An Intrusion-Tolerant Password Authentication System

Xunhua Wang[‡], M. Hossain Heydari
Commonwealth Information Security Center &
Department of Computer Science
James Madison University
Harrisonburg, VA 22807 USA
{*wangxx,heydarmh*}*@jmu.edu*

Hua Lin
WySTAR Global Retirement Solutions
Wachovia Corporation
Rockville, MD 20850 USA
hua.lin@wystar.com

Abstract

In a password-based authentication system, to authenticate a user, a server typically stores password verification data (PVD), which is a value derived from the user's password using publicly known functions. For those users whose passwords fall within an attacker's dictionary, their PVDs, if stolen (for example, through server compromise), will allow the attacker to mount off-line dictionary attacks. In this article, we describe a password authentication system that can tolerate server compromises. The described system uses multiple (say n) servers to share password verification data and never reconstructs the shared PVD during user authentications. Only a threshold number (say t, $t \leq n$) of these servers are required for a user authentication and compromising up to $(t-1)$ of these servers will not allow an attacker to mount off-line dictionary attacks, even if a user's password falls within the attacker's dictionary. The described system can still function if some of the servers are unavailable. In this paper, we give the system architecture and implementation details. Our experimental results show that the described system works well. The given system can be used to build intrusion-tolerant applications.

Keywords: *Intrusion tolerance, off-line dictionary attack, password-authenticated key exchange (PAKE)*

1. Introduction

Passwords are widely used for authentication. In a password-based authentication system, a user (also called client hereinafter) holds a memorizable password and a server stores related password-verification data (PVD). PVD is a value derived from the password using publicly known functions and is used by the server for client authentication. For example, in most Unix systems, PVD is the hash of user ID, a salt, and the user's password. Password-based systems are notoriously vulnerable to the dictionary attack, in which an attacker does not exhaust all possible passwords but works on a smaller dictionary of likely passwords for password searching. It should be noted that for a specific user, his password might not always fall within an attacker's dictionary but a high fraction of the actual passwords may match passwords of a carefully constructed dictionary, even when a proactive password checking program is used*. Wu [31] has demonstrated that, in a Kerberos system where a proactive password checker is used, about 8% (2045 out of 25,000) of the passwords are still vulnerable to dictionary attacks.

With a password, the client can use two different modes to authenticate himself to the server. First, he can simply send his password to the server, which applies the related PVD to check the validity of the received password. For security reasons, the password should be encrypted before it is transferred. Most password-based authentication systems on the web employ this mode: a Secure Socket Layer (SSL) connection is established first between the client and the server and then a password is sent to the server via the SSL connection for client-side authentication. Since each SSL session establishes a random session key by which the password is encrypted, if an attacker eavesdrops the encrypted password, he will not be able to replay it or decrypt it. Neither can he use the encrypted password to mount an off-line dictionary attack, as the encrypting SSL session key is random. Throughout this article, this type of authentication is

‡ A part of this work has been supported by a Cisco CIAG grant.

* Most proactive password checkers use well-defined rules for password checking [31] and an attacker could construct a better password dictionary so that some passwords may escape the proactive password checking but still fall within the attacker's dictionary. Interested readers can read [31] for methods to construct such a password dictionary.

called *secure password transfer*. We stress that in this mode the server stores PVD for password verification.

The second mode for client-side authentication is to demonstrate the possession of the password without sending it. The password-based challenge/response authentication system [21] falls into this mode: a client authenticates himself by computing a *response* using his password and a *challenge* chosen by the server. It is this computed response, not the password, that is sent back to the server. However, the password-based challenge/response system is still not secure since an attacker can eavesdrop a (*challenge*, *response*) pair and use them to mount off-line dictionary attacks. A new protocol paradigm following this path, called *password-authenticated key exchange (PAKE)*, was developed [3]. In PAKE, user's password is not transferred at all and instead, the client authentication is accomplished through the capability of establishing an *authenticated* session key† with the server, who stores the related PVD. This authenticated session key would not be possible if the client does not have the password or the server does not have the related PVD. A good PAKE protocol is secure against both the eavesdropping-based dictionary attack and any active attacks. PAKE protocols achieve these security goals through the marriage with public key exchange techniques. To date, several PAKE protocols have been proposed, including the Encrypted Key Exchange (EKE) [3, 4], Secure Password Exponential Key Exchange (SPEKE) [18, 17], Simple Remote Password (SRP) [30], the PAK protocol [5], the BPR00 protocol [1], the SNAPI protocol [23], and the KOY01 protocol [20]. The SPEKE, SRP, PAK and KOY01 protocols use the Diffie-Hellman key exchange algorithm [12] while BPR00 and SNAPI use the RSA algorithm [27]. We stress three facts about PAKE: 1) a PAKE user possesses a password only; 2) the client program used by the user to log into the system has only system parameters (such as the g and q for Diffie-Hellman) and no secrets (say, a private key) are hard coded into it. Thus, an attacker cannot compromise PAKE security by obtaining a copy of the client program and examining it; 3) the client program should be reliable in that its code is not corrupted. (Otherwise, a corrupted client program can simply keep a copy of the password typed by the user and send it to the attacker.)

