Addressing Online Dictionary Attacks
with Login Histories and Humans-in-the-Loop
(Extended Abstract)

S. Stubblebine[1] and P.C. van Oorschot[2]

[1] Stubblebine Research Labs, Madison, NJ, USA
[2] Computer Science, Carleton University, Ottawa, Canada

Abstract. Pinkas and Sander's (2002) login protocol protects against online guessing attacks by employing human-in-the-loop techniques (also known as Reverse Turing Tests or RTTs). We first note that this, and other protocols involving RTTs, are susceptible to minor variations of well-known middle-person attacks, and suggest techniques to address such attacks. We then present complementary modifications in what we call a history-based protocol with RTT's. Preliminary analysis indicates that the new protocol offer opportunities for improved security, improved user-friendliness (fewer RTTs to legitimate users), and greater flexibility (e.g. in customizing protocol parameters to particular situations).

1 Introduction

Recent interest has arisen in tests which distinguish humans from computers, and in using such tests to ensure human involvement in a wide range of computer-based interactions. The idea is to find simple tasks which are relatively easily performed by a human, but which appear difficult or infeasible for automated programs to carry out – for example, visually recognizing distorted words. Mechanisms involving such tests have been referred to as *human-in-the-loop protocols*, *mandatory human participation schemes*, and *Reverse Turing Tests* (RTTs) [19, 5, 22].

One specific purpose for which RTT challenges have been proposed is protecting web sites against access by automated scripts. RTTs are currently being used to protect against database queries to domain registries, to prevent sites from being indexed by search engines, and to prevent "bots" from signing up for enormous numbers of free email accounts [5]. They have also been proposed for preventing more creative attacks [4].

Our main interest in RTTs is their use to protect web servers against online password guessing attacks (e.g. online dictionary attacks). The idea is that automated attack programs will fail the RTT challenges. A specific instance of such a protocol was recently proposed by Pinkas and Sander [20] (see §3). While this protocol appears to be quite simple, closer inspection reveals it to be surprisingly subtle and well-crafted. Simpler techniques preventing online dictionary attacks are not always applicable. For example, account lock-out after a small number
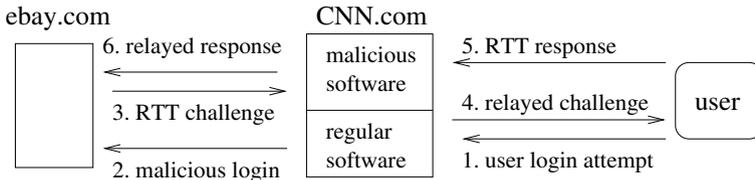
Fig. 1. RTT Relay Attack

of failed password attempts may result in unacceptable side effects, such as increased customer service costs for additional telephone support related to locked accounts, and new denial of service vectors via intentional lock-out of other users [24]. Another standard approach is to use successively longer delays as the number of successive invalid password attempts on a single account increases. This may lead to similarly unacceptable side effects.

In this paper, we begin by noting that many RTT-based protocols, including that of Pinkas and Sander, are vulnerable to an *RTT relay attack*: RTT challenges may be relayed to possibly unsuspecting parties, who generate responses which are then relayed back to the challenger. We explore this threat and mechanisms to address it, and propose additional (orthogonal) enhancements to the Pinkas-Sander protocol.

This paper is organized as follows. §2 presents the RTT relay attack. §3 discusses background context and assumptions, including a reference version of the basic RTT-based login protocol. §4 presents a new variation, with enhancements aimed towards usability, security against online dictionary attacks, and parameter flexibility. §5 discusses standard techniques to augment general RTT-based login protocols to prevent, detect or deter attacks including the relay attack. §6 provides background and a summary of related work. §7 contains concluding remarks.

2   RTT Relay Attack

A relay attack (see §6) may be carried out on online protocols involving an RTT by relaying the RTT challenge to an auxiliary location or "workforce" which generates responses, which are relayed back to the challenger. The original RTT target thus escapes computing RTT responses.

One attack variant might proceed as follows (see Fig. 1). Assume there are two web sites[1]. The first, say ebay.com, is assumed to be the target of regular online dictionary attacks, and consequently requires correct responses to RTT challenges before allowing access. The second, say CNN.com, is a popular high volume web site, which for our purposes is assumed to be vulnerable to compromise. The attack begins with an adversary hacking into the CNN.com site and installing attack software.

