Editors: Brian Chess, chess@hp.com
Brad Arkin, barkin@adobe.com

The Case for Mobile Two-Factor Authentication

Dimitri DeFigueiredo
Adobe

User authentication is a core building block of any secure collaborative computing system. And, because of the enhanced interaction between mobile applications and Web services, mobile device user authentication is even more frequent than desktop user authentication. However, the current password-based approach to authentication used on the Web and desktop isn't well suited to the mobile setting. For example, entering a password on a mobile phone while walking is much more cumbersome than entering it while sitting comfortably in front of a desktop.

Here, I examine this problem in more detail and show how *two-factor authentication* can provide added security for mobile devices. For example, attackers won't be able to delete all your pictures on photoshop.com simply by stealing your phone. They also won't be able to access your blog and delete all entries simply by stealing your password.

Mobile vs. Desktop

To understand how to build secure mobile authentication systems, we need to first explore the nuanced differences between mobile and desktop devices. The transition from desktop or laptop to mobile devices introduces new usage patterns and usability constraints. These new ways users interact with mobile devices highlight security problems that were never fully solved for the desktop:

- *Device loss.* It's much more common to lose mobile phones than desktops.
- *Phishing.* It's much easier to trick users into giving up passwords when they use mobile phones than when they use desktops.

Software development has left the risk of laptop loss largely as a problem for the OS or third-party products to handle. Users and companies concerned about data on their laptops use disk encryption to protect it in case a laptop is lost. Disk encryption can save users from the lost mobile phone problem too. It helps protect all the passwords for a laptop or mobile device.

However, disk encryption can't protect against phishing. In the mobile world, there's no browser "chrome." The Web in a mobile device is an immersive experience that looks just like the device's own applications. Mobile users don't see the difference between HTTP and HTTPS URLs, unless they beg for it. Much more interaction occurs between Web services and mobile devices than with desktop clients. Many mobile applications ask users to authenticate themselves to a Web service. Many applications (such as Google Maps) work only if the user is online. The user transitions between authenticating online and on the device much more frequently and doesn't know or care about the difference between the two.[1] Under these conditions, attackers can much more easily trick users into typing passwords into the wrong webpage or application. So, if you're relying only on passwords to provide authentication for your website, you might face much more password theft when going mobile.

Two other mobile phone characteristics actually help security:

- Mobile phones are personal devices.
- A user quickly notices a lost or stolen phone.

The first point is extremely useful for authentication. A mobile phone is usually used by either an individual or a mutually trusting group. Unlike the desktops in Internet cafés, there are no mutually distrusting users of the same phone. Although losing a mobile phone is more common than losing a laptop, users notice the loss more quickly. This enhances the usefulness of popular (and creepy) remote-wipe capabilities.

Now that I've shown this isn't merely a porting task from desktop to mobile, let's examine the basics and the problems we need to solve.

Authentication Is Tricky

Whenever two parties communicate remotely, the possibility ex-

 1540-7993/11/$26.00 © 2011 IEEE COPUBLISHED BY THE IEEE COMPUTER AND RELIABILITY SOCIETIES

ists that one party isn't who they claim to be. Authentication involves ascertaining the identity of the person at the other end of a communication channel to be sure about whom you're talking to. At least two parties are always involved, and they take on the roles of prover and verifier. The prover wants to prove his or her identity; the verifier verifies that claim.

For example, when Mike is accessing his bank account at http://my-example-bank.com, he needs to be sure he's really talking to his bank and not somebody else. Similarly, the bank needs to know it's Mike, not an impostor, who's trying to access the account. In this setting, both parties need to authenticate each other; that is, we need mutual authentication.

