Exploring Mobile Proxies for Better Password Authentication

Nitesh Saxena[1] and Jonathan Voris[2]

[1] University of Alabama at Birmingham
[2] Columbia University

Abstract. Traditional textual password authentication techniques have numerous well-documented security and usability flaws, yet have seen near universal deployment due to their desirable efficiency properties. As a result, many users who may prefer alternative authentication approaches are forced to use passwords or PINs on a daily basis due to a lack of control over third party servers. This work explores the use of a mobile device as a proxy for password management, in an attempt to improve remote password authentication without making changes to remote servers.
A universal proxy-based authentication framework is presented which allows users to employ a method of their own choice to authenticate locally to their mobile devices (e.g., biometrics or graphical passwords). The framework is also compatible with many communication channels between the mobile proxy and local terminal (e.g., Bluetooth or audio). To demonstrate the practicality of this general framework, a concrete implementation using an "out-of-band" audio channel, called PIN-Audio, is also provided. While existing password management solutions may provide a reasonable level of security for commonplace services, PIN-Audio is recommended for a user-friendly deployment for security critical applications, such as online banking.

Keywords: User Authentication, Passwords, Mobile Devices

1 Introduction

The goal of user authentication is to ensure that only legitimate users are granted access to a computer system while all others are restricted. User authentication can be achieved by establishing credentials between a user and a system, and having users demonstrate that they possess them whenever they wish to access the system. Authentication is one of the most widely studied problems in the realm of computer security. This is due both its fundamental nature, as few security guarantees can be made for a system which allows unauthorized access, as well as the frequency and wide variety of settings in which it takes place.

While many innovative authentication techniques have been proposed, historical and economic factors have stymied the adoption of these novel methods in practice. Updating an entire system of computers to use an alternate approach might be costly and time consuming. As such, despite the great theoretical advancements made in this domain, the vast majority of computer systems are left using basic passwords as their primary form of authentication. Recent developments in mobile devices, such as cell phones, can be utilized to help address this issue. The past decade has seen the emergence of smarter and cheaper mobile phones that have both the computational power and user interfaces necessary to support a wide variety of potential new authentication techniques. Furthermore, phone usage habitats have evolved alongside this technology to the point where some people consider their mobile devices to be more important than their wallets [12].

This paper proposes a way in which mobile phones can be used to place users in control of what authentication method they use. Updating a single mobile phone is far more cost effective than altering an entire computer system; indeed, most cell phone users are already accustomed to installing new applications and software. As a result, this would allow users to select the authentication method that works best for them rather than waiting for a less likely event that the operators of a remote service (that needs authentication) updates their system with a more suitable mechanism.

The core improvement detailed in this work is a framework for providing more secure authentication without necessitating any changes be made to remote servers. The technique is referred to as “proxy-based authentication”. The basic concept is to provide users with a mobile proxy for authentication to a local terminal, which in turn authenticates users to a remote service. Rather than forcing users to remember passwords themselves, leading to short and predictable passwords, passwords will be stored in the portable device, allowing them to be long and fully randomized.

While phone based password management software has been previously proposed, our innovation lies in the automated transfer of credentials from the mobile appliance to the terminal. Furthermore, previous portable password managers again restrict users to standard PIN or passphrase techniques for authentication to the mobile proxy. In contrast, proxy-based authentication allows users to select whichever technique they are most comfortable with for authenticating to the proxy phone. This opens up the possibility for utilizing novel authentication technology that is best suited for mobile hardware without forcing service providers to make any alterations to their systems.

2 Related Work

A vast majority of remote services that are available today utilize password-based authentication. In the absence of action on behalf of service providers, attempts have been made to improve the security and usability of authentication while preserving backward compatibility with passphrases. This section briefly outlines previous solutions of this kind, which are known as *password managers*.