THE PROBLEM. Both the secure password transfer approach and the PAKE approach address off-line dictionary attacks from the network very well. However, the secure password transfer approach and most of the existing PAKE protocols use a single server to store users' PVD. Recall that PVD is derived from passwords using a publicly known function. This makes them vulnerable to another type of off-line dictionary attack: for those passwords that fall within an attacker's dictionary (such as the 2045 passwords out of the 25,000 reported in [31]). If the attacker managed to compromise the centralized server and steal the PVD, he could simply guess a password (from his dictionary), compute the corresponding PVD (using the publicly known function) and verify the correctness of the password by comparing the computed PVD against the stolen PVD. On the other hand, server exposure seems inevitable. For instance, an attacker might gain the `root` privilege of the server by exploiting bugs in server software (for instance, bug [9] in the Apache web server, and bug [8] in Kerberos server). It should be noted that simply applying a threshold secret sharing scheme (such as the Shamir secret sharing) over PVD does not solve the problem since the shared secret (i.e., the PVD) is reconstructed when used, which still allows an attacker to mount an off-line dictionary attack if the reconstruction point is compromised.

OUR CONTRIBUTION. In this article, we describe an intrusion-tolerant password authentication system. This system employs multiple (say n) servers — called `PVD servers` — to store PVD. Among the multiple PVD servers a user's PVD is shared and the shared PVD is never reconstructed during user authentication. The described system is intrusion-tolerant in the sense that compromising up to $(t-1)$, $2 \leq t \leq n$, servers will not allow the attacker to mount off-line dictionary attacks and the system can still function if $(n-t)$ servers go down. We explore the architecture and implementation details of the intrusion-tolerant system. Our experimental results show that the described system works well. One example application of our described system is to enhance the security of the web applications that use password-based authentication. Other password-based applications such as `Telnet` and `FTP` can also be made intrusion-tolerant by integrating the described system.

This article is organized as follows. Section 2 gives the related work and Section 3 presents the system architecture. Section 4 describes the system setup procedure and Section 5 gives the data flow among the system components. In Section 6, we describe the implementation details of the intrusion-tolerant authentication system, give some performance data and present methods to improve the performance. Section 7 discusses some operational issues and Section 8 explores how the described system might be used to build intrusion-tolerant applications. Concluding remarks are given in Section 9.

2. Related Work

For trusted third party (TTP)-based authentication systems such as Kerberos, Gong [15] gave techniques to add intrusion tolerance to the trusted third party. The intrusion-

† This authenticated session key is cryptographically strong and can be used to protect subsequent communications after the authentication.

tolerant password authentication system described in this article is not based on any trusted third party.

MacKenzie *et al.* [24] proposed the first threshold PAKE protocol (called `MSJ02` hereinafter), which uses multiple servers to share PVD and is provably secure under the random oracle model [2, 7]. Di Raimondo and Gennaro [10] gave another provably secure (under the standard assumption model) threshold PAKE protocol (called `DG03` hereinafter), which is built on the `KOY01` PAKE protocol [20]. (The `KOY01` PAKE protocol provides provable security (under the standard assumption model) but it uses a single server to store PVD and the `DG03` protocol enhances it by using multiple servers.) Both the `MSJ02` protocol and the `DG03` protocols are theoretically significant in that they provide provable security (under different security models). However, they are relatively inefficient due to their complexity. For example, the `DG03` protocol uses one-time digital signature as building block. Since one-time digital signatures are typically inefficient, this makes `DG03` inefficient in communication. (For instance, if the modulus p is 1024 bit and the Merkle one-time digital signature scheme is used in `KOY01` and `DG03`, both the one-time verification key and the one-time digital signature on $(\beta'|K)$ will be 20 kilobytes, which is pretty big for a password-based authentication.) We believe that, for a password-based authentication system, both security and efficiency are important. In contrast, the intrusion-tolerant password authentication system described here uses a different model first proposed in [29]. In this model, the assumption on the client program is further weakened and, besides (g_1, g_2, N, q), an additional *public* value, $\beta = {g_1}^{\delta}$, is also hard coded into the client program and δ^{-1} is shared among the multiple PVD servers. This change makes it much easier for us to achieve intrusion tolerance on the server side. We like to stress that, in our intrusion-tolerant system, the client program has no secret hard coded into it and that the client program can be examined by attackers without losing security. Compared to [29], the protocol used in the described system is similar but more scalable, which we shall elaborate later.