---

[1] The authors have no affiliation with ebay.com or CNN.com, and no reason to believe either site is insecure. These sites are used as examples simply due to their popularity.

Upon a user initiated HTTP connection to CNN.com, the attack software receives the request and initiates a fraudulent login attempt to ebay.com. The attack software, presented with an RTT challenge from ebay.com, redirects it to the CNN.com user connection, instructing that user to answer the RTT to get access to CNN.com. (Many users will follow such instructions: most users are non-technical, unsuspecting, and do as requested.) The CNN.com user responds to the RTT challenge. The attack software relays the response to ebay.com, completing the response to the challenge to the fraudulent login attempt. In conjunction with replying to eBay's RTT challenge, after a sufficient number of passwords guesses (e.g. dictionary attack), an eBay account password can be cracked. The procedure is repeated on other accounts, and the attack program summarizes the online dictionary attack results for the adversary.

The attack is easy to perform if the adversary can control *any* high volume web site – e.g. a popular legitimate site the attacker compromises (as above), or an owned malicious site to which traffic has been drawn, e.g. by illegally hosting popular copyrighted content, a fraudulent lottery, or free software. A related attack involves attack software which relays RTTs to groups of human workers ("sweatshops"), exploiting an inexpensive labor pool willingly acting as a mercenary RTT-answering workforce. An unconfirmed real-world variant reported [21] to involve an "adult web site" requiring users to solve RTTs before being served the content; presumably those running the site relayed the answers to gain access to legitimate sites which posed the original RTT in the hope of preventing automated attacks. Our discussion of mechanisms to counteract such threats continues in §5.

3   Background, Constraints, Assumptions, and Objectives

For reference, Fig. 2 provides a simplified description of the original RTT-based login protocol (for full details, see [20]). The system parameter p is a probability which determines the fraction of time that an RTT is asked, in the case that an invalid userid-password pair is entered. In the case of a successful login, the protocol stores a cookie on the machine from which the login occurred; the cookie contains the userid (plus optionally an expiration date), and is constructed in such a way (e.g. using standard techniques involving symmetric-key encryption or a MAC) that the server can verify its authenticity.

For context, we next state a few assumptions and observations relevant to both the original and new protocols. We begin with a basic constraint.

Constraint 1: Account Lock-out Not Tolerable. We are interested in protocols for systems where locking-out of user accounts after some number of failed login attempts is not a viable option. (Otherwise, online login attacks are easily addressed – see §1.)

Trust Model Assumptions: Trusted Host and Ephemeral Memory. We assume that client computers, and any resident software at the time of use, are trusted (e.g. no keyboard sniffers or malicious software run on the machine). This is standard for (one-factor) password-based authentication protocols – otherwise,

```
1   fix a value for system parameter p, 0 < p ≤ 1 (e.g. p = 0.10)
2   user enters userid/password
3   if (user PC has cookie) then server retrieves it
4   if (entered userid/password pair correct) then
5     if (cookie present & validates & unexpired & matches userid) then
6       login passes
7     else % i.e. cookie failure
8       ask an RTT; login passes if answer correct (otherwise fails)
9     endif
10  else % i.e. incorrect userid/password pair
11    set AskAnRTT to TRUE with prob. p (otherwise FALSE) †
12    if (AskAnRTT) then
13      ask an RTT; wait for answer; then say login fails
14    else
15      immediately say login fails
16    endif
17  endif
      † This setting is a deterministic function of the userid/password pair [20]
```

Fig. 2. Original RTT-based Login Protocol (simplified description)

the password is trivially available to an attacker. For similar reasons, we assume client software leaves no residual data on user machines after a login protocol ends (e.g. memory is cleared as each user logs out). In practice it is difficult to guarantee these assumptions are met (e.g. for borrowed machines in an Internet cafe); but without them, the security of almost all password protocols seems questionable.

Observation 1: Limited Persistence by Legitimate Users. A typical legitimate user will give up after some maximum (e.g. $C = 10$) of failed logins over a fixed time period, after which many will check with a system administrator, colleague or other source for help, or simply stop trying to log in. Large numbers of successive failed logins, if by a legitimate user, may signal a forgotten password or a system availability issue (here login failures are likely not formally recorded by the system); or may occur due to an attacker, as either a side effect of attempting to crack passwords, or intentionally for denial-of-service in systems susceptible to such tactics.

