Securing Passwords Against Dictionary Attacks*

Benny Pinkas†
HP Labs
benny.pinkas@hp.com

Tomas Sander
HP Labs
tomas.sander@hp.com

ABSTRACT

The use of passwords is a major point of vulnerability in computer security, as passwords are often easy to guess by automated programs running dictionary attacks. Passwords remain the most widely used authentication method despite their well-known security weaknesses. User authentication is clearly a practical problem. From the perspective of a service provider this problem needs to be solved within real-world constraints such as the available hardware and software infrastructures. From a user's perspective user-friendliness is a key requirement.

In this paper we suggest a novel authentication scheme that preserves the advantages of conventional password authentication, while simultaneously raising the costs of online dictionary attacks by orders of magnitude. The proposed scheme is easy to implement and overcomes some of the difficulties of previously suggested methods of improving the security of user authentication schemes.

Our key idea is to efficiently combine traditional password authentication with a challenge that is very easy to answer by human users, but is (almost) infeasible for automated programs attempting to run dictionary attacks. This is done without affecting the usability of the system. The proposed scheme also provides better protection against denial of service attacks against user accounts.

Categories and Subject Descriptors

K.6.5 [**Computing Milieux**]: Management of computing and information systems—*Security and Protection*

General Terms

Security

*Most of this work was done while the authors were with STAR Lab, Intertrust Technologies.

†Part of this work was done while the author was with DIMACS, Rutgers University.

CCS'02, November 18–22, 2002, Washington, DC, USA.

1. INTRODUCTION

Passwords are the most common method of authenticating users, and will most likely continue to be widely used for the foreseeable future, due to their convenience and practicality for service providers and end-users. Although more secure authentication schemes have been suggested in the past, e.g., using smartcards or public key cryptography, none of them has been in widespread use in the consumer market. It is a well known problem in computer security that human chosen passwords are inherently insecure since a large fraction of the users chooses passwords that come from a small domain (see, e.g., the empirical studies in [19, 15]). A small password domain enables adversaries to attempt to login to accounts by trying all possible passwords, until they find the correct one. This attack is known as a "dictionary attack". Successful dictionary attacks have, e.g., been recently reported against eBay user accounts, where attackers broke into accounts of sellers with good reputations in order to conduct fraudulent auctions [8].

When trying to improve the security of password based authentication, one wants to prevent attackers from eavesdropping on passwords in transit, and from mounting *offline* dictionary attacks, namely attacks that enable the attacker to check all possible passwords without requiring any feedback from the server. Eavesdropping attacks can be prevented by encrypting the communication between the user and the server, for example using SSL (see also [12, 5, 16, 11]). Offline dictionary attacks are prevented by limiting access to the password file (and can be made even harder by adding well-known measures such as the use of salt).

In our discussion here we assume that the security measures described above are already implemented and therefore the attacker can only mount *online* dictionary attacks. Namely, attacks where the only way for the attacker to verify whether a password is correct is by *interacting* with the login server. This might be a reasonable assumption for an Internet based scenario, where SSL is used to encrypt passwords and the server uses reasonable security measures to secure its password file. In Section 1.1 we show that it is hard to provide security even against this limited type of attack. We then describe new protocols designed against online dictionary attacks.

1.1 Common countermeasures against online dictionary attacks

There are two common countermeasures that are taken against online dictionary attacks:

Delayed response: Given a login-name/password pair

the server provides a slightly delayed yes/no answer (say not faster than one answer per second). This should prevent an attacker from checking sufficiently many passwords in a reasonable time.

Account locking: Accounts are locked a few unsuccessful login attempts (for example, an account is locked for an hour after five unsuccessful attempts.) Like the previous measure, this measure is designed to prevent attackers from checking sufficiently many passwords in a reasonable time.

These countermeasures can be quite useful in a single computer environment, where users login to a local machine using, say, a keyboard that is physically attached to it. We describe below how both these measures are insufficient in a network environment. Furthermore, we show that implementing the account locking measure is very costly and introduces additional risks.

1.2 Weaknesses of the countermeasures

1.2.1 Global password attacks

Consider a system that has many user accounts, and which enables logins over a network that is accessible to hackers. (Again, we do not assume that the attackers can sniff network traffic, we only assume that they can connect to the network and try to login to the server, pretending to be a legitimate user.)

Consider an attacker that is interested in breaking into *any* account in the system, rather than targeting a specific account. The attacker can try many login attempts in parallel and circumvent the timing measure using the fact that user logins are typically handled by servers that can handle many login sessions in parallel. For example, the attacker can send a login attempt every 10 milliseconds, obtaining a throughput of 100 login attempts per second, regardless of how long the server delays the answers to the login attempts.

The account locking feature can also be circumvented by such a "global" attacker, if it tries to login using different username/password pairs, and operates without trying the same user name twice. Since every user name is used only once, the "account with many failed login attempts" alarm is never triggered.[1]

1.2.2 Risks of the account locking measure

The account locking security measure introduces additional risks, which can actually make a strong case against using it in practice. (Indeed, as was suggested in a recent news article [8], eBay does not implement this measure for these reasons).

[1]This attack is further simplified by the fact that in many practical scenarios an attacker can easily get hold of a large database of valid user names. A list of usernames is often known in interactive web communities, such as auction sites. In many large corporations user names equal the email handle and can just be grabbed from corporate web sites or employee lists. For large Internet services (e.g., MyYahoo) almost any reasonable user name has already been taken, so that valid user names are trivial to guess for an attacker. The latter statement applies to any Internet service with a huge user base, e.g. to identity services such as Microsoft Passport. Furthermore, a valid bank account number is often also easy to generate, as only part of it is random. (Parts of the account number are used to identify the branch, and one digit is used for a checksum.) Thus it is relatively easy to generate valid user names for Internet banking services that use account numbers as user names.