3. System Architecture

Depicted in Figure 1 is the architecture for the intrusion-tolerant password authentication system. There are four components in the system: *the client program*, *the service server interface*, *the PVD servers* and *the management server*. The client program runs at the client side. It accepts user name and password, connects to the service server as requested, performs a PAKE authentication on the user's behalf to the service server and establishes an authenticated (cryptographically) strong session key to protect subsequent communication between the client and the service server. There is no secret hard coded in the client program and the client program erases the user's password from memory immediately after the successful login. The service server interface runs on the service server such as `Telnet`, `FTP` and web server. It is activated when a user login request is received and it then connects to the available PVD servers for a user authentication. The PVD servers stay behind the service server and are invisible to the client. A user's PVD is shared among the multiple PVD servers and is never reconstructed during the user authentication. For a user authentication, only a threshold number of PVD servers are required. If a user authentication is successful, the participating PVD servers help the service server compute an authenticated session key which can also be computed independently by the client program since it has the password typed in by the user. The management server is used to enroll new users to the system, reset/regenerate user passwords, and audit system activity.

Communication links. In our system, the communication links between the service server interface and the PVD servers are protected. These secure connections are required for the transfer of session key shares from participating PVD servers to the service server and they are realized through `SSL` [11], in which the service server interface acts as the SSL client and the PVD servers act as the SSL server. Both the SSL client and server authentications are enabled. The connections between the management server and the PVD servers are also protected through `SSL`, in which the management server acts as the SSL client and the PVD servers act as the SSL server. In these connections, both SSL client and server authentication are also enabled. Over these secure connections the PVD shares are distributed to the PVD servers after the management server generates them.

4. System Setup

There are two parameters about the PVD servers, t and n. n is the number of PVD servers and t, $t \leq n$, is the minimum number of PVD servers to be trusted. t is chosen when the system is initialized while n can dynamically change. The multiple PVD servers are numbered as PVD server $1, 2, \ldots, n$.

When the system is initialized, on the management server, four public parameters, (g_1, g_2, N, q), are generated where N is a safe prime, that is, $N = 2q + 1$ and q is also a prime. g_1 and g_2 are two elements in finite field F_N and their orders are q. g_1 and g_2 are randomly chosen so that both the discrete logarithm of g_1 to the base g_2 (that is, $\log_{g_2} g_1$) and the the discrete logarithm of g_2 to the base g_1 (that is, $\log_{g_1} g_2$) are unknown.

- The selection of N and q. In our implementation, we used the prime from the IKE (Internet Key Exchange)

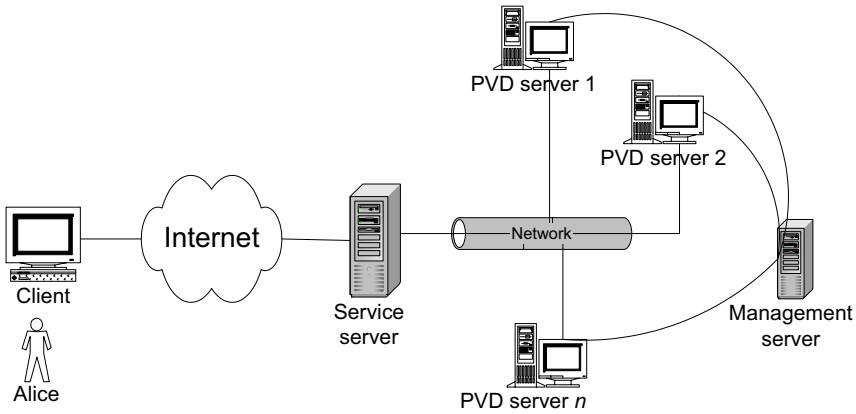

Figure 1. The intrusion-tolerant password authentication system

Well-Known Group 2 of the modular exponentiation groups (MODP) (given in Section E.2 of [25]), where N is a 1024 bit safe prime whose primality is rigorously proved. That is, $q = (N-1)/2$ is also a prime.

- The selection of g_1 and g_2. Note that in our system, g_1 and g_2 were not generators of F_N and instead, their orders are q. F_N has $(q-1)$ elements of order q. That is, almost half of the elements in F_N has order q. Thus, one can simply pick a random number in F_N and test if its order is q. By repeating this process a couple of times g_1 and g_2 can be found.