Observation 2: Users Will Seek Convenience. If a login protocol is necessary to access an online service, and users can find a similar alternate service with a more convenient login (though possibly less secure), then many users will switch to the alternate service. User choices are rarely driven by security; usability is usually a far greater factor, and poor usability typically leads to loss of business.

These observations lead us to our usability goal; we state it informally.

Usability Goal – Minimal Inconvenience to Users. Relative to standard userid-password schemes, we wish to minimize additional inconvenience experienced by a user.

As usual, the usability goal must be met in a tradeoff with security, and we have a two-part security goal. One part is protecting specific accounts (e.g. certain users may be more concerned about, or require more, protection; or a service provider may worry more about specific accounts – say those with high sales ratings, or high account values). The second is protecting all accounts in aggregate (e.g. a web service provider might not want *any* user accounts misapropriated to host stolen software; a content service provider might want to protect access to content available to authorized subscribers).

Security Goal – Control Access to both Specific Accounts and Non-specific Accounts. Constrain information the adversary learns from trial password guesses before being "stopped" by an RTT challenge in the context of fully-automated attacks directed towards a specific account (single-account attack) and towards any account (multi-account attack).

In practice, for authentication schemes based on user-selected passwords, prevention of unauthorized access cannot be 100% guaranteed for a specific account or all accounts in aggregate, due to the non-zero probability of correctly guessing a password, and the ubiquity of poor passwords. Nonetheless, the quality of a login protocol may be analyzed independent of particular password choices, and this is what we pursue. For a given password, we are interested in how effectively a given protocol allowing online interaction prevents extraction of password-related information. As little information as possible should be leaked.

Requiring mandatory human participation increases the level of sophistication and resources for an attack. If RTTs are effective and RTT relay attacks are countered (e.g. by means such as embedded warnings – see §5.1), then constraining information leaked before being "stopped" by an RTT challenge is an important security characteristic of a password-based login protocol.

4 History-Based Login Protocol with RTT's

Here we modify the original protocol, intending to both improve the user experience and increase security, e.g. to increase the percentage of time that an adversary is challenged with an RTT, without further inconveniencing legitimate users[2]. The modifications do not themselves prevent RTT relay attacks (§2), but are complementary to those in §5 that do, and can thus be combined. We also provide analysis of the new protocol.

4.1 New Protocol

We assume familiarity with the original protocol (§3). The new protocol is given in Fig. 3. Line changes from the original protocol (Fig. 2) are: lines 7.1-7.6 replace 8; and 11.1 replaces 12. The new protocol with failed-login thresholds $(b_1 = 0, b_2 = \infty)$ behaves the same as the original protocol.

---

[2] One might try to improve usability by allowing a small number of trial passwords per userid without triggering an RTT. While this reduces security only minorly for a single-account attack (see §4.2), the problem is greater with multi-account attacks.

1   fix values for $0 < q \leq 1$ (e.g. $q = 0.05$ or 0.10) and integers $b_1, b_2 \geq 0$
2   user enters userid/password
3   **if** (user PC has cookie) **then** server retrieves it
4   **if** (entered userid/password pair correct) **then**
5   **if** (cookie present & validates & unexpired & matches userid) **then**
6   login passes
7   **else** % i.e. cookie failure
7.1   set AskAnRTT to TRUE if account is in owner mode (otherwise FALSE)
7.2   **if** (AskAnRTT) OR (*FailedLogins[userid]* $\geq b_1$) **then**
7.3   ask an RTT; login passes if answer correct (otherwise fails)
7.4   **else**
7.5   login passes
7.6   **endif**
9   **endif**
10 **else** % i.e. incorrect userid/password pair
11   set AskAnRTT to TRUE with prob. q (otherwise FALSE) †
11.1   **if** (AskAnRTT) OR (*FailedLogins[userid]* $\geq b_2$) **then**
13   ask an RTT; wait for answer; then say login fails
14   **else**
15   immediately say login fails
16   **endif**
17 **endif**
   † This setting is a deterministic function of the userid/password pair [20]

Fig. 3. New Protocol (History-based Login Protocol with RTT's). *FailedLogins[userid]* is set to the userid's number of failed logins in a recent period T, and updated (not shown). See §4.1 re: handling cookies and definition of owner mode

We next discuss some differences between the new and original protocols, including: cookie-handling (related to owner and non-owner mode) – cookies are now stored only on trustworthy machines; per-user tracking of failed logins; and setting failed-login thresholds. The idea of dynamically changing failed-login thresholds has been previously mentioned [20, §4.4-4.6]; we detail a concrete proposal and comparison.