Denial of service attacks. The "account locking" feature enables denial of service attacks against users. These attacks are mounted by trying to login several times to a user's account with invalid passwords, thus causing this account to be blocked. Yahoo!, for example, report that users who compete in auctions use these methods to block the accounts of other users who compete in the same auctions. This attack should be especially worrisome to mission critical applications, for example to enterprises whose employees and customers use the web to login to their accounts.

One could even imagine a *distributed denial of service attack* against servers that implement the "account locking" feature. Similar to other distributed denial of service attacks (DDoS), the attacker could plant hidden agents around the web. All the agents could start operating at a specific time, trying to login into accounts in a specific server using random passwords (or using a dictionary attack). This attack could block virtually a large proportion of the accounts of the attacked server.

Customer service costs. Another major drawback of the "account locking" feature is that it causes user accounts to be locked, either by mistake (e.g. by users that do not type their passwords correctly) or as a result of dictionary attacks. The service provider must therefore operate customer service centers to handle calls from users whose accounts are locked. (Even if the service provider installs mechanisms for automated unlocking of accounts, e.g. using private questions whose answers are known to the legitimate users, it is still needed to operate a customer service center for those users who fail to use these automated mechanisms.) The cost of running these centers is high, and is estimated to cost more than $25 per customer call. Imagine that each user locks his account once every five years, then the service cost, per user, per year, is at least $5. (An online auction industry observer commented [8], that in her opinion eBay had not implemented account locking features due to the costs of operating customer support centers.)

1.3 Pricing via processing

The "pricing via processing" paradigm was introduced by Dwork and Naor in the context of filtering junk email [9]. This paradigm requires that any attempt to use a resource requires the party that makes the attempt to send a proof that it invested some non-trivial computation time in constructing its request. The required computational time is negligible per usage attempt, but becomes a real barrier if one makes a large number of attempts. As a specific example in the context of preventing dictionary attacks, the server could require that a login attempt is accompanied by a value x that satisfies the requirement, say, that the last 20 bits of $H(x,\texttt{username,password,time-of-day})$ are all 0, where H is a hash function such as SHA. If we assume that SHA behaves as a random function, then the attacker would need to check on the average 2^{19} values for x before it finds a value that satisfies the test. that The computation of x adds a relatively negligible overhead to a single login attempt, but can significantly slow down the operation of a dictionary attack.

One drawback of this solution is that the user's client must be able to run a program that performs the computation. This either requires the use of a special client, or the download of mobile code (e.g. a Java or JavaScript program). In

the latter case we could not hope to have a universal coverage of all users since not all browsers or environments (e.g. corporate firewalls) support mobile code. Another drawback of this solution is that a legitimate user could be running a slow machine, whereas the attacker could be using a very powerful machine. Since the challenge should not be too demanding for the user, a dictionary attack might only be relatively hard for the attacker.

Our approach. Our solution follows the "pricing via processing" paradigm. We observe that a legitimate login attempt is done by a *human* user, whereas a dictionary attack is done by an automated program. We therefore require that any login attempt be accompanied by the product of a "computation" that is very easy for humans, but is hard for machines.

We identify in Section 5 several of the practical requirements for authentication protocols in e-commerce applications and argue that our protocol scores very well with respect to these requirements. In particular the protocol has almost no impact on usability and requires almost no change in user behavior. The authentication method is portable, i.e. it can be used from a number of different machines. It does not require the use of additional hardware, or downloading software. It is easy to implement for service providers and can be easily integrated with existing authentication technology. The protocol allows service providers to deal with denial of service attacks more effectively by removing the need to lock accounts after a few unsuccessful login attempts. It thus overcomes most of the practical shortcomings of previous suggestions to strengthen user authentication (which we review in Section 5). The added security of our scheme relies on the security of tests that distinguish between human users and machines. Some recent results (that we will describe in more detail in Section 2) suggest that these tests might indeed be sufficiently strong, although this assumption certainly requires further careful testing.

These properties and the low costs of implementing our protocol (especially for service providers) suggest that it has a realistic chance of being deployed in electronic commerce (and other) applications.

The paper is organized as follows. In Section 2 we describe (known) tests that allow to distinguish between human users and machines and some of their properties. In Section 3 we describe our new authentication protocols. In Section 4 we analyze their security. In Section 5 we describe related work and review authentication protocols with respect to real world requirements.

2. TOOLS – REVERSE TURING TESTS

Our protocols may use one of the several tests that attempt to distinguish between a human user and a computer program. These tests should be easy for human users to pass, yet be hard for automated programs. They were first suggested by Naor [20], and were denoted as reverse Turing tests (RTTs). Alternative names for these tests are CAPTCHAs [6], and mandatory human participation protocols [22].

A reverse Turing test should satisfy the following requirements:

- *Automated generation:* It should be easy to generate many instances of the test in an automated way.
- *Easy for humans:* A generated test should be easy for human users to solve.
- *Hard for machines:* An automated adversary is a program that cannot interact with a human user after receiving its input. We require that any automated adversary cannot answer the test correctly with probability that is significantly better than guessing. The input of the adversary can be of two types: (1) It can include a complete description of the algorithm that generates the RTTs. In this case the adversary can generate by itself many instances of the RTT together with their solutions. (2) A weaker notion of security is against adversaries that receive up to m examples of RTT instances and their solutions, where m is a parameter.
- *Small probability of guessing the answer correctly:* We require that the probability that a random guess produces a correct answer to the test is small. For example, a test that presents an image of a person and asks the viewer to type whether this is a male or a female is insufficient for our purpose, since a random guess of the answer is correct with probability of at least 50%. An example of a test that satisfies this requirement is one that displays a random 6 character string rendered in a way that is hard for OCR programs to decode, and asks the user to type the string.

Recent research efforts in this area include a patent by Lillibridge et al [14], a paper by J. Xu et al [22], and a workshop at Xerox PARC [25] sponsored by the CAPTCHA project at CMU [6]. The CAPTCHA web site includes examples of RTTs, as well as source code. Reverse Turing tests are currently used in commercial deployments: Yahoo!, AltaVista, and PayPal use them to ensure that only human users sign up for certain of their services [26, 1, 21]. A typical implementation of such a test presents the user with a distorted text image, presumably one that cannot be parsed by current OCR (Optical Character Recognition) programs, and requires the subject of the test to type the characters of the image (the encoding of the image is typically 6K to 30K bytes long).