The management server also generates another parameter, $\beta = {g_1}^{\alpha} \bmod N$ where α is a random number in finite field F_q. A Shamir secret sharing is run on δ, $\delta = \alpha^{-1} \bmod q$, to generate shares $(\delta_1, \delta_2, \ldots, \delta_n)$ and δ_i, $1 \leq i \leq n$, is securely distributed to PVD server i[‡]. The public values $\beta_i = {g_1}^{\delta_i} \bmod N$, $1 \leq i \leq n$, are also distributed to the PVD servers. Moreover, each PVD server is also equipped with a 1024-bit RSA public/private key pair (for SSL connections).

The values (g_1, g_2, q, N, β) are hard coded into the client program which is also configured with the IP address of the service server. The service server is configured with the IP addresses of the PVD servers.

In the intrusion-tolerant password-based system, for each user, the system has two phases, the user enrollment phase and the user log-in phase. A user will first register himself with the management server in the user enrollment phase. The user log-in phase is used when an enrolled user wants to access the service server.

‡ α and β can also be generated distributively by the PVD servers, as described in [14], so that the overall values of α and δ never appear at any single location throughout their lifetime. In our implementation, α is generated by the management server and then shared among the PVD servers. The management server erases δ from its memory after its shares are distributed.

Enrollment and PVD Sharing. When a user registers himself with the management server, he picks a password p and a user ID, I. The management server picks a random value, s, as the salt and the user's PVD is computed as $x = H(s, I, p)$ where H is the secure hash algorithm. Then, a (t, n) Shamir secret sharing is performed on the PVD and PVD shares, $(x_1, x_2, \ldots, x_n)$, are generated. The values (I, s, x_i), $1 \leq i \leq n$, are securely sent (via the secure SSL connections between the management server and PVD servers) to PVD server i.

User Log-in. After a user is registered with the system, he can perform a distributed PAKE computation (described in Section 5) to log into the system using the client software in which (g_1, g_2, N, q, β) are hard coded. For this step, what a user needs is his password and the client software.

5. Data Flow

In this section we give the data flow details for the user log-in scenario, which is summarized in Table 1. Uninterested readers can skip the remainder of this section without losing the high-level view of the described system.

In the first step of the user log-in, the user types in his ID, I, and password p to the client program. The client program sends I to the service server, which forwards I to the available PVD servers. (Recall that only t PVD servers are required for a user authentication.) Using I as index, the participating PVD servers look up the user's salt s and PVD share x_i. Salt s is then sent to the service server, which forwards it to the client. In Step 2, the client computes $x = H(s, I, p)$. In Step 3, on the server side, with the help of a subset of the PVD servers, the service server computes B (see the following paragraph for details) and passes it to the client. At the same time, the client program generates a random number, a, in F_q and computes $A = {g_1}^{a} \bmod N$. In Step 4, the client computes S_c, W_1, W_2 using the given

formula (k is a random number in F_q) and sends them to the service server, which forwards them to the participating PVD servers. The service server uses the subset of PVD servers to compute S_s (see the following paragraph for details). Note that in step 4, each participating PVD server contributes to the value of S_s and S_s is combined only at the service server. At this point, the client has S_c and the service server has S_s. The client and the service server use steps 5, 6 and 7 to confirm that S_c and S_s are equal. In Step 5, the client computes M_1 and sends it to the service server, which checks M_1 against $M_1' = H(A, B, W_1, W_2, S_s)$. If $M_1 \neq M_1'$, the service server aborts the protocol. Otherwise, the service server computes M_2 and sends it to the client. The client program checks M_2 against $M_2' = H(A, M_1, S_c)$. The client program aborts the protocol if $M_2 \neq M_2'$. Otherwise, both sides are sure that they share a common value, S, $S = S_c = S_s$, and the session key K is derived as $K = H(S)$. K can be used to protect subsequent communication between the client and the service server.

The server-side computation of step 3. Let the participating PVD server set be $\Gamma \subseteq \{1, 2, \ldots, n\}$, $|\Gamma| \geq t$ (i.e., the size of Γ is not less than t). B is computed by the participating PVD servers as follows:

1. Each participating PVD server j generates a random b_j in F_q and computes $B_j = {g_1}^{b_j} \times {g_2}^{x_j \times \lambda_{j,\Gamma}} \bmod N$ where $\lambda_{j,\Gamma}$ are the Lagrange coefficients. PVD server j then sends B_j to the service server.
2. The service server computes B as $B = \prod_{i \in \Gamma} B_i \bmod N$ and sends B back to the client.

t or more PVD servers are needed to perform the above steps. We stress that PVD x is *not* reconstructed in this step.