Password managers are programs that accept weak passwords as input and output passphrases that are considered to be strong. This is accomplished by using a computing device to generate strong passwords rather than humans themselves, who behave poorly when asked to create passphrases of sufficient entropy. The appliance can then store the secure passwords that have been generated and output them to its user whenever he or she requires access. Password management software is divisible into three broad categories: desktop, remote, and portable managers.

Desktop password management systems store passwords directly on the terminal that is used to authenticate to remote hosts. High profile examples of programs in this category include Mozilla Firefox [9] and RoboForm by Siber Systems [13]. In contrast, remote managers such as LastPass, developed by the corporation of the same name [8], and Mozilla Weave [10] use one or more non-local servers to keep track of passphrases. The third class of managers utilize auxiliary mobile hardware like cell phones as a password bank. Sperle's KeePassMobile for J2ME enabled devices [5] and OI Safe for the Android platform by OpenIntents UG [11] both fall into this category.

All of these solutions utilize a master password to protect the numerous passwords which they store, therefore increasing usability but having no effect on security. Beyond this, each manager category has its own set of shortcomings. Desktop managers offer no portability for people who use more than a single terminal to authenticate to remote servers. That is, since these programs use the terminal itself as a password repository, they do not provide a mechanism for retrieving these passwords when a different terminal is in use.

While remote managers do allow for use from numerous terminals, they force users to place trust in the system of a third party service provider. This branch of passphrase managers operate by encrypting individual passwords with a master value prior to storing them remotely. They are therefore vulnerable to an offline dictionary attack in the event that these remote machines are compromised. Furthermore, if one computer is used to store passwords for more than one user, an adversary will be able to recover passwords belonging to several users by compromising a single machine. As a final drawback, remote managers are often proprietary, allowing their operators keep the precise details behind how passwords are treated after they leave a user's system guarded as a secret.

In contrast, it is easier to place trust in a portable manager as it can be managed locally by users themselves rather than relying on an external entity to do so [1]. It is also more difficult to eavesdrop on authentication with portable devices due to the small form factor of mobile hardware. Unfortunately, existing mobile password managers suffer from poor usability by requiring that the long and random passwords stored on the portable appliance be manually copied to the authentication terminal. This also provides malicious entities with a potential opportunity for observing the password entry. Such an attack could be accomplished either through casual non-technical attacks like shoulder surfing or sophisticated attacks such as Balzarotti, Cova, and Vigna's video based technique [6] or the audio logging technique introduced by Zhuang, Zhou, and Tygar [7].

3 Secure Authentication Framework from the Client Side

3.1 Threat Model

Before delving deeper into the details of the proxy-based authentication framework, it is necessary to establish the capabilities attributed to adversaries in our system as well as which devices are trusted with which pieces of data. The parties involved in this system are a human user U, a mobile device M, a local terminal T, and a remote server S. In order to provide increased security, rather than placing the burden of generating and remembering a password on U, a password is assumed to be pre-established between M and S.

While U is responsible for recalling the credentials used to authenticate to M, U need not remember or even be aware of the password shared between M and S. It can therefore be as long and random as dictated by the security needs of the application in question rather than the memory and security knowledge of a human user. Whenever U would usually authenticate to S through T, U instead authenticates to M. M reacts by retrieving the encrypted password for S from secure, tamper-resistant storage. This secure storage medium is available on many portable appliances. Only when U authenticates to M is the password corresponding to S unlocked. Next, M authenticates to T, encrypts the passphrase for S and transmits it to T.

If M and T were to share a traditional high bandwidth wireless channel such as WiFi or Bluetooth, this could be utilized to efficiently transmit the encrypted PIN. Doing so would have a unfavorable impact on the framework's universality, usability, and security, however. Along the same lines, since wireless channels are not physically authenticatable, they would leave the channel vulnerable to man-in-the-middle attacks on the framework. For these reasons OOB channels are recommended over conventional wireless channels for forming the secure communication link from M to T. Adversaries are assumed to be capable of eavesdropping on, but not modifying, transmissions over an OOB channel.