2.1 Security of RTTs

Of course, it is not clear whether a true reverse Turing test (RTT) exists at all, and the success of any real-world deployment of a proposed RTT depends on a moving target: the current state of the art in designing programs to carry out the (presumably) difficult human task on which the test is based, as well as reductions in the cost of computational resources due to Moore's law. There are, however, good indications that RTTs that are based on images of distorted strings are rather secure, at least for our application:

- Alta Vista use RTTs in the following way. Users that submit URLs to the Alta Vista search engine are required to pass an RTT before the URL is accepted [1]. This requirement was added after it was realized that web spammers were using automated programs to submit many URLs, in order to increase the ranking of their web pages. Alta Vista report that the number of submitted URLs was reduced by 90% after the RTT test was added to the URL submission process.

- A different indication is presented in a recent paper [2], written by experts in OCR technology. The conclusion of that paper is optimistic with regard to the future of RTTs, suggesting that "The slow pace of evolution of OCR... suggests that pessimal print will defy automated attacks for many years".
- Furthermore, in our paper we design solutions for dictionary attacks that use RTTs. These constructions ensure that in order to guess a password correctly the attacker must pass *many* RTTs. We therefore do not have to require that solving the RTTs is infeasible for automated adversaries. The constructions are secure as long as the task of breaking RTTs requires some considerable computational investment from the adversary.

A note on accessibility. A possible issue with the current implementations of RTTs is that they are based on visual capabilities of humans. Solving them might prove hard for people with vision problems, or with other disabilities. Any attempt to design a solution that will be widely used must address this issue. A possible solution is to enable visually impaired users to use audible RTTs that ask users to type back a sequence of letters that is read over a noisy background. In fact, PayPal enables users to choose this type of RTT instead of the default visual test.

3. USER AUTHENTICATION PROTOCOLS

We first describe a basic protocol that provides good security against dictionary attacks but affects the usability and scalability of the system. We then describe a protocol that solves the usability and scalability issues.

3.1 A basic protocol

The basic protocol combines reverse Turing tests (RTTs) with any password based user authentication system in the following way:

> Users are required to pass an RTT before starting the authentication process (namely, before entering their username and password).

This solution prevents automated programs from trying many passwords, since each password guessing attempt requires the adversary to answer an RTT. A detailed discussion of the security appears in Section 4. Intuitively, any attack against the original login scheme that requires N login attempts, now requires the adversary to answer N RTTs. This either means that the adversary should perform N interactions with human users that answer the RTTs, in which case the operation of the attacker is definitely slowed down, or requires the adversary to design an automated program that breaks the RTTs. Furthermore, even if the adversary designs such an automated RTT solver, it would still slow down the operation of the dictionary attack, unless it can break RTTs in negligible time[2].

In spite of its improved security, this method is not optimal for large-scale systems, for two major reasons.

Usability: For a system to be widely deployed, it should provide users with a user experience that is as similar as possible to their existing experience. Currently the user's login procedure is a fairly automatic mental task: the user types the same username and password in every login attempt. A system that asks users to pass an RTT before entering their username and password requires them to pass a much more demanding task than the one required by the current procedure. This would probably annoy many users.

Scalability: The solution requires an RTT to be generated for every login attempt. Current systems require users to pass an RTT when they register for a new service (for example in Yahoo! or Paypal), and this happens much less frequently than login attempts. It is not clear whether RTT generation can scale up easily to generate and serve an RTT per login attempt.

Comment: In order to solve the scalability problem, a protocol could require users to answer an RTT only for a fraction of their login attempts. The drawback of this approach is that the decision whether to require an RTT is performed *before* the user enters the username and password. An adversary that mounts a dictionary attack can choose to continue with the login process if it is not required to pass an RTT, and abort it if an RTT is required. If RTTs are required for a fraction p of the login attempts, the adversary's operation is slowed down by a factor of $1/(1-p)$, which is a small factor unless p is very close to 1 . We show below how to achieve scalability while requiring the attacker to solve a large number of RTTs.

Another naive protocol operates by letting users login to their accounts without solving an RTT, except for the following condition: if a *wrong* password is entered for an account then all subsequent login attempts to this account require the user to pass an RTT. This requirement is in effect until a successful login is performed to this account. This method does not require an adversary to pass an RTT for the first attempt of testing a password for an account, but does require it to solve RTTs for each additional login attempt to this particular account. This protocol provides reasonable protection against an adversary that tries to break into an individual account. However the protocol is flawed as it provides little security against a *global* password attack. An adversary that is interested in breaking into any account from a large user base could still test a single password for every username without ever having to pass an RTT, and would likely succeed if a (significant) fraction of the users have chosen weak passwords (e.g. if there are 100,000 users and 50% of them choose passwords from a small dictionary of 10,000 words). Furthermore, when the legitimate users login to their accounts after a login attempt was made by the adversary, they solve the RTT and enable the adversary to test an additional password per account for every successful login (although the users can identify that an unsuccessful login attempt was performed, it would still be hard for the system administrator to distinguish between the legitimate and the illegitimate login attempts).

3.2 A protocol answering the usability and scalability issues

A major observation that substantially improves the protocol is that each user typically uses a limited set of computers, and that it is unlikely that a dictionary attack would be mounted against this user's account from one of these computers. This observation can be used to answer the scalability and usability issues, while reducing security by only

[2]In this paper we refer to the adversary as "it" in order to emphasize that we assume it to be an *automated* adversary.

a negligible amount. Another helpful observation is that it would be hard for the adversary to attack accounts even if it is required to answer RTTs for only a fraction of its login attempts (and we show below how to do this without running into the problem pointed out in the comment above). This enables a great improvement in the scalability of the system.