The server-side computation of step 4. In this step, the participating servers compute $S_s = A^b \bmod N$ where $b = \sum_{i \in \Gamma} b_i$. The computation proceeds as follows:

1. Each participating PVD server j, $j \in \Gamma$, computes $S_j = A^{b_j} \bmod N$. Server j also picks a random k_j, $1 \leq k_j \leq q$, computes $(\sigma_j = {g_1}^{k_j}, \mu_j = \beta^{k_j} \times S_j) \bmod N$ and broadcasts them to all participating PVD servers.
2. Each participating PVD server j, $j \in \Gamma$, computes $\zeta = (\prod_{j \in \Gamma} \sigma_j)/W_1, \xi = (\prod_{j \in \Gamma} \mu_j)/W_2$. It then computes $\eta_j = \xi^{\delta_j}$ and broadcasts it. Besides η_j, PVD server j also generates a random r_j, computes $v_j = {g_1}^{r_j}$, $\omega_j = \xi^{r_j}$, $\tau_j = H(g_1, \xi, \beta_j, \eta_j, v_j, \omega_j)$, $z_j = \delta_j \times \tau_j + r_j$. $(v_j, \omega_j, \tau_j, z_j)$ allows other PVD servers to verify that η_j is indeed ξ^{δ_j} by checking if $\tau_j \stackrel{?}{=} H(g_1, \xi, \beta_j, \eta_j, {g_1}^{z_j} \beta_j^{-\tau_j}, \xi^{z_j} \eta_j^{-\tau_j})$.
3. Each participating PVD server j, $j \in \Gamma$, verifies the correctness of all η_i that it receives and then computes $\gamma = \prod_{i \in \Gamma} {\eta_i}^{\lambda_{i,\Gamma}}$ and checks if $\gamma \stackrel{?}{=} \zeta \bmod N$. If the above equation holds, it will securely send S_j to the service server. Otherwise, it aborts the protocol.
4. The service server computes $S_s = \prod_{i \in \Gamma} S_i \bmod N$.

We have the following observations about the above data flow:

- (W_1, W_2) is the ElGamal encryption of S_c. ElGamal encryption is multiplicative, which allows the participating PVD servers to verify that value A does come from a user who knows x before revealing $S_j = A^{b_j} \bmod N$. Technically, A could be g_1 (or some ${g_1}^c$ originated from an attacker who knows c)[§] and thus A^b would be ${g_1}^b$, which, if revealed to the attacker, could be used with $B = {g_1}^b \times {g_2}^x \bmod N$ (sent out in step 3) by the attacker to mount an off-line dictionary attack. To prevent this attack, in step 4 of the protocol, before revealing A^b, the participating PVD servers verify that (W_1, W_2) is the ElGamal encryption of A^b. Note that only the user who knows x can construct a valid set (A, W_1, W_2) where (W_1, W_2) is the encryption of A under β.
- $(v_j, \omega_j, \tau_j, z_j)$ prevents a (possibly corrupted) participating PVD server from cheating to bias the decision of $\gamma \stackrel{?}{=} \zeta \bmod N$.
- Compared to [29], x is shared among the PVD servers via a Shamir secret sharing, instead of a DDB94 secret sharing. As such, for each password, each PVD server is assigned only one PVD share and this reduces the management load. g_2 is used to prevent an active attacker who can spoof the service server from mounting off-line dictionary attacks.
- In the protocol, none of the S_c, password p and PVD x is exposed at any single PVD server.
- In the protocol, the client side authentication is accomplished through the use of ${g_2}^x$ and the server-side authentication is accomplished through β.
- A passive eavesdropper may be able to observe $(B = {g_1}^b \times {g_2}^x, A = {g_1}^a, W_1 = {g_1}^k, W_2 = \beta^k \times S_c)$. However, this does not give him/her any knowledge about the session key $S = {g_1}^{ab}$.
- For an attacker who wants to impersonate a user whose PVD is x, since the attacker does not know x, he will not be able to generate a valid set $(A = {g_1}^a, W_1, W_2)$, where (W_1, W_2) is the ElGamal encryption of ${g_1}^{ab}$ and the protocol will stop at Step 5.
- For an attacker who tries to spoof the service server, he cannot generate a value B in the form of ${g_1}^b \times {g_2}^x$ and at the same time knows b. In this way, the attacker

§ Say, an attacker has corrupted the service server and thus can manipulate the value of A.