Authenticating Mike gets especially tricky when he's talking to more than one bank at the same time. If the bank receives the message, "Make this $1,000-dollar payment from my account," knowing it came from Mike is insufficient. How does the bank know Mike intended to make payments from this account and not from some other bank? In other words, how can the bank know it was the message's intended recipient? Also, what if Mike has automatic recurring payments and sent exactly the same message last month? How can the bank be sure that somebody pretending to be Mike isn't replaying the message? One more snag: what if Mike is being tricked and only sent the message because he thinks he received $1,000 from the bank as a tax refund mistake?

When authenticating a message, keep in mind these factors:

- Who's the message's source?
- Am I the intended recipient?
- In what context was this message sent?

Answering these questions requires more than establishing the message's source; it also requires knowing the sender's intent. If the authentication process doesn't answer the last two questions, it's flawed, and an attacker can impersonate honest people such as Mike.

I've surreptitiously shifted the conversation from authentication of a party (Mike) to authentication of a message (as being sent by Mike). Intuitively, a difference exists between authenticating the source of messages and authenticating each message separately. In the previous example, this is the difference between the bank asserting that

- I'm talking to Mike (authenticating the source), or
- Mike sent me this message (authenticating the message).

This difference can cause major security breaches. Source authentication requires keeping track of all messages and ensuring they all come from Mike and arrive in the correct order. This is harder than message authentication—that is, ensuring that a single message came from Mike. Think of source authentication as a way to authenticate all messages that follow, in a timely manner and the correct sequence, whereas message authentication only requires establishing a single message's source. As I describe here, source authentication requires (but isn't required by) message authentication.

Consider what the bank is doing when asking for a password. It wants to use a single message containing a password to establish that it's talking to Mike. In other words, the bank wants to somehow use message authentication to obtain source authentication. Going from message authentication to source authentication is by no means trivial. How can we use just one message to ensure all other messages also come from Mike and have arrived in the correct order? A website could easily use a solution for message authentication—for example, calculating HMAC (hash-based message authentication code) for each

message—and mistakenly assume it provides source authentication.

There's a standard way to extend message authentication to source authentication. Set an authentication cookie (such as JSESSIONID) after a password challenge. This technique has an important drawback: capture of the cookie by an attacker is a complete break. Cookies are simply sent back in the clear with every request; the capture of a single request enables authentication of any other request. If an attacker captures the cookie used to authenticate a GET request, he or she can use it to craft a malicious POST request.

Contrast this with an HMAC-based scheme (such as the one for Amazon Web Services storage[2]) that sends a different MAC for each time-stamped request. In the HMAC scheme, compromise of one valid request doesn't let an attacker forge others. This extra security is transparent to the user. Mobile applications don't have to run in a browser and have more options beyond just the cookie mechanism for authentication.

Authentication is so tricky that we risk implicitly making false assumptions if we don't precisely define it. So, what exactly is two-factor authentication, and how is it better than just using a password?

Two Factors, Three Guarantees

We can ask for at least three kinds of proofs of identity:

- something you have (for example, your house keys),
- something you know (for example, your password), or
- something you are (for example, your fingerprints).

These are three distinct authenticating "factors." An authentication mechanism based solely on a username and password is a one-factor authentication mechanism. Two-factor authentication requires the prover to provide two distinct factors to the verifier. For example, a user might use an OTP (one-time password) token (something you have) and a secret PIN (something you know). In two-factor authentication, the verifier won't accept a prover that provides only one factor; both must be provided.

Many one-factor authentication mechanisms are based only on "something you have." A good "something you have" authentication factor has two key features:

- The user doesn't have to remember it.
- It's hard to duplicate.

This implies that mobile device IDs don't make good "something you have" authentication factors, because they're not really secret and therefore become easy to duplicate. They're much more like a username (that is, unique but not secret) than a password (not necessarily unique, but secret). A better alternative is to generate a secret random number on the phone upon first use and use this random number as your "something you have" factor.