The protocol assumes that the server has a reliable way of identifying computers. For example, in a web based system the server can identify a web browser using cookies (see [10] for a discussion of how to use cookies securely in the context of user authentication). Other identification measures could be, for example, based on network addresses or special client programs used by the user. To simplify the exposition we assume that the login process uses a web browser, and that cookies are enabled. (Our protocol can handle cookie theft, as we describe below.) The protocol is described in Figure 1. We first describe the usability and scalability effects of this protocol, and then discuss its security.

User login protocol

Initialization: Once the user has successfully logged into an account, the server places in the user's computer a cookie that contains an authenticated record of the username, and possibly an expiration date. ("Authenticated" means that no party except for the server is able to change the cookie data without being detected by the server. This can be ensured, for example, by adding a MAC that is computed using a key known only to the server. See [10] for details.) Cookies of this type can be stored in several computers, as long as each of them was used by the user.

Login:

1. The user enters a username and a password. If his computer contains a cookie stored by the login server then the cookie is retrieved by the server.
2. The server checks whether the username is valid and whether the password is correct for this username.
3. **If the username/password pair is correct**, then
   (a) If the cookie is correctly authenticated and has not yet expired, and the user identification record stored in the cookie agrees with the entered username, then the user is granted access to the server.
   (b) Otherwise (there is no cookie, or the cookie is not authenticated, or the user identification in the cookie does not agree with the entered username) the server generates an RTT and sends it to the user. The user is granted access to the server only if he answers the RTT correctly.
4. **If the username/password pair is incorrect**, then
   (a) With probability p (where $0 < p \leq 1$ is a system parameter, say $p = 0.05$), the user is asked to answer an RTT. When his answer is received he is denied access to the server, regardless of whether it is correct or not.
   (b) With probability $1 - p$, the user is immediately denied access to the server.

   Comment: The decision whether or not to serve an RTT must be a *deterministic* function of the entered username/password pair. That is, for any specific pair of username and password values, the user is either always asked to pass an RTT, or is never asked to answer one.

Figure 1: The login protocol

3.2.1 Usability

The user is experiencing almost the same interaction as in the original login protocol (that uses no RTTs). The only difference is that he is required to pass an RTT in two instances (1) when he tries to login from a new computer for the first time, and (2) when he enters a wrong password (with probability p). We assume that most users are likely to use a small set of computers to access their accounts, and use other computers very infrequently (say, while traveling without a laptop). We also have evidence, from the use of RTTs in Yahoo!, Alta Vista and Paypal, that users are willing to answer RTTs if they are not asked to do so very frequently. Based on these observations we argue that the suggested protocol could be implemented without substantially downgrading the usability of the login system.

3.2.2 Scalability and the operation of the server

The main difference in the operation of the server, compared to a login protocol that does not use RTTs, is that it has to generate RTTs whenever they are required by the protocol. To simplify the analysis assume that most users use a small set of computers, and very seldom use a computer that they haven't used before. We can therefore estimate that the server is required to generate RTTs only for a fraction p of the *unsuccessful* login attempts. This is a great improvement compared to the basic protocol of Section 3.1, which requires the server to generate an RTT for *every* login attempt. Note that the overhead decreases as p is set to a smaller value.

As for the cost of adding the new protocol to an existing system, note that no changes need to be made to the existing procedure, where the authentication module receives a username/password pair and decides whether they correspond to a legitimate user. The new protocol can use this procedure as a subroutine, together with additional code that decides whether to require the user to pass an RTT, and whether to accept the user as legitimate.

Caching: The security of the protocol is not affected if the same RTT is given every time that the same username and password are provided. That is, the RTT itself could be a deterministic function of the username and password that are entered by the user. The server can then, for example, cache, for every username, the RTT that is required when the correct password is entered. When the user tries to login (from a computer that does not contain his identifying cookie), and the server recognizes that the correct password was entered, it can retrieve the cached RTT, and send it to the user. This change is preferable from both the scalability and usability perspectives. Scalability is improved since the server does not have to generate an RTT every time the user logs into the account from a new machine. Assuming that most of the times the user uses the same set of machines and enters his correct password, the server is only required to generate RTTs for about a fraction p of the login attempts of the dictionary attack. Usability is improved since the user is always presented with the same RTT. We can expect that the task of solving the RTT will become pretty automatic, and therefore the login experience would not be very different from the current one, even when the user uses new computers.

Finally we sketch a variant of the scheme, in which $p = 1$, i.e., for every failed login attempt the user is served an RTT. In this case the decision whether to serve an RTT no longer

depends on the entered username/password pair. Thus the protocol can be simplified from a two-round to a one-round protocol, if the server has already retrieved the relevant cookie from the user before the login screen is displayed to the user. In this case a user who cannot present the required cookie is given a login screen that contains fields for entering his username/password pair and a randomly generated RTT challenge. This variant seems particularly easy to integrate with an existing authentication module.

4. SECURITY ANALYSIS

4.1 User-server interaction

The implementation of the protocol should be carefully designed to satisfy the following requirement:

IMPLEMENTATION REQUIREMENT 1. *A user that enters a username/password pair and does not present a valid cookie receives one of the following two feedbacks from the server:*

- *Feedback 0: "The username/password pair is invalid.*
- *Feedback 1: "Please answer an RTT" (and then you will be told whether the username/password pair you entered is correct)*

The choice of the feedback must be a deterministic function of the username/password pair, and divides the set of all such pairs into exactly two subsets, S_0, S_1. The feedback that the user receives must be the same for all pairs in the subset S_1 (regardless of whether the pair is valid or not).

The requirement implies that the decision whether or not to require an RTT in the case that a wrong password is entered, must be a *deterministic* function of the username and password. Namely, for a fraction p of the wrong passwords an RTT is always required, and for a $1-p$ fraction of the wrong passwords an RTT is never required. If this was not the case, then an adversary could have simply tried each password several times (e.g. c times). If this is the correct password then the adversary is required to pass an RTT in all attempts, whereas if the password is wrong the probability that an RTT is required in all attempts is p^c, which can be made arbitrarily small.