Handling cookies. The original protocol stores a cookie on any device after successful authentication; the new protocol does not. Optional user input controls cookie storage similar to web servers using a login page checkbox asking if users want to "remember passwords", e.g. "Is this a trustworthy device you use regularly? YES/NO". This part of the page appears if no cookie is received by the server. Upon a YES response, a cookie is pushed to the user device only after the user successfully authenticates (requiring a successful RTT response, if challenged). This cookie approach reduces exposure to cookie theft vs. the original protocol, with negligible usability downside because the question appears on the same screen as the login prompt (default answer NO).

The original protocol requires that cookies be tracked by the server and expire after a limit of failed login attempts with the particular cookie [20, §4.5]. We follow a similar approach. Each time a login fails (e.g. lines 7.3, 13, and 15),

we increment the failed login count associated with the cookie if a valid cookie was received. If the cookie exceeds a failed login threshold, we invalidate it. Line 5 includes a check that the cookie hasn't been invalidated. The *cookie failure threshold* is the number of failed logins allowed before a cookie is invalidated. We recommend setting this to the minimum of b_1 and b_2.

Definition of owner, non-owner. A user is more likely to login from "non-owned" devices when traveling (e.g. borrowing an Internet access device in a library, guest office, conference room, or Internet cafe). Also, a user submitting a login request which does not include a cookie is likely to be using a non-owned device. As a consequence of how cookies are handled, we can assume (with small error) that a user is on a non-owned device if their most recent successful login does not include a cookie. We initially define a user account to be in "*owner*" mode, and expect an account to be in owner mode most of the time if most of the time they use their regular device (e.g. one of the devices they own). An account transitions to "*non-owner*" mode when a login is successfully authenticated without the server receiving a valid cookie (Fig. 3, line 7.5), and returns to owner mode after a specified time-out period W (e.g. 24 hours) or a successful login with a cookie present. The timeout period is restarted, and the account remains in non-owner mode, if there is another cookieless successful login. The time-out period reduces the number of accounts in non-owner mode, which lowers the security risk; accounts in non-owner mode are more susceptible to multi-account dictionary attacks (see §4.2).

Tracking failed logins. We define *FailedLogins[userid]* to be the number of failed login attempts for a specific userid within a recent period T (e.g. 30 days). Here *failed login attempts* includes: non-responses to RTT challenges, incorrect responses, failed userid-password pairs, and outstanding authentication attempts (e.g. the adversary may simultaneously issue multiple login attempts; one strategy might be to issue a very large number, and respond to only a subset of resulting RTT challenges, perhaps being able to exploit some "weak sub-class" of RTTs for which computer-generated responses are feasible).

Setting the failed-login thresholds (bounds b_1, b_2). Low values for b_1, b_2 maximize security at the expense of usability (e.g. for users who frequently enter incorrect passwords). A reasonable bound may be $b_1, b_2 \leq 10$ (perhaps larger for large T). In the simplest case the protocol bounds b_1, b_2 are fixed system variables; in a more elaborate design, they (and q) are dynamic and/or set on a per-user basis (varying for a particular userid, based on a history or profile and possibly subject to system wide constraints e.g. maximum bound on b_2). For example, certain users who regularly enter a password incorrectly might be given a higher failed-login threshold (to increase usability) compared to users who almost always enter correct passwords. If it is expected or known from a historical profile that a user will log in L times over a period T, and that say 5% of legitimate login attempts fail, then b_2 might be set somewhat larger than $(0.05) * L$ (e.g. $T = 30$ days, $L = 100$, $b_2 = 5$). Over time, per-user rates of legitimate failed logins (e.g. mistyped or forgotten/mixed up passwords, perhaps more frequent on unfamiliar machines) can be used to establish reasonable thresholds. To simplify

presentation, updating of per-user table entries *FailedLogins[userid]* in Fig. 3 is not shown. Note that while per-user values require server-side storage when these values cannot be user-stored via cookies, a small amount of per-user server-side storage is already required in both the original and new protocol to ameliorate cookie theft (see above). (Optionally, setting the RTT challenge probability q on a per-user basis also allows flexibility for tuning usability and security on a per-account basis.)

