.

While this paper has focused the discussion on leveraging the technology in the context of a corporate environment with the use of a reverse proxy or calls from a mobile application server, it is possible to deploy this technology as a service. In our earlier social networking example (Section 2), it is possible for the social networking service provider to make their Security App available for download, similar to how QRCode and other services are made available in today's app stores. Third party application providers can then rely on the Security App being installed on the mobile devices and ready to be called by the service provider.

The limitations of our approach are associated with the deployment of one or more separate mobile applications to perform the authentication protocols. If shared across multiple network services, these applications need to protect against spoofing and phishing attacks. These applications also need to have mechanisms and protocols to allow for updates to the challenge-response protocols that they support. If the Security App is not shared across multiple network services, then there may be multiple Security Apps concurrently running on the device that need to be managed. Additionally, in some mobile systems, the Security App may need to interrupt the running Client App, and take over the user interface (screen), to perform authentication. This interruption may result in a suboptimal user experience.

6. CONCLUSIONS

For mobile applications, we are unaware of an existing generalized solution, without modifying the platform or operating system, which will allow us to easily adapt existing applications and services to support new authentication techniques such as biometrics and location. While there are many authentication solutions for the traditional desktop environment, we believe that there is a need for a solution that provides a simple, flexible and configurable authentication framework that supports existing mobile applications and services with no application code changes, and that work without "jail breaking" the device.

We also believe that simplicity of mechanism is very important for security, reliability, and usability. The approach described in this paper can be used with reverse proxies or integrated with networked services in a manner that is consistent with existing authentication and authorization services. Existing service providers can offer the Security App as a downloadable application that can be called by the service provider as necessary without any need for interaction with third-party application providers.

The Security App described in this paper was developed in the context of a usable multi-factor authentication project that supports the use of biometric and non-biometric factors in the authentication process. The proposed solution is simple, flexible, and supports existing and new mobile applications. It also keeps security credentials separate from the mobile application. It is easy to update the mobile security service to support new authentication protocols as they are created. We believe that this approach will be successful in supporting a wide range of authentication techniques for a broad range of mobile applications with no, or minimal, modification.

7. ACKNOWLEGEMENTS

This work is supported by a grant from the United States

Department of Homeland Security under contract FA8750-12-C-0265.

8. REFERENCES

[1] http://developer.android.com/reference/android/app/Activity.html.

[2] FriendCaster for Facebook. http://friendcasterapp.com/.

[3] Google Cloud Messaging for Android. http://developer.android.com/google/gcm/index.html.

[4] IBM Security Access Manager for Enterprise Single Sign-On. http://www-01.ibm.com/software/tivoli/products/access-mgr-esso/.

[5] Java^{TM} Authentication and Authorization Service (JAAS) Reference Guide. http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html.

[6] RSA SecurID. http://www.emc.com/security/rsa-securid.htm.

[7] W3C RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, section 10 Status Codes Definitions. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

[8] Will Your Next TV Manual Ask You to Run a Scan Instead of Adjusting the Antenna? http://www.symantec.com/connect/blogs/will-your-next-tv-manual-ask-you-run-scan-instead-adjusting-antenna.

[9] Higgins Personal Data Service, 2009. http://www.eclipse.org/higgins/.

[10] T. Book, A. Pridgen, and D. S. Wallach. Longitudinal Analysis of Android Ad Library Permissions. In *Mobile Security Technologies (MoST)*, San Francisco, CA, May 2013.

[11] D. Chappell. Introducing Windows CardSpace, Apr. 2006. http://msdn.microsoft.com/en-us/library/aa480189.aspx.

[12] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[13] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers. User Authentication and Authorization in the Java^{TM} Platform. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC)*, Phoenix, AZ, Dec. 1999.

[14] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer System. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[15] S. Shekhar, M. Dietz, and D. S. Wallach. AdSplit: Separating Smartphone Advertising from Applications. In *Proceedings of the 21st USENIX Security Symposium*, Bellevue, WA, Aug. 2012.

[16] T. Tong and D. Evans. GuarDroid: A Trusted Path for Password Entry. In *Moble Security Technologies (MoST)*, San Francisco, CA, May 2013.

[17] Y. Zhou and X. Jiang. Dissecting Android Malware: Characterization and Evolution. In *IEEE Symposium on Security and Privacy*, San Fransisco, CA, May 2012.