Another aspect of the implementation which requires careful analysis, is that the delay between the time that the password is entered and the time that the RTT is served should be the same whether the password is correct or not. (Extra care should be taken to make sure that this requirement is satisfied when the RTT that corresponds to the correct password is cached.) If this requirement is not satisfied then the attacker could use a timing attack to decide whether the password entered is correct or not.

4.2 Analysis for a single account

The main feature of the protocol that is relevant for the security analysis is that for a p fraction of the incorrect passwords, as well as for the correct password, the attacker must solve an RTT *before* it receives a yes/no answer that distinguishes the correct password from the incorrect ones. Suppose that the set of all passwords includes N passwords. The attacker can identify, without answering any RTT, that the correct password is a member of a subset of size $p(N-1)+1 \approx pN$ of the passwords – these are the passwords for which it does not receive an immediate "reject" answer but is rather asked to solve an RTT. Given that the implementation requirement is satisfied, there is only one way that the attacker can receive additional information about the passwords – by choosing a password value that it wants to verify and paying with an "RTT solution" in order to learn whether it is the correct value of the password.

Let us assume for simplicity that all password values have the same probability of being the correct password. (This is the case if passwords are chosen at random by the server. Even if the passwords are chosen by the user we can limit P in our analysis to the set of passwords that occur with high probability, and assume that the password is chosen among them with uniform probability. The analysis for the case of a non-uniform probability distribution of passwords is similar.) We can model the attacker as playing the following game: it receives pN identical envelopes, and it knows that a winning ticket is hidden in exactly one of them, chosen uniformly at random. In order to open an envelope it has to pay one coin (i.e. RTT solution). Its goal is to minimize its expected payment until it finds the winning ticket. The analysis of this game is simple. The expected number of coins that have to be paid (i.e. RTTs that have to be solved) is $pN/2$. If the attacker pays c coins (i.e. solves c RTTs), it finds the winning ticket (password) with probability $c/(pN)$.

Analysis for multiple accounts. Assume that the adversary knows ℓ user names, that passwords are chosen uniformly at random, and the the goal of the adversary is to break into any one of the user's accounts. Without answering any RTTs the adversary can learn, for every account, the subset of pN passwords for which an RTT is required. It knows that the passwords were chosen uniformly at random, and that the correct passwords are in these subsets. In a different formulation, it is given ℓ sets of pN envelopes, and it knows that each set has one envelope with a winning ticket. Again, it can choose which envelopes to open but is required to pay one coin per envelope. Since the sets are independent of each other, the adversary's best strategy is to treat each set independently, giving essentially the same results as for a single account.

4.3 Setting the parameters

Assume that we want to make the expected cost for the attacker so high that breaking into accounts does not make sense financially. The design of the protocol should follow the following steps:

- First, we have to estimate the benefit that the attacker gains from breaking into an account. This value differs of course between different servers and accounts.
- We must estimate the size of the domain of passwords that are likely to be used. This is easy if the passwords are chosen by the server (e.g. as random 4 digit numbers), and is more complicated if users are allowed to pick their own passwords.
- We must estimate the cost of solving a single RTT by a potential attacker. An attacker could use three different methods of solving RTTs: The first and trivial method is to guess the answer of the RTT. The second method of breaking RTTs is by employing low paid human workers whose job is to solve RTTs, and implementing a system that performs a dictionary attack,

receives RTT queries and forwards them to the human workers. The third method is to invest resources in designing an automated program that breaks RTTs, and then run servers that perform this attack (we hope that this approach would not be successful, but we should at least estimate the costs of an attacker that wishes to mount an automated attack.)

- We should set the value of p to make the cost of breaking into an account higher than the potential gain from the break. (Of course, if the attacker believes that breaking into a specific account could be highly beneficial we cannot prevent it from targeting this account. We can hope, however, to make the cost of breaking into an account sufficiently high so that the benefit from breaking into an "average" account does not justify the required investment.)

As an example we try to quantify the security of accounts in a specific system, relevant to the three attack scenarios described above.

Assume that accounts are secured with a password with $N = 10^6$ possible values (e.g. the password is a random six digit number, or a random choice of two words from a corpus of 1000 popular words), and that we set p to be $p = 0.1$. For the first scenario assume that there are 1000 possible answers to the RTT query, and only one of them is correct. (For example, the RTT displays a distorted image of a word from a set of 1000 words. The set of possible answers can of course be made much larger.) Then the expected number of attempts that the attacker must do in order to break a single account is $0.5 \cdot 10^6 \cdot 0.1 \cdot 1000 = 5 * 10^7$, which is very high. At a rate of a 100 login attempts per second the attacker should invest $5 * 10^5$ seconds (about one week) in order to break into a single account.

Assume now that an RTT can be broken in 3 seconds, either by human users, or by an automated program (we believe that a plausible scenario is that even if attackers could design automated programs that break RTTs, their running time would not be negligible). In this case the expected number of seconds that have to be invested in order to find the password is $150,000$, which is equal to about 42 hours (about 5 days of human work in 8 hour shifts).

4.4 What happens if the RTT is broken? – dynamic setting of system parameters

Since our experience with RTTs is relatively new, we cannot be sure that RTTs that are considered secure today will still be secure in the future. In particular, we must assume that once RTTs are used to secure something of value they will become the target of many hacking attempts, and attackers might find a way of efficiently breaking RTTs. We should therefore be prepared for the event that RTTs that are used by the system are broken. The analysis below also applies to the situation where an adversary is not able to technically break the RTT, but can obtain a large amount of human help to answer RTTs.

Identifying a successful attack. The first challenge is identifying that attackers are able to break the RTTs that are used by the system. The following techniques will help to identify large scale break-in attempts. During normal operation we can expect a certain fraction of login attempts to pass the RTT but fail in the password entry. These are mostly due to users that mistype their passwords, or to amateurs that run manual dictionary attacks. When the RTT is broken by an automated program, or possibly by attackers that organize "sweatshops" of low paid workers that are solving RTTs, this fraction should go up noticeably (once the RTT is broken the attackers can pass this test, but should still find it hard to guess the correct password). When the system recognizes that the fraction of unsuccessful login attempts that solve the RTT increases, it should assume that the RTT is broken.