4.2 Comparitive Analysis – Security and Usability

For a comparitive analysis of the new protocol with the original protocol, we first focus on the analysis for a single account, with respect to security and usability. We generally follow the assumptions from the original protocol [20], including that passwords are from a fixed set (dictionary) of cardinality N, and that for analysis purposes they are equally probable. The probabilities p and q are as defined in the protocols.

Discussion of Security (Single-Account Attacks). To aid our single-account security analysis, we use the following questions (and assume for now no cookie theft, i.e. an attacker knows a userid but has no corresponding cookie). For a single user account, for the original and new protocols, what is the ...
Q1: expected number of passwords eliminatable from the space, answering no RTT's? Q2: expected number of RTT's an attacker must answer to correctly guess a password? Q3: probability of a confirmed correct password guess for attacker willing to answer c RTT's?

The answers summarized in Table 1 are based on the best attack strategies known to us[3]. For Q2 and Q3, perhaps surprisingly, this involves an attacker simply answering the first c RTT's sent[4]. Since better attack strategies may exist, e.g. our answers to Q3 should be interpreted as lower bounds, albeit under conditions favorable to the attacker: we assume that failed login counts are 0 at the start of an attack.

Some observations follow. Rows Q1 and Q2 indicate that the number of passwords that an attacker is able to eliminate "for free" (without any RTT's) is substantially greater in the original protocol[5]. A second observation favoring the new protocol is evident from rows Q3b and Q3c: the probability of a successful attacker guess in the new protocol (on the order of $1/N$) is generally significantly smaller than in the original (on the order of $1/pN$), except that when b_2 is relatively large the new protocol's behaviour effectively becomes that of the original, with probability c/qN matching the table entry c/pN for the Original Protocol; when b_2 is small, $b_2 + c$ is less than c/q, so the probability in the new protocol is better, i.e. less than the original.

---

[3] Currently, we make a simplifying assumption: an account is in one of the two modes.
[4] Additional details on attack strategies and Table 1 will be provided in the full paper.
[5] For the new protocol, these figures are per time period T. However for a sophisticated multi-period attack, the new protocol remains better (fewer passwords are eliminatable), assuming $p = q$, unless at least N/b_2 time periods are used (e.g. about 1600 years for $T = 1$ month, $N = 100\,000$ and $b_2 = 5$).

Table 1. Tabular data for comparitive analysis (single-account attack). q is used for p in the new protocol to emphasize possible use of different values ($p = q$ is also possible).

Question	Original Protocol	New Protocol	
		Account Mode	
		Owner	Non-owner
Q1	$(1-p)N$	$z_1 = (1-q)b_2$	$z_2 = \max(b_1, (1-q)b_2)$
Q2	$\frac{1}{2}pN$	$\frac{1}{2}(N-z_1) \approx N/2$	$\frac{1}{2}(N-z_2)$
Q3a ($c=0$)	0	0	b_1/N
Q3b ($c=1$)	$1/pN$	$(1-(1-q)^{b_2+1})/qN$	$(b_1+1)/N$
Q3c ($c \geq 2$)	c/pN	$\min(\frac{c}{q}, b_2+c)/N$ †	$(b_1+c)/N$

†Upper bound

Table 2. Fraction of the time a legitimate user must answer an RTT ($1.0 = 100\%$). As in Table 1, q is used in place of p in the new protocol.
†After the failed login bound is crossed in the new protocol, in several cases – e.g. on incorrect passwords, and correct passwords without valid cookies – RTT's occur more frequently (i.e. 100% of the time after the bound is crossed within period T). However for accounts in owner mode we expect a large number of users select a "Remember password" option (standard in many applications) which stores passwords locally on their regular machines. No failed passwords are expected from such users; but note their failed login thresholds may still be crossed due to attacker activities.

	Original Protocol	New Protocol	
		Account Mode	
		Owner	Non-owner
Incorrect password	p	q†	q†
Correct password - valid cookie	0	0	0
Correct password - no valid cookie	1.0	1.0	0†

Note: for both $c = 1$ and $c \geq 2$, the table gives a probability (i.e. an expectation over a large number of runs). The new protocol has a guaranteed upper bound on the probability: $(b2 + c)/N$.

According to row Q3a, for an attacker unwilling to answer any RTTs, the security is the same in both protocols except we relax security (i.e. to b_1/N) for some small number of accounts in non-owner mode to improve usability (see usability improvement in Table 2, bottom row).

