DPAC: A Reuse-Oriented Password Authentication Framework for Improving Password Security *

Hua Wang, Yao Guo, Xiangqun Chen
Key Laboratory of High Confidence Software Technologies (Ministry of Education)
Institute of Software, School of EECS, Peking University
{wanghua04, yaoguo, cherry}@sei.pku.edu.cn

Abstract

Traditionally, password authentication is distributed to each application, so developers have to take countermeasures by themselves to defend passwords against various threats. This requires a great amount of effort, a lot of which is repetitive. The high cost poses a potential hindrance to the adoption of countermeasures.

This paper proposes a new reuse-oriented password authentication framework, called Desktop Password Authentication Center (DPAC)*, to reuse counter-measures among applications, thus reducing the cost of defending passwords against threats. In DPAC, we move the task of authentication, as well as the responsibility for protecting passwords, from applications to a dedicated Authentication Center (AuthCenter), so that countermeasures only need to be taken in AuthCenter and afterwards are reused by all applications. This solution can eliminate a lot of repetitive work and reduce the cost significantly. We demonstrate the feasibility of DPAC by implementing a prototype, in which we migrate the widely used OpenSSH to DPAC and implement two example countermeasures.*

1 Introduction

The security of passwords is extremely important for password-based network applications. Numerous researchers have attempted to enhance the security of passwords from various perspectives. This paper focuses on enhancing the password security in the client side by lowering the cost of taking countermeasures against threats.

The traditional password authentication scheme distributes authentication tasks to each application. This scheme therefore is referred to as decentralized password authentication (DPA) in this paper. With DPA the responsibility for protecting passwords is also distributed to each application, because passwords are processed in applications' memory. The developers have to take corresponding countermeasures by themselves to defend against various threats, so protecting passwords in DPA often increases the development cost considerably, resulting few or even none of existing countermeasures being adopted.

An effective approach to lowering the cost is to reuse countermeasures among multiple applications. The idea of reuse has been adopted by the Pluggable Authentication Module (PAM) framework [8] to lower the cost of applying and changing authentication mechanisms. But PAM is incompetent to reuse countermeasures in the client side because passwords still appear in the memory of applications. PAM modules receive passwords from applications, so users have to input their passwords to applications. Besides, PAM modules are loaded into the address spaces of applications and all their data is accessible by applications, including passwords. Consequently, countermeasures must be taken in each application to protect passwords. Essentially, the password authentication scheme of PAM is DPA.

We propose a new reuse-oriented password authentication framework, called *Desktop Password Authentication Center (DPAC)*, to reuse countermeasures among applications in the client side. The key idea behind DPAC is to move the task of password authentication from applications to a dedicated password manager, called Authentication Center (AuthCenter), thus keeping applications away from passwords. AuthCenter is shared by all applications, so countermeasures can be reused, eliminating lots of repetitive work and reducing the cost significantly. It is easier to apply many sophisticated and effective countermeasures to AuthCenter to improve the password security for all applications. In addition, DPAC also provides great flexibility so that adding or replacing countermeasures and authentication protocols is convenient.

We demonstrate the feasibility of DPAC by implementing a prototype, in which two example countermeasures are

*This research is supported by the National High Technology Development 863 Programs of China under Grant No. 2007AA01Z462, 2007AA010304 and 2008AA01Z133.