Countermeasures. Once it is recognized that attackers can break the RTT, the first line of defense should be increasing the fraction p of login attempts for which an RTT is required. (At an extreme, the system can require *more* than one RTT to be solved for every login attempt, until it finds a different solution against the attack. This corresponds to a value p greater than 1, in terms of calculating the expected number of RTTs that have to be broken). This step requires the attacker to solve more RTTs for every account that it attempts to break. If the attack is conducted using human workers, or is computationally intensive, then the operation of the attacker is slowed down until it recruits more human workers or installs more RTT breaking servers. We should therefore examine whether the fraction of login attempts that pass the RTT but fail the password entry decreases when p is increased. If this is *not* the case, then we should assume that the attacker has found a scalable way of breaking RTTs. The system must therefore change the RTT that is being used, in hope that the attacker does not have an efficient way of solving it. (This change does not slow down human attackers, but we believe, based on the quantitative analysis presented above, that the cost of human attacks is too high for them to be effective, e.g. requires almost a week of human work per account.) The operators of the system must therefore have a set of different RTT tests. Once a successful automated attack is detected the system should switch to a new test, preferably taken from a new domain and sufficiently different from the existing test.

Additional security against attacks on individual accounts can be provided in the following way. When the server notices that there is a high number of failed login attempts to one (or a small number of) accounts, a flag is raised warning the security administrator that with high probability an attack is mounted on these accounts. The system could then increase, for these individual accounts, the fraction of login attempts that require RTTs, or even change the type of RTTs that should be solved, in order to slow down an attacker. The administrator may also contact the user via an out-of-band channel, such as email or phone, to agree on an alternative login method. The key advantages of the RTT scheme in this case are that (1) the security administrator is given significantly more time to react than in "conventional" schemes where an account can be locked within seconds (e.g. after 5 unsuccessful login attempts), and that (2) a legitimate user will still be able to login to his account during this time period.

4.5 Protecting against cookie theft

Cookies contain data that is stored by a server on a client's machine, and can only be retrieved by the server that generated it. There are, however, known attacks that enable hackers to steal cookies from the machines of unsuspecting

users. The protocol we described uses cookies to identify machines from which users have been successfully identified in the past. If a login to a user's account originates from a machine that stores a cookie that contains this user's data then the login procedure does not require solving an RTT. This feature, together with the known weaknesses that enable cookie theft, pose a security threat: once hackers obtain the cookie of a user they can mount a dictionary attack against his account, without being required to answer RTTs.

We describe here a simple procedure that ensures that hackers cannot substantially benefit from obtaining cookies of other users. The procedure requires the server to keep a counter for every cookie that it stores in users' machines. When it receives a cookie as part of a login attempt, and the login attempt fails, it increases the value of the counter of this specific cookie. When the counter reaches some preset value, say 100, a flag is set, causing the server to ignore this cookie in future login attempts. This means that when the user tries to login in the future, and sends the same cookie, the login procedure continues as if no cookie was sent, and the user is required to pass an RTT. When the user succeeds in authenticating himself, a new cookie is generated and is stored on the user's machine with a counter set to 0 (and the previous cookie remains useless).

The effect of this procedure is that an attacker that obtains a cookie is able to check a limited number of passwords (e.g. 100) without having to answer RTTs. The advantage it obtains compared to its normal operation (with, say $p = 0.1$), is negligible: there are about 10 additional passwords that it can check without solving an RTT (out of the 100 login attempts that the attacker runs using the cookie, we expect that 90 would not have required an RTT even if no cookie was present, since $p = 0.1$). The effect on usability is negligible, too. The user has to pass an additional RTT in the rare case that he fails in one hundred login attempts, or when a cookie is stolen from one of his machines and is used for a dictionary attack.

4.6 Account locking after unsuccessful login attempts

As described in Section 1.1, a common measure against dictionary attacks is to lock users' accounts after a certain number of unsuccessful login attempts. We described in Section 1.1 the disadvantages of this measure in terms of vulnerability to denial of service attacks, and the cost of customer service calls.

When a system implements the protocols described in this paper, it can substantially increase the threshold of unsuccessful login attempts that cause an account to lock. Consider a system where passwords have N possible values, and the password is chosen among them uniformly at random. Assume that the system has m accounts whose identities are known to the attacker, that it does not implement the RTT method, and that an account is locked after L unsuccessful login attempts. Then the expected number of accounts that an attacker can break into is $m \cdot L/N$. Assume now that the RTT scheme is used, that an RTT instance has S possible equiprobable answers (say, $S = 1000$), and that the attacker cannot answer an RTT better than guessing the answer at random. Then the effective size of the password space is increased to $N \cdot p \cdot S$, where p is the fraction of incorrect passwords for which an RTT is required (say, $p = 0.1$). This means that the threshold L that causes an account to be locked can be increased by a factor of $S \cdot p = 100$ without changing the expected number of accounts that the adversary can break into.

The system could start its operation using the higher account lock threshold, and work while monitoring the percentage of login attempts that pass the RTT test and fail the password. When the server observes that this figure increases it should suspect that an attacker is able to break the RTT, and implement the countermeasures described in Section 4.4. In addition, it should decrease the threshold that causes accounts to lock, until it implements a new RTT that the attacker is unable to solve.

5. RELATED WORK AND REQUIREMENTS FOR AUTHENTICATION METHODS

5.1 Requirements for authentication methods

In this section we identify some functional requirements and criteria to estimate the costs and benefits of user authentication technology in the context of commercial service providers on the Internet. We will then review related work with respect to these criteria.

Authentication technology on the Internet is typically used to give users access to their accounts. Thus the requirements for the authentication technology are driven by the requirements of the services to which they enable (and restrict) access. As the service provider (and not the end user) is the party that ultimately makes the decision which authentication technology to deploy we will list some common requirements from a service provider's perspective.