A security analysis for multi-account attacks (wherein an attacker's goal is to break into any one of many accounts, not necessarily a specific account) and parallel login attacks (wherein an attacker may try to simultaneously login to one userid a large number of times on different servers) is left for the full version of this paper.

Discussion of Usability. For comparing usability between the original and the new protocols, Table 2 notes the proportion of time a legitimate user is queried with an RTT on entering a correct or incorrect password, with and without a valid cookie. A case of particular focus for the new protocol is the

legitimate "travelling user", who generally operates with an account in non-owner mode and without a valid cookie. The new protocol is significantly more user-friendly to such users. We also believe that such users are typically more likely to enter incorrect passwords (see discussion in caption of Table 2), and therefore increasing usability in this case is significant as one would expect that "incorrect password" cases occur far less often in owner mode.

Also related to usability – the value of the parameter q may be reduced in the new protocol without loss of security, due to the use of the failed login bound b_2 and depending on its value relative to q (see Table 1). This further increases usability in the incorrect password case, independent of the discussion in the paragraph above.

Discussion of Cookie Theft. The above analysis assumes that no cookie theft occurs; here we make a few observations in the case cookie it does.

1. *New Protocol.* If a cookie is stolen, then within the cookie's validity period, under the recommended cookie failure threshold, the attacker gets $\min(b_1, b_2)$ password guesses on the userid. The attack we consider is one where the attacker quits all guesses that return an RTT, and having a good cookie, hopes to reach line 6 with a lucky guess[6].
2. *Original Protocol.* Similarly, the attacker gets free guesses up to the cookie failure threshold. A correct password guess on any of these trials allows a successful login without having to answer an RTT.

Comments. (a) It is less likely that a cookie is stolen under the new protocol, since they reside in fewer places – e.g. cookies of the original protocol would show up in airport Internet rooms. (b) A combined cookie and non-cookie attack against a single account is less likely to be successful in the new protocol, primarily because the attacker can reduce the password space to a p-fraction in the original protocol even before using the stolen cookie (see related discussion on questions Q1 and Q2).

5   Additional Techniques Augmenting RTT-Based Authentication

Here we propose a number of techniques to augment the original protocol (Fig. 2), without changing its basic functionality. This includes addressing RTT relay attacks (§2). These techniques are intended primarily to improve security, and are independent of (orthogonal to) the changes proposed in §4. We present them briefly without additional analysis.

5.1   RTT with Embedded Warning

Here we propose a simple method to prevent RTT relay attacks. A drawback of the proposal is that it requires some thought on behalf of users (which is, in

---

[6] This attack may take place in conjunction with one that reduces the password space without answering an RTT, or one where the adversary answers c RTTs.

some cases, unfortunately unrealistic). However, we believe the general idea may be adapted to significant advantage.

The general idea is to rely upon self-awareness of legitimate users to prevent unwitting participation in an RTT relay attack. One approach is to make RTT challenges user-directed by incorporating a user's specific userid *within the RTT itself*. Preferably, removing this information is of comparable difficulty as answering the RTT itself.

For example, as part of answering a text RTT, a portion of the text is a userid field[7], which the user is warned to compare to their own userid, thereby confirming that the RTT is targeted specifically at them (within the embedded warning the user is instructed to not answer the RTT if the match fails). As an additional optional feature, the RTT might also contain an embedded short "help URL", for a site giving further instructions on the use of this type of RTT.

This idea is analogous to the now generally accepted, and recommended, practice in authentication protocols of putting party identifiers within the protected (i.e. signed or MAC'd) region of protocol messages. It is also analogous to the typical automated check, when using secure browser cookies, that cookies match a particular userid or IP address; and to the matching userid check in the original protocol (line 5, Fig. 2).

5.2   Notification Regarding Failed Logins

Here we propose a simple method to detect automated dictionary attacks and trigger counter-active measures[8]. Once a small threshold (e.g. 3-10) login failures occurs for any single account, an automated, out-of-band communication (e.g. email) is sent to an address-on-record of the associated legitimate user. If the failed logins resulted from the user's own actions, the user will be aware of the failures and can safely ignore the message; otherwise, it signals malicious activity, and may lead the user to take such actions as to request[9] changes to server-side user-specific login protocol parameters (see §4), or to change their own password to a more secure password using the normal change password method.