The third factor, "something you are," isn't as good as the first two. It's weaker because it is easy to copy and, once copied, can never be revoked. For example, you left a copy of your fingerprints on just about every piece of glass you touched today. Polished surfaces make copying fingerprints easy. In 2008, the German magazine *Die Datenschleuder* published the fingerprint of the German interior minister Wolfgang Schauble, a strong proponent of biometrics, to prove the point.[3] This situation is also true for retina scans, DNA, and the like. They aren't secret, can usually be copied, and can never be revoked. "Something you are" is also much more like a username than a password.

To summarize:

- Avoid using mobile device IDs as "something you have" authentication factors.
- Avoid using "something you are" in two-factor authentication.

Let's look now at what mobile two-factor authentication should do. For simplicity, I'll call the "something you have" factor an *in-phone token* and the "something you know" factor a PIN. Mobile two-factor authentication systems should provide three guarantees.

- By itself, a compromised PIN shouldn't provide a way to authenticate an attacker and shouldn't provide extra information about secrets in any in-phone tokens.
- By itself, a stolen in-phone token shouldn't provide a way to authenticate an attacker and can't leak the corresponding PIN. An attacker with a valid token can, at best, mount a dictionary attack on the PIN and will have to perform many attempts before being authenticated.
- Finally, a compromised verifier (in other words, a hacked authentication server) has no information about the PIN and therefore can't leak it.

The first two guarantees are pretty standard requirements for two-factor authentication. They just state that each factor by itself is useless to an attacker. The second guarantee is strong and was designed to allow the use of short, easy-to-remember PINs from small sample spaces. It means that an exhaustive search among all possibilities is the best attack to find a secret PIN, even when the attacker has the phone. Stolen phones can't compromise user accounts as long as the authentication servers can lock out attackers after too many failed attempts.

The third guarantee prevents the escalation of privilege attacks associated with reusing passwords in distinct services. You don't want your Facebook account to be compromised because someone hacked the Gmail server. If the Gmail server has no information about your Facebook password, its compromise can't by itself lead to compromise of your Facebook account.

Because of mobile applications' underlying use of Web services, such applications use passwords more frequently than desktop applications. So, password reuse and reset probably occur more often in the mobile space. The previous guarantees make two-factor authentication systems safe while acknowledging two facts about "something you know":

- PINs should be short and easy to remember (and type).
- PINs get reused everywhere.

As we'll see, two-factor authentication can even prevent offline dictionary attacks, if you use an online PIN.

Online vs. Offline PINs

Many smartphones can be configured to require an unlock code before they can be used. Many corporate IT security policies also require this. Just like locking your computer screen, this mechanism protects any data on the phone. Also, because it completely blocks access to the phone (except for dialing emergency numbers), the authentication procedure must be able to verify the user's identity—that is, check that the user typed the correct PIN, even when the phone is offline. We call this unlock code an *offline PIN*. Online or offline refers to when users can use the PIN, rather than what kind of data the PIN protects.

Rather than relying on the mobile device itself to prevent a dictionary attack, online PINs rely on servers blocking requests after too many failed attempts. Both online and offline PINs can protect data stored on the phone.

Requiring a PIN to unlock your phone is a two-factor authentication solution. An attacker must both steal the phone (something you have) and discover the unlock code (something you know) to access the phone. This is great for protecting the data on your phone but isn't optimal for accessing online services. Offline PINs rely on the tamper-proof nature of your phone to prevent a dictionary attack, but phones aren't really tamper-proof. A determined attacker will likely be able to bypass the offline PIN, once he or she has your phone.

An online PIN relies on the networked nature of mobile phones to avoid a dictionary attack.[4] The key idea is that the phone itself has no way to verify an online PIN. To verify an online PIN, the phone must make a request to an online server. This is the same way passwords stored in browsers work. The browser has no way to verify that the password is correct by itself; it must contact the correct webserver. Rather than relying on the mobile device itself to prevent a dictionary attack, online PINs rely on servers blocking requests after too many failed attempts. Both online and offline PINs can protect data stored on the phone.