In this system, T, S, and M are all trusted with knowledge of the password, but it is only permanently stored in an encrypted form on M. T transmits password values to S without storing them, while S need only store the value produced by hashing password values with a weakly collision resistant hash function. While it is natural that S and M share this secret value, T's awareness of the secret is undesirable. This is because it would be beneficial to be able to authenticate to S using Ts that are public or may be compromised. Unfortunately, T's knowledge of the secret password is a necessary consequence of avoiding any server side changes in this proxy-based authentication framework. If changes to the server were permitted, T could instead blindly pass the encrypted password through to S who would then be delegated the responsibility of decrypting it and recovering the plaintext secret. A third party could perform the same service, but this presents similar and perhaps deeper security challenges to those incurred by trusting T.

3.2 Design

The proxy-based authentication framework is comprised of two overall components. First, a phase occurs where U authenticates to M. This is followed by a phase where the M authenticates itself to T instead of U doing so directly.

User-to-Phone Authentication The authentication primitive that U selects to access M is critical to the framework as a whole both in terms of security and usability. The method used to secure M will unlock all of the passwords it stores, so it must be as resilient to attack as possible. Adding to its importance, using a mobile proxy burdens users by requiring them to interact with an additional device, so the authentication mechanism put into effect must be as usable as possible. Biometric authentication is a good match for these needs and adapts more easily for use on portable devices than for use in alternative settings, such as remote servers.

Phone-to-Terminal Authentication Prior to using this proxy-based authentication framework to authenticate to a given service, users must first initialize M and T. This process need only be carried out once for each T U wishes to authenticate through. The rest of the initialization step depends on the time of channel in use. If M and T share compatible interfaces for a conventional wireless channel, this channel must be bootstrapped by establishing a shared key. Previous work on this topic, also known by "device pairing," can be used to achieve this.

If an OOB channel is to be used in place of an in-band wireless channel as discussed in Section 3.1, a suitable channel must first be selected based on the transmitters and corresponding receivers available on T and M. Since a bidirectional channel is needed, M and T must be equipped with both complementary input and output interfaces. One possibility is to form an audio channel using microphones and speakers. This particular OOB channel is particularly promising due to the ubiquity of these interfaces and is explored in more detail in its own section, 4.

Finally, a shared secret must be established between M and T. Once a given M and T have been properly initialized, the next required step is for M to authenticate to T. This can be executed immediately following the initialization phase when it is required by using particular a particular combination of M and T for the first time. The authentication protocol can be accomplished by using the keys established in the initialization phase to execute any challenge-response (C-R) authentication protocol of the user's choosing.

The difference between the distinct initialization and registration phases should be noted. The initialization phase is required whenever a new M and T are to be used in conjunction with each other. The registration phase, on the other hand, is required whenever a new S is used with M for the first time or when U wishes to refresh the password used to authenticate to S. After initializing, M can register with any number of Ss, after which M can skip directly to the authentication phase when using this service in the future.

3.3 Framework Security Guarantees

In this section the security implications of the proxy-based authentication framework are explored. In Section 3.1, it was established that M, S, and T are all trusted with the password used to authenticate to S. Let p bits be the length of the shared password and k bits be the size of the key by U to authentication to M. Additionally, assume for the sake of simplicity that both M and S have a policy in effect restricting U to q authentication attempts.

Given these values, an adversary has at most a $q/2^p$ chance of success by bypassing M completely and simply attempting to pose as U by guessing passwords and sending them to S for verification. If the attacker compromises the M's tamper-proof hardware and is able to copy the contents M, but is not able to bypass M's access control mechanisms, he or she will have 2^k key possibilities to try in order to gain access to M's data, implying a $q/2^k$ probability of success for this attack. If an adversary instead compromises S, he or she will only be able to recover the weakly collision resistant hashes of the passwords stored on S, since S is assumed to store these values in lieu of saving the passwords themselves.