	Client		Service Server		PVD Server 1	...	PVD Server n
1		$\xrightarrow{I}$		$\xrightarrow{I}$	(lookup s, x_1)		(lookup s, x_n)
2	$x = H(s, I, p)$	$\xleftarrow{s}$		$\xleftarrow{s}$			
3					**Collectively** compute		
	$A = {g_1}^a \bmod N$	$\xleftarrow{B}$		$\xleftarrow{B}$	$B = {g_1}^b \times {g_2}^x \bmod N$		
4	$S_c = (B/{g_2}^x)^a \bmod N$ $W_1 = {g_1}^k \bmod N$ $W_2 = \beta^k \times S_c \bmod N$	$\xrightarrow{A,W_1,W_2}$		$\xrightarrow{A,W_1,W_2}$			
					Collectively compute		
				$\xleftarrow{S_s}$	$S_s = A^b \bmod N$		
5	$M_1 = H(A, B, W_1, W_2, S_c)$	$\xrightarrow{M_1}$	(verify M_1)				
6	(verify M_2)	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, S_s)$				
7	$K = H(S_c)$		$K = H(S_s)$				

Table 1. The Data Flow

will not be able to generate a valid message M_2 and as such the protocol halts at Step 6.

- If the service server is corrupted, the current session is compromised. However, from the compromised session key $S_s = {g_1}^{ab}$, the attacker will get no knowledge about the corresponding password p.
- If an attacker compromises up to $(t-1)$ PVD servers, he will not be able to mount off-line dictionary attacks.

6. Implementation Details and Performance

In this section, we will give some implementation details and performance data about our system.

6.1. Some details

All the client program, the service server interface, the PVD servers and the management server are implemented in `Java` language and wrapped in the `sci.crypt.pake` package. The SSL connections are implemented through the IAIK-iSaSiLk toolkit, a SSL package by IAIK. (More details about IAIK-iSasiLk can be found at http://jce.iaik.tugraz.at/products/index.php.) To improve the performance, SSL caching is turned on. The 1024-bit RSA private keys stored on the PVD servers, used for the SSL connections, are stored as `PKCS12` soft tokens and are protected by passwords.

6.2. Performance

We ran some tests on the described system and got some indicative results. Our testing environment comprised several PCs with Intel Pentium 4 CPUs. These PCs ran various types of Microsoft Windows operating systems including Windows XP and Windows 2000 Professional and were connected by a 100Mbps local area network, which was located in one university's campus network. The service server and the multiple PVD servers were running on these PCs.

	Test over the Internet		Test over LAN	
(t, n)	$(2, 4)$	$(3, 4)$	$(2, 4)$	$(3, 4)$
Log-in Time (seconds)	4.313	5.683	2.697	4.216

Table 2. Sample Performance

We tested the system in two cases. In the first case, the client program was running on a Sun workstation with Ultra-60 CPU and Sun OS 5.9 operation system, which was located in another university's campus network (different from the service/PVD servers). These two university campus networks were connected through the Internet and there were 8 intermediate routers between the client program and the service server. In the second case, the client program was running on a PC located in the same local area network as the service/PVD servers. Some example configuration files for our tests are given in the Appendix.

In these experiments, we used `itpake` as the user name and `a1b2c3d4` as the password. (In [31], this password passed the proactive password checking but still fell prey to the dictionary attack.) Table 2 gives the performance of the system for the above two test situations, each with (t, n) as $(2, 4)$ and $(3, 4)$. Table 2 indicates that the intrusion tolerance architecture incurs some performance penalty but the system still performs reasonably well. On the other hand, significant security improvement has been achieved.

6.3. Further discussions

We observed that, among the system components, the computation on the PVD servers are the most computation intensive. The system achieved best performance when PVD servers were running on dedicated fast computers. On the other hand, the computation by the service server interface is rather light, which is attractive as the integration of the service server interface to a service server won't affect the performance of the service server much.

We observe that the system performance can be further improved by the following methods.

- Precomputation. From Section 5, we note that the server-side value B in step 2 is independent of user log-in attempt and thus, can be precomputed. This will improve the system performance by about one-sixth.
- When t is small (say, t is 3 or 4), through some storage, the computation of γ in Step 4.3 can be performed in a single modular exponentiation.
- Caching mechanism can be implemented by the client program and the service server interface to quickly resume an old but still valid session, as done in SSL.

7. Operational Considerations

Using multiple servers for system intrusion tolerance implicitly assumes that breaking into multiple machines is more difficult than compromising just one. To meet this assumption, extra cautions must be exerted in the system deployment. In this section, we will discuss several operational issues.