Unfortunately, asking a user to enter an offline PIN to unlock the phone and then immediately enter an online PIN to use a cloud storage application is annoying. Some security-sensitive applications, such as mobile banking, benefit from having an online PIN, but because we can only ask the user to enter a PIN once, offline PINs are winning the race.

Two-factor authentication can provide the three guarantees I described for both online and offline PINs. Consider signing into your Gmail account using an iPhone when the passwords are automatically stored in the browser (and don't need to be reentered). The Gmail servers never learn your iPhone unlock code. So, even if the Google servers get hacked, that PIN isn't compromised.

Clearly, the stored credentials are a technical detail for this user authentication workflow because the user never has to type them. If we completely remove the stored Gmail password and replace it with the device + PIN combo, we can provide the three guarantees.

Avoiding Passwords

We can avoid using passwords when designing a mobile-only application. The user can simply go through the "forgot my password" workflow every time a new device must be provisioned. The only difference is that the workflow's outcome is to establish trust in a new device rather than reset a password.

Many banking websites leave a persistent cookie on their clients' desktops. This cookie identifies the computer used to access the bank website. The bank usually verifies the cookie is present before asking for the user's password. This solution is, in essence, a two-factor authentication solution because the bank verifies both the user's password and the computer being used to access the website. However, because this implementation is tied to the authentica-

tion mechanisms in the browser, it doesn't currently provide the three guarantees. The user's password is usually transmitted to the website during login, and the server keeps a file with salted password hashes. So, if the server is compromised, the password can be leaked. I hope HTML 5 client-side storage improves this and lets us use two-factor authentication also on the desktop.

In the mobile world, we get the device personalization that's achieved by setting a persistent cookie for free. Users can increase the security of their Web banking experience by being required to always type into their desktop browser a six-digit OTP provided by their mobile phone, rather than a password. Alternatively, users can scan a QR (Quick Response) code on the login webpage and do Web authentication through the mobile device itself.

Many possible Web-centric workflows can be implemented that provide the three guarantees and don't use passwords. Their one shortcoming is that because they're two-factor solutions, they require "something you have"—your phone. In other words, unlike the persistent-cookie solution, these solutions all require that a mobile device is available for Web authentication. This can't be made as convenient as storing your passwords in the browser because users must also be in possession of their mobile device. However, it can be made much more secure and provide the three guarantees. Unlike this Web workflow, the mobile-application workflow is easier because the phone itself is the "something you have," making two-factor authentication easier to use than password-based authentication.

The personal nature of mobile devices and the ability to provide the three guarantees make two-factor authentication a better match for the mobile world. Not only can we make the mobile experience more convenient and secure, we can also export that experience back into the desktop. Let's use mobile two-factor authentication and avoid death by a thousand passwords. ☐

Acknowledgments

Thanks to David Every, Chuck Schnider, Priyank Choudhury, and Rob Borcic for many insights.

References

1. A. Felt and D. Wagner, "Phishing on Mobile Devices," presentation at W2SP: Web 2.0 Security and Privacy Workshop, 2011; http://w2spconf.com/2011/papers/felt-mobilephishing.pdf.
2. "Authenticating REST Requests," Amazon Web Services; http://s3.amazonaws.com/doc/s3-developer-guide/RESTAuthentication.html.
3. D. Goodin, "Get Your German Interior Minister's Fingerprint Here," *The Register*, 30 Mar. 2008; www.theregister.co.uk/2008/03/30/german_interior_minister_fingerprint_appropriated.
4. P. MacKenzie and M.K. Reiter, "Networked Cryptographic Devices Resilient to Capture," *Proc. 2001 IEEE Symp. Security and Privacy*, IEEE CS Press, 2001, pp. 12–25.

***Dimitri DeFigueiredo** is a cryptography and security researcher at Adobe. Contact him at ddefigue@adobe.com.*