1530-2059/08 $25.00 © 2008 IEEE
DOI 10.1109/HASE.2008.22

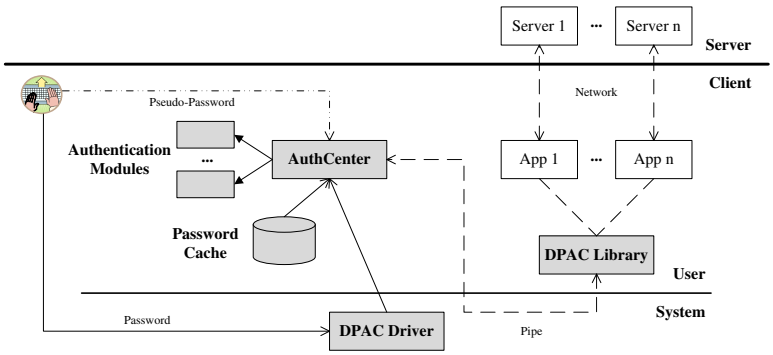

Figure 1. Architecture of DPAC Framework

taken in AuthCenter to thwart user mode keylogger and fake login form respectively. We also migrate the widely used OpenSSH to DPAC.

The rest of this paper is organized as follows. Section 2 describes the design of DPAC. Section 3 describes two example countermeasures used in DPAC. Section 4 describes the migration of OpenSSH to DPAC. Section 5 discusses related work. And the conclusion is given in section 6.

2 Design of DPAC

2.1 Architecture of DPAC Framework

The main purpose of DPAC is to isolate passwords from applications during authentication, so that countermeasures only need to be taken in AuthCenter and are reused by all applications. Thus we need to process passwords in a separate and dedicated address space inaccessible by applications. The overall architecture of the DPAC framework is illustrated in Figure 1. The shaded parts are DPAC components, whose functionalities are described below.

AuthCenter is the main component in the framework, performing all password-related operations. It is designed as a separate program having its own address space, so that passwords in it are not accessible by other applications.

The authentication service provided by AuthCenter is encapsulated in a dynamic library, called *DPAC library*. This library hides the details of interactions between applications and AuthCenter, and provides a simple and clear interface to application developers. Currently the interface contains only one function, namely *dpac_authenticate*.

For the sake of convenience, we add the capability of caching passwords to DPAC. The *password cache* is a group of files used to store passwords. All cached passwords are under the control of AuthCenter. Each user has its own password cache file, and is allowed to set an access password, preventing accesses from other users.

To increase flexibility, the client-side operations of different authentication protocols are packed into separate modules, namely *authentication modules*. These modules are implemented as dynamic libraries that can be loaded and replaced at runtime.

We also modify the keyboard driver to thwart user mode keyloggers. The modified driver is called *DPAC driver*, which intercepts and translates all typed characters when a user is entering his password.

2.2 Authentication Process

In DPAC, the password authentication process involves four parties: user, server, AuthCenter and application. The outline of the password authentication process is as follows. First the application calls the *dpac_authenticate* function in the DPAC library to launch AuthCenter, which in turn displays a login form to the user. In this form the user may choose to input a new password or select a cached one. Then AuthCenter loads a proper authentication module according to *modname* passed from the application and begins to communicate with the server to verify the password. The messages between them are relayed by the application.

Various authentication protocols may be used in DPAC, as long as they satisfy the following principle: the messages between AuthCenter and the server should not reveal passwords to applications. This principle ensures that passwords will not be propagated to applications. We use a simple challenge-response protocol based on HMAC [2] as an example to demonstrate the authentication process. The steps are as follows:

1. The application first connects to the server and sends a login request to the server. Then the server sends *modname* back to inform the application which authentication protocol should be used, as well as necessary parameters for the authentication module. Here the parameter is *nonce*, a random value used as the challenge.

2. The application calls *dpac_authenticate* to launch AuthCenter, passing *modname* to AuthCenter to inform

which authentication module should be loaded. *nonce* is also passed to AuthCenter.

3. According to the received *modname*, AuthCenter looks up and loads a proper module. Then AuthCenter prompts the user to input his *id* and *password* or select a cached one.

4. AuthCenter calculates *auth*, the HMAC code of *nonce*, with the password as the shared key. *id* and *auth* are sent back to the application. The application relays them to the server.

5. The server checks *auth* by recalculating the HMAC code of *nonce* and comparing it with *auth*. The result of the authentication is returned to the application.