If a malicious entity was able to gain access to both M and S, he or she could perform a brute force attack to recover the password corresponding to S by performing 2^k hash operations at worst. The most direct attack on this framework would involve recovering the secret U uses to authenticate to M as well as compromising M, in which case all the passwords on M could be unlocked, breaking the security of the framework entirely. If reasonable parameters are selected, such as $p = k = 80$ and $q = 3$, proxy-based authentication achieves computational security against all adversaries except the one who is able to both compromise M and learn the secret to authenticate to M.

4 Illustrative Instantiation Using an Audio Codec

In this segment of the paper, we discuss PIN-Audio, a practical implementation of the theoretical proxy-based authentication framework introduced in Section 3. It is possible to use a conventional wireless channel as a communication link between M and T was mentioned. Due to the universality, security, and usability issues involved, it may not always be possible or desirable to use such a channel. Fortunately, M and T will always feature some other forms of output interfaces. In scenarios where M and T share corresponding input and output interfaces, these can be used to construct an OOB channel instead. This section proposes the use of an audio channel as a basis for transmissions from T to M and vice versa. In essence, a C-R protocol adhering to the framework established in Section 3.2 will be executed over this audio channel. While any authentication mechanism can be used by U to access M, we opted to use a standard PIN based approach. A pictorial representation of this concrete version of proxy-based authentication is provided in Figure 1

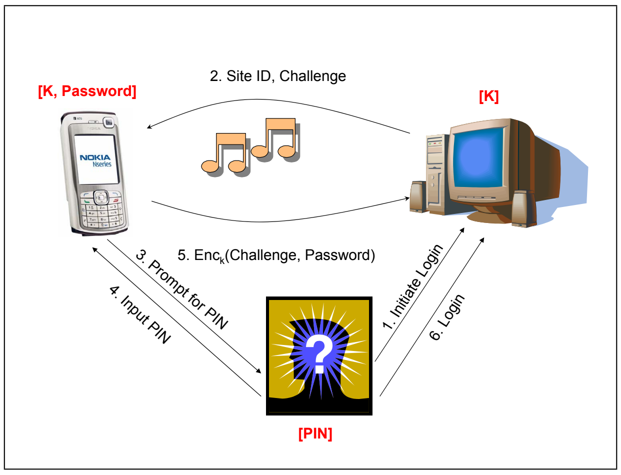

Fig. 1. Authenticating to a Remote Server Using a Mobile Proxy and an Audio Channel

4.1 Design and Implementation

In order to support this authentication system it is necessary to install new software on T and M. T requires a password client application while a password server program is needed by M. Recall, however, that no modifications of S need to be made in order to accommodate PIN-Audio. In this implementation the initialization phase was assumed to already be completed. That is, a pre-shared symmetric key was simply copied on to M and T at installation time. In practice, this can be achieved by performing a Diffie-Hellman key exchange over the audio channel as described in Section 3.2. This technique does handle both the registration and authentication steps required to access a remote service, though. Both of these phases proceed as per the framework outlined in Section 3.2.

Device Setup Before proceeding with the implementation, specific devices had to be chosen to fit all of the players involved in the proxy-based authentication framework provided in Section 3.1. A Dell desktop computer with Windows XP as an operating system was selected for T as would be the case in a practical implementation. Rather than actually using a cell phone as M, however, a simulated proof-of-concept prototype was developed using a Dell laptop computer that was also running Microsoft Windows XP. With built in microphones and speakers, these devices had all the hardware necessary to serve the roles of T and M respectively. Thus, a password server application was designed for the laptop and a password client application was crafted for the desktop. Since no changes to it were necessary, S was left out of this simulation.