The first requirement is *availability*. Web based services should be accessible from a number of machines that can in general only be expected to have standard available tools, in particular a web browser. A typical user expects to access his email and Internet stock trading account from any machine he has access to, for example from computers in Internet cafes that he might use on vacation. To ensure availability for most applications and usage scenarios, authentication methods must also be *portable* among a number of different computers and devices.

The second criterion is *robustness and reliability*, i.e. a legitimate user should always succeed in logging into his account.

The next criterion is *user friendliness*. Internet services strive to provide a good user experience and many of them try to encourage usage. For example processing payments electronically via Internet banking is much cheaper for the bank than processing paper checks, telephone banking or worse, having the customers walk into the branch [4]. Thus the authentication step should not confuse even a fraction of the users, in order not to reduce usage or drive potential customers away from the service. The authentication step should be non-confrontational and should not require a steep learning curve. Ideally authentication should be as seamless and as invisible as possible.

Another key requirement from the service provider perspective is that authentication technology should have *low costs to implement and operate*. These deployment costs go beyond purely technical costs, but also involve the significant costs for customer support calls, in case users are not able to log into their accounts.

Passwords are the most widely used authentication mech-

anism today, despite their known security weaknesses. This phenomenon can probably be explained by the fact that passwords score well with relation to all of the requirements above. Service providers could in principle switch to a more secure authentication technology. However, rational service providers will only switch to a new technology if the expected benefits exceed the switching costs.

We have listed above some of the common functional requirements for authentication technology. Another important factor to is the loss that is incurred by account break-ins. For services such as web based email or sites that contain personalized user information (e.g. financial information), the risk is typically carried by the *user*, *not* by the provider. Furthermore in many scenarios there are typically no financial losses incurred by any of the parties. In particular hackers may not be able receive much financial gain from breaking into these accounts. The main risk is that the privacy of a user's information is compromised. This risk is carried by the user, not the service provider. Here service providers carry mostly a risk to their reputation, in particular if part of their business is based on confidence and trust of their customers. The press around the eBay break-ins (see e.g. [8]) demonstrates that this risk to the reputation of the service provider is real, although we do not know how it exactly translates into lost sales. In spite of the fact that these risks to reputation are harder to quantify, we believe that they give service providers at least a moderate incentive to improve security against account break-ins.

To keep things in perspective it is important to remember that, from a risk management perspective, passwords work reasonably well for most applications (even eBay states that fraud cases are commercially non-significant compared to the overall amount of transactions). Service providers are likely to be reluctant to adopt solutions with high costs. Emerging identity services such as Microsoft Passport or the Liberty Alliance that enable a "single sign-on to the Internet" present higher pay-offs for hackers and thereby make attack attempts more likely. The ongoing public discussion around the security, privacy and trustworthiness of identity services suggests that identity services still need to gain the trust of consumers and may thereby are susceptible to higher risks to their reputation, encouraging them to implement good security mechanisms. A detailed analysis of which risks are carried by which party in an electronic commerce scenario is beyond the scope of this paper. We refer the reader to Bohm et. al [4] for more information on the allocation of risks in, e.g., Internet banking.

The research community has made a number of interesting proposals that improve the security provided by user-chosen passwords. This is a promising and necessary field of research, and may eventually lead to much stronger authentication schemes. On the other hand, commercial mass-market deployment of authentication schemes is a practical problem, which needs to consider factors such as the available standard software tools, the existing infrastructure (e.g. reader devices) and the functional requirements outlined above. We will now look at some of the related work on authentication from this perspective.

5.2 Existing user authentication mechanisms

The first approach for strengthening passwords aims at helping users to choose better passwords. Password management systems may enforce certain rules (e.g. requiring passwords to have at least 6 characters including a numeral, etc). Another variant of this approach is the use of password cracking programs [3, 24] to test which user passwords can be easily broken. Yet another solution is to directly assign to users strong passwords that are chosen by the server. The main drawback of this approach is that these stronger passwords might not be easily remembered by the users. Furthermore, if different service providers have different strategies to enforce strong passwords, users would have to remember a number of different passwords. This runs contrary to the common user strategy of reusing the same password with many accounts. An additional problem is that rejection of user passwords during account opening may not be a smart move for businesses who want to attract users, as it makes authentication more cumbersome. We can therefore conclude that enforcement of strong passwords leads to a number of usability problems.

Hardware based solutions for authentication technology use smartcards, authentication tokens and hardware devices that generate one-time passwords. The obvious drawback of these devices is their expense. It is unclear who should pay for the cost of the devices. The use of smartcards incurres an additional expense of providing users with readers, as mass-market computers today are not equipped with smartcard readers. Requiring smartcards for login would therefore violate the portability requirement. Usability problems occur when users lose or forget their devices. Another usability problem is that unless different service providers build a unified infrastructure (that is very complex and expensive), users would have to use a different hardware token for every service to which they enroll. Requiring users to carry a different token per service could be another major obstacle for adopting this technology. The universal access issue is also relevant to biometric authentication techniques which require extra hardware, as consumer computers are not typically equipped with biometric sensors. We conclude that hardware token methods are unlikely to become a universal authentication method any time soon.

Another class of software based authentication methods is based on high entropy secrets that are stored on the client machine, such as client certificates and the corresponding secret keys. The main challenge for this method is portability, namely making sure that the certificate and the corresponding secret key can be available on any new machine a user would like to use. An additional problem is the secure storage of secret keys.

Another type of a biometric scheme that does not require special readers, enhances the entropy of user passwords by measuring users' key-stroke dynamics while they enter their passwords (cf., e.g. [17, 18] and references given therein). For commercial mass-market applications the reliability of the authentication method is a key requirement. In particular the false negative rate has to be negligible [18] (a false negative occurs when a legitimate user enters his (correct) user name password pair, but will be denied access). Empirical tests [18] suggest that false negative rates are still significant (in the single digit percentage for reasonable parameters). Further research might be able to drive down these figures to a commercially acceptable level. Portability might still clearly remain an issue, since different keyboards could lead to different measurements (e.g. using American vs. German keyboards), and devices such as PDAs may not be able to support such methods at all.