As an alternative, albeit less desirable[10], after some larger number of failed logins (e.g. 25), the system might automatically reset the user's password to a computer-generated secure password emailed to the user. This would prevent a user's typically weak self-chosen password from being cracked through standard dictionary attacks. (Depending on the security policy in use, the user might be allowed to change the password back to a weak one if they wish, but at this point they may also be motivated to follow recommended password rules.)

---

[7] A variant instead includes the name of the site being visited, with similar explanation. (An anonymous referee suggested this.) The choice between web site name and userid could be made dynamically, e.g. selecting the shorter of the two.

[8] This expands on administrators manually sending out-of-band messages [20, §4.4].

[9] For example, through an authenticated channel such as an email to an un-advertised pre-arranged address, or a hidden URL provided in the email alert to the user.

[10] This may raise customary issues related to system-generated passwords and system-initiated password changes. If used, this alternative must be crafted so as not to generate additional customer service calls, which are not tolerated within our scope.

This proposal is less effective against multi-target attacks, and *slow-channel dictionary attacks* wherein an automated program tries passwords on a certain account after there is likely to have already been a successful login attempt (e.g. waiting for a random but minimal delay, such as one-day intervals). In some systems, an attacker can confirm if a user has logged in recently (e.g. an eBay user), and mount only a limited number of trial password guesses some fixed period after each such successful login. This proposal may nonetheless be helpful, and other parameters may limit the success of slow-channel attacks. A small amount of per-user server-side state is needed, but the original protocol has a similar requirement to address cookie-theft [20, §4.5]. A remaining drawback of this proposal is degraded usability (additional user attention is required).

5.3   Consuming Client Resources
Using Zero-Footprint Software Downloads

We propose that login protocol variants (e.g. see §4) be augmented by known techniques requiring that clients solve "puzzles" consuming client resources, and return answers prior to the server verifying a login. This follows research lines to combat junk mail (e.g. [8, 1]) and denial-of-service attacks [15]. Another augmenting technology is to harden passwords with auxilliary protocols that can interact directly with the server [11].

Since functionality for performing client puzzles is not resident in standard client software (e.g. browsers), this proposal requires allowing Java applets, Javascript, or other zero-footprint downloads. We no longer agree with dismissing special client-side software outright (cf. [20]); rather, we see opportunity for advantageous use. Though perhaps worrisome, most users and organizations now operate under the assumption that Java, and certainly Javascript, are turned on[11]. Nonetheless, since popular web services should work for 100% of potential users, to accommodate those who cannot use zero-footprint software, RTT-based login protocols can be designed as follows. Client puzzles (or the like) are sent to users. For those unable to answer the puzzles for any reason (in some case the server may learn this *a priori*), the protocol branches to a path replacing the puzzle by an (extra) RTT. This RTT will be less convenient to the user (requiring user attention, vs. machine resources), but we expect this to be a relatively small percentage of users, and therefore viable.

6   Background and Related Work

The RTT relay attack of §2 is related to general classes of *middle-person attacks* and *interleaving attacks* involving an active attacker inserting itself between legitimate parties in a communications protocol, and/or using information from one instance of a protocol to attack a simultaneous instance. Such attacks

---

[11] These are in fact the settings that result from the Internet Explorer default ("medium" security), and which we expect remain unchanged by most users.

are well-known in cryptographic protocols and have a long history ([6, 7]; [18, pp.530-531]).

For example, challenge-response protocols have long been used to identify military aircraft in *identify-friend-or-foe* (IFF) systems. IFF challenges from enemy challengers have reportedly been forwarded in real-time to the enemy's own planes, eliciting correct responses which were then successfully used as responses to the enemy's original challenges [2, pp.19-20]. Note that responses in such systems are typically automatic; the protocols do not involve entity authentication of the querying party.

Related to this is the well-known grandmaster postal-chess attack: an amateur simultaneously plays two grandmasters by post, playing white pieces in one game and black in the other, using the moves of his opponents against each other, resulting in an overall outcome better than the two losses he would have achieved on his own.

The term *strong authentication protocols* is often used for protocols designed to preclude attacks which first obtain appropriate data related to one or more protocol runs, and then proceed to crack passwords offline (i.e. without further interaction). This line of research began with the early work of Gong and co-authors [17, 12, 13]; Bellovin and Merritt's EKE protocol [3] then inspired a number of others (e.g. Jablon's SPEKE [14]; Wu's SRP [23]; see also [16]).