Construction of a Robust Audio Channel Data is encrypted prior to transmission. The resultant ciphertext is used as input to a Base64 encoder in order to facilitate transfer via audio. Base64 was selected because this encoding leads to lower error rates for audio transmissions. This is owed to the fact that byte encoded data produces values that are outside the range of sounds that can be reliably produced on low quality audio hardware. In contrast, a Base64 encoding ensures that data is within the required range. unfortunately, this higher reliability comes at the cost of a decrease in efficiency, as Base64 encoded data takes 1.33 times as long to transmit as equivalent data under a byte encoding.

Once Base64 encoded, the data is next passed to Schifra, which is a robust and open source implementation of the Reed-Solomon (RS) Error Correcting Code (ECC) developed by Partow [2]. RS ECC is required to guard the audio data against transmission errors and to perform forward error correction. This is a necessary component as retransmitting data in this setting is too costly to be viable. With these preprocessing steps out of the way, the proper conversion of data to sounds can begin. To attain this, the RS ECC processed data is encoded on last time using the Pulse-Code Modulation codec of Lopes and Aguiar's Digital Voices project [3, 4]. This codec is robust, working well in environments with high levels of ambient noise, as well as usable, since it uses a pleasant "Soprano Flute" sound as a basis for its transmissions. A start marker or "initial hail sequence" and end indicator or "stop sequence" are employed in order to detect the beginning and end of the audio based data transmissions.

Intuitively, once encoded the audio data is sent through the originating computer's speakers and received by the destination device's microphone. The decoding process at the recipient's end is the inverse of the encoding process. In order to provide security, it is only necessary to encrypt the data being sent from M to T. Leaving the channel from T to M open will impact the protocol's privacy, however, since T's responses contain an identifier of the S that U wishes to access. In order to achieve privacy as well as security the link from T to M can be encrypted as well. This implementation opted for security as well as privacy by encrypting transmissions in both directions.

Desktop Password Client Application The password client program developed for T can be divided into five main components: a keyboard listener, an active window handler, an encryption/decryption and encoding/decoding engine, an audio codec engine, and a key thrower. The keyboard listener comes into play first. When U presses the keyboard shortcut associated with the password client (for PIN-Audio, the F8 key was used in this capacity) this portion of the program triggers the application.

The software's active window handler then checks if the window that is currently active is a web browser. If this is the case, it extracts the name of the web site that is currently active in the browser. Note that while PIN-Audio only supports authentication to web sites through a browser, the proxy-based authentication framework can be extended to support authentication to any remote server S. Next, the client generates a 80 bit long random nonce and concatenates it with the specified request type and S's identifier. Possible options for the request type are login, registration, and password change. The two engine segments are then called upon to encrypt and encode this data as detailed in Section 4.1, which is played through T's speakers.

Once T has finished its audio transmissions, it shifts to its listener component to wait for the start sequence of the response from M's password server program. When this special value is detected, T's application captures audio until it notices the designated stop sequence value. This acquired audio is decrypted and decoded again using the process provided in Section 4.1. Finally, if the nonce that was initially sent by T matches the nonce M sent back to it, the software's key thrower places the transmitted password in the correct field of the web site that is currently being viewed.

Laptop Password Server Application The password server for our laptop M was written in Java, with the exception of Schifra, Partow's C++ RS ECC implementation. Just as with the desktop password client, the laptop server executed this code through a shell. Further, the encryption, decryption, encoding, and decoding processes all occur in the same fashion as T's client program as it is laid out in Section 4.1. As soon as M's server application starts it begins listening for the unique audio start sequence. When this has been detected the program decodes the received audio and asks U to authenticate to M by entering his or her PIN.

After authenticating, M requests that U confirm the request sent by T. If U accepts, M reacts as dictated by the transmitted request type. If the solicitation is for registration, a password of the minimum length deemed secure for the application at hand is generated uniformly at random. The passphrase is then encrypted and stored. Note that in a real setting tamper resistant hardware would be used to store this sensitive data. If a login type request is received, an existing password corresponding to S is retrieved from the phone's memory. Password change requests were omitted from the PIN-Audioprototype. Irrespective of the request type, M always concludes by transmitting the proper passphrase over the audio channel. After handling a request from T's client, M's server immediate resumes waiting for the next client request to arrive as indicated by the initial hail sequence.