Another general hurdle for authentication methods occurs when their use requires downloading additional software. It is our experience that service providers would like to avoid this for usability reasons, as it may confuse consumers and discourage them from using the system. Furthermore, there may be situations (such as Internet cafes) where users are not allowed to install extra software, i.e. the portability requirement is violated.

Another class of authentication methods replaces textual passwords altogether by graphical password schemes [13, 23]. Graphical passwords could presumably be easily remembered by humans but have significantly higher entropy than text based passwords. Although this is an interesting line of research, massive deployment of this technology requires significant changes in current user behavior and will require user training. Thus early adopters of this technology are likely to deal with significant risks in terms of usability and bootstrapping the system. If only 10% of users of a service with a 1 million customer base would not be able to handle such a transition this might lead to 100,000 (costly) support calls. Service providers that are conservative and risk-averse, such as financial institutions, might be reluctant to introduce a new, and potentially disruptive, technology.

5.3 Our approach

We believe that, in contrast to the technologies surveyed here, the RTT based authentication scheme proposed in this paper scores very well with relation to all the requirements stated above. The scheme i is portable as string RTTs can be displayed by any standard web browser. No additional software downloads or hardware tokens are necessary. User behavior remains almost unchanged. Users only need to remember their passwords, as they do today. Mass deployments by PayPal, Yahoo! and AltaVista have demonstrated that users are able to handle RTTs without major problems, at least as long as they are not required to solve RTTs too often. Additionally, these deployments demonstrate that serving RTTs scales up, at least in principle, to mass deployments. Integration of our basic RTT scheme with deployed password management software appears easy – it only requires changes in a limited set of code in the control logic of the software, and the process of placing cookies on client machines is well understood. Thus the integration of our RTT authentication scheme with existing systems appears to be relatively easy and does not lead to high costs for service providers. The low costs lead us to the conclusion that the scheme has a good chance of adoption even for service providers that are only moderately incentivized to make their authentication technology more secure.

6. REFERENCES

[1] Alta Vista, submission of new urls. `http://addurl.altavista.com/sites/addurl/newurl`

[2] A. L. Coates, H. S. Baird, and R. J. Fateman, *Pessimal Print: A Reverse Turing Test*, Proc., ICDAR 2001, pp. 10-12, 2001.

[3] Bishop, M., *Proactive Password Checking*, 4th Workshop on Computer Security Incident Handling, August 1992.

[4] N. Bohm, I. Brown, B. Gladman, *Electronic Commerce: Who Carries the Risk of Fraud?*, 2000 (3) The Journal of Information, Law and Technology. `http://elj.warwick.ac.uk/jilt/00-3/bohm.html`

[5] M. K. Boyarsky, *Public-Key Cryptography and Password Protocols: The Multi-User Case*, 6th ACM Conf. on Comp. and Comm. Security, 1999.

[6] The CAPTCHA Project. `http://www.captcha.net/`

[7] The CAPTCHA Project: Gimpy. `http://www.captcha.net/gimpy.html`

[8] *Hackers find new way to bilk eBay users*, CNET news.com, March 25, 2002.

[9] C. Dwork and M. Naor, *Pricing via Processing or Combating Junk Mail*, Adv. in Cryptology - CRYPTO '92, Springer-Verlag LNCS 740, pp. 139–147, 1992.

[10] K. Fu, E. Sit, K. Smith, and N. Feamster, *Dos and Don'ts of Client Authentication on the Web*, 10th USENIX Security Symp., August 2001.

[11] O. Goldreich and Y. Lindell, *Session-Key Generation using Human Passwords Only*, Crypto 2001, Springer-Verlag (LNCS 2139), pages 408-432, 2001.

[12] Shai Halevi and Hugo Krawczyk, *Public-key cryptography and password protocols*, ACM Transactions on Information and System Security, Vol 2, No. 3, Pages 230-268, August 1999.

[13] I. Jermyn, A. Mayer, F. Monrose, M.K. Reiter and A.D. Rubin. *The design and analysis of graphical passwords.* 8th USENIX Security Symp., August 1999.

[14] M.D. Lillibridge, M. Abadi, K. Bharat, and A.Z. Broder. Method for selectively restricting access to computer systems. U.S. Patent 6,195,698 (2001).

[15] D.V. Klein, *Foiling the Cracker: A Survey of, and Improvements to, Password Security*, 2nd USENIX Unix Security Workshop, 1990, pp.5-14

[16] P. MacKenzie, *More Efficient Password-Authenticated Key Exchange*, Topics in Cryptology – CT-RSA 2001, pp. 361-377, 2001.

[17] F. Monrose and A. Rubin. *Authentication via keystroke dynamics.* In 4th ACM Conference on Computer and Communications Security, April 1997.

[18] F. Monrose, M. Reiter and S. Wetzel *Password hardening based on keystroke dynamics*, to appear in the International Journal of Information Security, Springer, 2002.

[19] R. Morris and K. Thompson, *Password Security: A Case History*, Communications of the ACM, Vol.22, No.11, November, 1979, pp.594-597.

[20] M. Naor, *Verification of a human in the loop, or Identification via the Turing test*, Manuscript (1996). `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human_abs.html`

[21] Paypal, new account reg. `http://www.paypal.com`.

[22] J. Xu, R. Lipton, I. Essa, M.-H. Sung, *Mandatory human participation: A new scheme for building secure systems*, Georgia Institute of Technology Technical Report GIT-CC-01-09, 2001.

[23] A. Perrig and R, Dhamija, *Dj Vu: A User Study Using Images for Authentication*, 9th Usenix security Symp., August 2000.

[24] Spafford, E. H., *Opus: Preventing Weak Password Choices*, Computers & Security, 11 (1992), 273-278.

[25] Workshop on Human Interactive Proofs `http://www.parc.xerox.com/istl/groups/did/HIP2002/`

[26] Yahoo!, new account registration.