In the authentication process, all password-related operations are done in AuthCenter. Therefore the application does not need to care about the protection of passwords. It only needs to call *dpac_authenticate* and receive the result from the server. So writing the authentication code in DPAC is much simpler than in the traditional scheme.

3 Taking Countermeasures in AuthCenter

This section describes the adoption of two example countermeasures, thwarting user mode keyloggers and fake login forms respectively. We demonstrate that during the adoption only the AuthCenter is affected, without requiring applications to be modified.

3.1 User Mode Keylogger

The keylogger is one of the most insidious and prevalent threats to passwords. To thwart this threat, we use a countermeasure similar to PwdHash [7], shown in Figure 2. PwdHash hooks low level key event handler and filters key events when a user enters his password. We enhance the idea by modifying the keyboard driver in the kernel, adding a translator and a password buffer, as well as dividing its working mode into normal mode and password mode. When the driver works in the normal mode, the translator simply passes what the user types to applications, just like the unmodified driver. While working in the password mode, the translator will send all characters to the password buffer. At the same time it translates each character, except for control characters such as enter and backspace, to a default character, e.g. "*", and sends them to applications. Finally, what the keyloggers and AuthCenter receive is a string only consisting of default characters. The real password is saved in the password buffer. It can be retrieved through a dedicated system call which can only be invoked by AuthCenter.

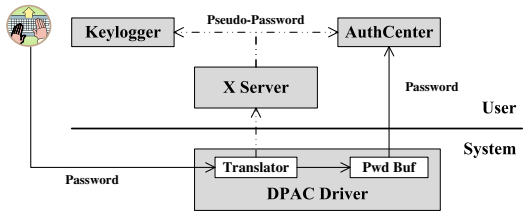

Figure 2. DPAC Driver

3.2 Fake Login Form

Malicious programs may display fake login forms to spoof users into entering their passwords. Some countermeasures have been proposed to thwart this threat. We use a picture-based countermeasure similar to Dhamija and Tygar's scheme [3]. The appearance of the login form displayed by AuthCenter is customizable by users. A user is permitted to register a picture to AuthCenter through a dedicated configuration tool, which is used as the secret between the user and AuthCenter. Subsequently, when the user launches AuthCenter, it will display the login form with the user's picture as the background. Malicious programs do not know what the picture is and can not display the correct background, so the user is able to recognize fake login forms easily.

4 Applying DPAC to OpenSSH

DPAC simplifies the task of writing authentication code when developing new applications. Besides, the cost of migrating existing applications to DPAC is also low. We demonstrate this by applying DPAC to widely used OpenSSH. The result shows that the required modification is very small, only involving two functions that belong to *ssh* and *sshd* separately.

To migrate OpenSSH to DPAC, we need to move the password related operations from *ssh* to AuthCenter. Fortunately, these operations are implemented in a single function, i.e. *userauth_passwd*. The new *userauth_passwd* function needs to complete the task of calling the DPAC library to launch AuthCenter and passing messages between AuthCenter and *sshd* back and forth.

The original authentication protocol of OpenSSH requires *ssh* to send the password to *sshd*. This protocol does not satisfy the aforementioned principle, because its messages will reveal the password to *ssh*. We replace the original protocol with a simple HMAC-based protocol. The client side logic of the protocol is packed into an authentication module, namely *pwd_hmac.so*.

As the authentication protocol changes, *sshd* also needs to be modified. For *sshd* has already used the PAM framework to hide the low-level authentication mechanisms, the

modification to *sshd* is even smaller. We implement the server side logic of the HMAC-based protocol as a PAM module, and modified the configuration file to use this module. The only modification needed is to rewrite the conversation function, i.e. *sshpam_passwd_conv*.

During the process of migrating OpenSSH to DPAC, we only modified two functions and less than 100 lines of code. It is a very small portion of OpenSSH, which contains about 58,000 lines of code.

5 Related Work

Kerberos [4, 9] is a widely used centralized authentication system in distributed environments. An important common place between DPAC and Kerberos is to keep secrets in a secure third party. The comparison between them is shown in Figure 3.