Sharing of the working folder The working folder where PVD shares are stored should not be shared to the network. Otherwise, by breaking into just a few machines (just one machine is needed if t PVD shares are accessible from that machine via network-shared folders), an attacker will be able to steal the PVD shares, recover the shared PVD and mount off-line dictionary attacks.

PVD share update Since PVD x is shared via a Shamir secret sharing, t or more PVD servers can collectively update their PVD shares while keeping the shared PVD (x) unchanged. This will add proactive security to the system [16, 6] and make the system more resilient.

Adding a new PVD server For a multiple-server system, it is natural that the system administrator may decide to retire one PVD server or add a new one. Since t or more PVD servers can help a new PVD server generate its PVD share, the administrative process of adding a new server can be significantly simplified.

Password change For a password authentication system, it is desirable that a user can change his password. For our intrusion-tolerant system, the password change proceeds as follows: the client performs a PAKE log-in with each participating PVD server (as opposed to the service server in the normal user log-in) and establishes a secure channel. It then sends the PVD shares of the new password via these secure channels to the participating PVD servers.

The management server In the described system, the management server is used to generate system parameters and user management. It can be used to reset a user's password but it is not involved in user log-in and user password change. For security reason, the management server should typically stay off-line and can be brought on-line when used.

8. Example Applications

Our described system can be straightforwardly integrated to the `Telnet` and `FTP` to enhance their security. It can also be integrated into password-based web applications. For web applications, the PVD servers and the management server can be used without any changes. The client program and the service server interface can exist in many forms.

- The client program. To support web browser on the client side, the client program can be in the form of Java Applet.
- The service server interface. Our service server interface is written in `Java` and can be easily integrated into JSP pages.

Our system can also be extended to store password-protected credentials [13, 19, 26, 22]. These credentials can be accessed by a remote entity authenticated by his password and the user can roam with his password only. For example, a user's private key (of a public/private key pair) can be encrypted by his password and the encrypted private key can be shared among multiple servers [28] — each server has a share of the password-encrypted private key. The mobile user can retrieve his private key from any new location by authenticating himself to a threshold of the multiple servers via his password, establishing secure connections with the participating servers and downloading the password-encrypted private key shares. The password-encrypted private key can be reconstructed and decrypted for use. To perform the above steps, a user only needs his password and a reliable client program. For this type of applications, some changes are necessary to integrate our system. The service server interface should be combined into each PVD server and the client program should be configured with the IP addresses of the PVD servers. For perfor-

mance reason, the data flow described in Section 5 can be optimized (details are omitted here due to space constraint).

9. Conclusion

Password-verification data is a value derived from the corresponding password using publicly known functions and is typically stored by a single server to authenticate the user. Previous research has shown that in a password authentication system, even when a proactive password checker is used, some passwords are still vulnerable to the dictionary attack. For these passwords, if the centralized server is compromised and their PVD is stolen, the attacker can mount off-line dictionary attacks against them. In this article, we described an intrusion-tolerant password authentication system, which uses multiple servers to share PVD and never reconstructs them during user authentications. Compromising up to $(t-1)$ such servers will not allow an attacker to mount off-line dictionary attacks and the system can still function in the presence of some server failures. We gave the system architecture and some implementation details. Our experimental results showed that the intrusion-tolerant architecture achieves high-level security at reasonable expense.

References

[1] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 2000.

[2] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73, Fairfax, Virginia, United States, 1993.

[3] S. Bellovin and M. Merritt. Encrypted key exchange: password–based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84, 1992.

[4] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: a password–based protocol secure against dictionary attacks and password file compromise. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 244–250, 1993.

[5] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 156–171, Bruges, Belgium, May 2000.

[6] R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: Long-term protection against break-ins. *CryptoBytes*, 1(3):1–8, 1997.

[7] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited (preliminary version). In *Proceedings of the 30th annual ACM symposium on Theory of computing*, pages 209–218, Dallas, Texas, United States, 1998.

[8] CERT. CERT advisory CA-2002-29 buffer overflow in Kerberos administration daemon. Available at http://www.cert.org/advisories/CA-2002-29.html, October 25 2002.

[9] CERT. Vulnerability note VU#124003 apache HTTP server on Win32 systems does not securely handle input passed to CGI programs. Available at http://www.kb.cert.org/vuls/id/124003, April 11 2002.

[10] M. Di Raimondo and R. Gennaro. Provably secure threshold password-authenticated key exchange. In E. Biham, editor, *Advances in Cryptology - EUROCRPYT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 507–523, May 4–8 2003.

[11] T. Dierks and C. Allen. The TLS protocol version 1.0. Internet RFC 2246, January 1999.