4.2 Implementation Security Guarantees

PIN-Audio utilizes a 4 decimal digit PIN for user-to-phone authentication. Assuming that 4 decimal digits are equivalent to 15 bits, the chance of success for an attack scenario where M alone is compromised becomes $\frac{q}{2^{15}}$ at best. This contrasts with the security offered by conventional user selected passwords, which can be guessed with a maximum probability of $\frac{q}{|D|}$ where D is the dictionary containing all of U's possible password choices. PIN-Audio clearly offers better security in cases where $|D| < 2^{15}$. Compromising S in place of M would yield an attacker no advantage when PIN-Audio is in use. This is also an improvement over normal passwords, which can be recovered by launching a dictionary attack on a compromised S. Like the general framework, PIN-Audio offers no real security in the scenario where both M and S were compromised, though. This is due to the fact that a malicious entity could recover the password for S by performing at worst 2^{15} hash operations.

5 Conclusion

This paper presented a mobile proxy-based framework for authenticating to remote servers. This system leverages a personal, portable device in a novel manner by using it as an intermediary between a user and the authentication terminal used. This provides the possibility of performing both secure user-to-phone authentication and cryptographic phone-to-terminal authentication. This scheme also provides better resistance to observational attacks compared to other approaches. Most critically, it can be readily utilized by users in search of stronger security without requiring any changes be made to existing server architectures. While the manual transfer of shorter, less secure passwords offered by alternative mobile password managers may be sufficient for less sensitive applications, PIN-Audio is recommended for authentication to online services, such as banking, that demand high levels of security.

Acknowledgments

This work is supported, in part, by the NSF grant CNS-1209280. The authors also thank Md. Borhan Uddin for his work on the audio codec used in the implementation of PIN-Audio.

References

1. A. Karole, N. Saxena and N. Christin. A Comparative Usability Evaluation of Traditional Password Managers. In *Information Security and Cryptology (ICISC)*, 2010.
2. A. Partow. Schifra Reed-Solomon Error Correcting Code Library. Available at: `http://www.schifra.com`, 2010.
3. C. Lopes. Digital Voices. Available at: `http://www.ics.uci.edu/~lopes/dv/dv.html`, 2003.
4. C. Lopes and P. Aguiar. Acoustic Modems for Ubiquitous Computing. In *Pervasive Computing*, 2003.
5. C. Sperle. KeePassMobile. Available at: `http://www.keepassmobile.com`, 2010.
6. D. Balzarotti and M. Cova and G. Vigna. ClearShot: Eavesdropping on Keyboard Input from Video. In *Symposium on Security and Privacy*, 2008.
7. L. Zhuang and F. Zhou and J. Tygar. Keyboard Acoustic Emanations Revisited. In *Conference on Computer and Communications Security*, 2005.
8. LastPass Corporation. LastPass Password Manager. Available at: `https://lastpass.com`, 2010.
9. Mozilla Corporation. Firefox Browser. Available at: `http://www.mozilla.com/firefox`, 2010.
10. Mozilla Corporation. Weave Sync. Available at: `http://labs.mozilla.com/projects/weave`, 2010.
11. OpenIntents UG. OpenIntents Safe. Available at: `http://www.openintents.org/en/node/205`, 2009.
12. R. Kim. The World's a Cell-phone Stage. In *The San Fransisco Chronicle*, Available at: `http://www.sfgate.com/cgi-bin/article.cgi?f=/c/a/2006/02/27/BUG2IHECT01.DTL`, 2006.
13. Siber Systems. RoboForm Password Manager. Available at: `http://www.roboform.com`, 2010.