In Kerberos, the Key Distribution Center (KDC) keeps secrets shared with the application servers. To access the application server, the client first authenticates itself to KDC. Then KDC sends the client a ticket that contains a session key encrypted by the shared secret and can be used by the client to log in the application server. During the process, the shared secret does not appear in the client.

In DPAC, the shared secret between AuthCenter and the server is the user's password. If an application is going to access the server, it sends a request to AuthCenter. Then AuthCenter sends necessary messages on behalf of the application to sign in the server. During this process, the password also does not appear in the application. Therefore to some extent, DPAC can be viewed as a simplified desktop version of Kerberos.

The PAM framework [6, 8] is proposed to hide low-level authentication mechanisms and provide a high-level abstraction. In PAM, different mechanisms are implemented as separate pluggable modules that can be replaced easily. DPAC also provides the similar flexibility.

PAM does not consider how to defend against malicious applications that invokes it. PAM modules are loaded into the address spaces of invoking applications and all their operations and data can be observed by applications. While, in DPAC, authentication modules are loaded into a separate password manager (i.e. AuthCenter). They are hidden from invoking applications.

Password managers provide users with the capability of managing all passwords with a single tool. To guarantee the security of passwords, some managers, such as GnomeKeyring [5] and Mac OS Keychain [11], encrypt passwords before they are cached. More secure managers use secure hardware to enhance the security of passwords [12, 1]. However, none of these managers provides the capability of verifying passwords, so passwords need to be delivered to applications during authentication.

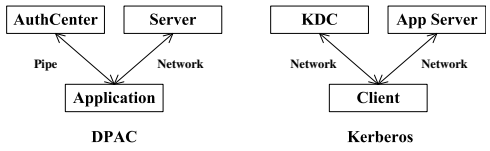

Figure 3. DPAC and Kerberos

6 Conclusion

Some vendors may be reluctant to take enough countermeasures in their products because of the high development cost. In this paper we propose a new reuse-oriented password authentication framework, DPAC, to lower the cost by moving the responsibility for protecting passwords from applications to a dedicated manager (i.e. AuthCenter) and reusing countermeasures for all applications.

We demonstrate the feasibility of DPAC by taking two example countermeasures, as well as migrating OpenSSH to DPAC.

References

[1] T. Aron. Client security solutions, October 2004.

[2] M. Bellare, R. Canettiy, and H. Krawczykz. Keying hash functions for message authentication. In N. Koblitz, editor, *the 16th Annual International Cryptology Conference on Advances in Cryptology, LNCS 1109*, pages 1 – 15, Santa Barbara, California, 1996. Springer-Verlag.

[3] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *the 2005 Symposium on Usable Privacy and Security*, pages 77 – 88, Pittsburgh, Pennsylvania, 2005. ACM Press.

[4] J. T. Kohl and C. Neuman. The kerberos network authentication service (v5), September 1993.

[5] A. Larsson. Proposal for inclusion in desktop: Gnome-keyring, November 2003.

[6] A. G. Morgan. Pluggable authentication modules for linux. *Linux Journal*, 1997(44es), 1997.

[7] B. Ross, C. Jackson, and N. Miyake. Stronger password authentication using browser extensions. In *the 14th Usenix Security Symposium*, pages 17 – 32, Baltimore, 2005.

[8] V. Samar. Unified login with pluggable authentication modules (pam). In *the 3rd ACM conference on Computer and Communications Security*, pages 1 – 10, New Delhi, India, 1996. ACM Press.

[9] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *the Winter 1988 Usenix Conference*, pages 191 – 202, Dallas, Texas, USA, 1988.

[10] TCG. Tcg specification architecture overview, August 2007.

[11] S. d. Vries. Securing mac os x, May 2006.

[12] Wave. Wave systemsembassy trust suite portfolio enables secure business computing, 2003.