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[13] W. Ford and B. Kaliski, Jr. Server–assisted generation of a strong secret from a password. In *Proceedings of the 9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises(WET ICE 2000)*, pages 176–180, Gaithersburg, MD, USA, June 14–16 2000.

[14] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT99*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310, 1999.

[15] L. Gong. Increasing availability and security of an authentication service. *IEEE Journal on Selected Areas in Communications*, 11(5):657–662, June 1993.

[16] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing. In D. Coppersmith, editor, *Advances in Cryptology — Crypto '95*, volume 963 of *Lecture Notes in Computer Science*, pages 339–352, Santa Barbara, California, U.S.A., August 27–31 1995. Springer-Verlag.

[17] D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *Proceedings of the 6th IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 248–255, June 18-20 1997.

[18] D. P. Jablon. Strong password–only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, October 1996.

[19] D. P. Jablon. Password authentication using multiple servers. In D. Naccache, editor, *Progress in Cryptology - CT-RSA 2001 Proceedings of the Cryptographers' Track at RSA Conference*, volume 2020 of *Lecture Notes in Computer Science*, pages 344–360, San Francisco, CA, USA, April 8–12 2001. Springer-Verlag.

[20] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In B. Pfitzann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 475–494. Springer-Verlag, 2001.

[21] J. Klensin, R. Catoe, and P. Krumviede. IMAP/POP authorize extension for simple challenge/response. Internet RFC 2095, January 1997.

[22] T. Kwon. Virtual software tokens - a practical way to secure PKI roaming. In G. Davida, Y. Frankel, and O. Rees, editors, *Proceedings of the Infrastructure Security (InfraSec)*, volume 2437 of *Lecture Notes in Computer Science*, pages 288–302. Springer-Verlag, 2002.

[23] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In T. Okamoto, editor, *ASIACRYPT2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 599–613. Springer-Verlag, 2000.

[24] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold password-authenticated key exchange (extended abstract). In M. Yung, editor, *Advanced in Crypto: – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 385–400. Springer-Verlag, August 2002.

[25] H. Orman. The OAKLEY key determination protocol. Internet RFC 2412, November 1998.

[26] R. Perlman and C. Kaufman. Secure password-based protocol for downloading a private key. In *Proceedings of the ISOC Network and Distributed Systems Security Symposium*, 1999.

[27] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signature and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[28] X. Wang. Intrusion-tolerant password-enabled PKI. In *Proceedings of the 2nd Annual PKI Research Workshop*, pages 44–53, Gaithersburg, MD, USA, April 28–29 2003.

[29] X. Wang and S. Redwine. An efficient threshold password-authenticated key exchange protocol. In *Proceedings of the 2003 International Conference on Distributed Multimedia Systems (DMS'2003)*, 2003.

[30] T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Network and Distributed System Security Symposium*, pages 97–111, 1998.

[31] T. Wu. A real-world analysis of Kerberos password security. In *Proceedings of the 1999 Network and Distributed System Security Symposium*, 1999.

Appendix

In this appendix, we give some example configuration files for our described system.

Example configuration for the server service interface

```
ServiceServerPort=8000
Server.SSL.1=http://129.174.87.243:8001
Server.SSL.2=http://134.126.21.129:8002
Server.SSL.3=http://134.126.21.157:8003
Server.SSL.4=http://134.126.24.122:8004
TrustedCertFile=DemoCA.cer
```

Example configuration file for PVD server 4

```
Server.SSL.1=http://129.174.87.243:8001
Server.SSL.2=http://134.126.21.129:8002
Server.SSL.3=http://134.126.21.157:8003
Server.SSL.4=http://134.126.24.122:8004
ServerID=4
SSLTokenFile=PVDToken4.p12
PVDShareFile=PVDShareFile.4
DeltaInverseShare=4754EF815B9 (snipped)
BetaShares.1=00F0130E08D4A58E (snipped)
BetaShares.2=2B4CAF7C5DBB60B4 (snipped)
BetaShares.3=41AECDD4E945D5DF (snipped)
BetaShares.4=0B6C51D6634856CA (snipped)
```

Example PVD share record on PVD server 4 for user "itpake"

```
itpake:D370:3:4:4:4F6A4A86B3AB0EB72BB1C
BEC1653A9D1DBDC3874A34703D956580A95BCF8
335187786250111E36B149AE72346AF00E5EA39
FED5691FE7E8CA3A50E487F9FD5F64FDE1D7CC2
4B125A4FF5773CAED89F4B3D9C632F69F310C75
003D38C5C229A69B5FB04B71E1F7C90BDDCC641
AE61F1F943A86B82645662CB79394B2DB85784E2
```