Offline exhaustive password-guessing attacks typically proceed by trying potential passwords in order of (perceived) decreasing likelihood. The most probable passwords are often in conventional dictionaries, or modified dictionaries specially tailored to this task. Offline attacks are thus often called *dictionary attacks*, although dictionaries are also used in online attacks (if account lock-out and time-delays are not used; see §2).

Use of system-generated passwords can provide higher security (by better password choices), but suffers severe usability issues. Pass*phrases* have also been proposed (e.g. see [27, 26]). Other approaches include system administrators running password-crack tools on their own systems (*re-active* password checking); enforcement of simple password rules or policies at the time of new password selection; and at such time, checking for its presence in large customized dictionaries built for this purpose (*pro-active* password checking, e.g. see Yan [25] for a recent summary).

7   Concluding Remarks

We expect that a large number of human-in-the-loop and mandatory human participation schemes, unrelated to the RTT-based login protocol discussed here, are also subject to the RTT relay attack of §2.

A major feature of our new protocol (§4) is the additional flexibility and configurability, including failed login thresholds and potentially lower RTT challenge probabilities (e.g. for suitable b_2 lowering q does not decrease security). This allows the protocol to be tailored to match particular environments, classes of users, and applications; while determining the optimal parameters for spe-

cific user profiles appears non-trivial, we expect further analytical study will be fruitful. Another new aspect is storing cookies only on trustworthy machines. As mentioned earlier, the new protocol can be parameterized to give the original protocol as a special case. While the configurability does complicate protocol implementation somewhat, we note that a number of the parameters which are optionally dynamic can be managed by automated tools; thus the additional human administrative costs are relatively minor. For example, an automated tool can keep a running ratio of successful logins to failed logins for the entire system, and alter system wide (or account-specific) parameter q, or system-wide (or account-specific) failed login thresholds b_1 and b_2, based on this ratio. A significant improvement of our protocol over prior work concerns protecting against relay attacks by forcing an RTT challenge on all login attempts after the number of failed logins reaches a threshold. Previous work enabled a significant fraction of the password space to be eliminated with an automated attack. Per-user failed-login counts (as used in Fig. 3) also provide protection against sweatshop attacks and RTT relay attacks, especially such attacks targeting a particular account. Note that embedding warnings within RTTs (§5.1) does not by itself protect against sweatshop attacks.

For practical protection in Internet-scale live systems, we recommend combining techniques from §5 with those of §4. We see a large number of ways to expand on the ideas of §4. In particular, we encourage others to explore the use of dynamic parameters (ideally managed by automated tools), and other ways to gain advantage by treating users logging in from non-owned devices (e.g. traveling users) different from those continually using their regular login machines.

Acknowledgements

References

1. M. Abadi, M. Burrows, M. Manasse, T. Wobber, "Moderately Hard, Memory-bound Functions", NDSS'03, San Diego, February 2003.
2. R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley, 2001.
3. S. Bellovin, M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attack", *Proc. IEEE Symp. Research in Security and Privacy*, Oakland, May 1992.
4. S. Byers, A. Rubin, D. Kormann, "Defending Against an Internet-based Attack on the Physical World", Workshop on Privacy in the Electronic Society (WPES'02), November 21 2002, Washington D.C.
5. CAPTCHA Project web site, http://www.captcha.net/ (first appeared: 2000).

6. W. Diffie, M. Hellman, "New Directions in Cryptography", *IEEE Trans. Info. Theory* vol.22 (1976), pp.644-654.

7. W. Diffie, P.C. van Oorschot, M.J. Wiener, "Authentication and Authenticated Key Exchange", *Designs, Codes and Cryptography* vol.2 (1992), 107-125.

8. C. Dwork, M. Naor, "Pricing via Processing or Combatting Junk Mail", *Lecture Notes in Computer Science 740 (Proceedings of CRYPTO'92)*, 1993, pp. 137–147.

9. Password Usage, Federal Information Processing Standards Publication 112, U.S. Department of Commerce, NIST, 1985.

10. Automated Password Generator, FIPS Pub 112, U.S. Dept. Commerce, 1993.

11. W. Ford, B. Kaliski, "Server-Assisted Generation of a Strong Secret from a Password", 9th Int'l Workshop on Enabling Technologi (WET-ICE 2000), IEEE, 2000.

12. L. Gong, "Verifiable-text attacks in cryptographic protocols", *1990 IEEE INFO-COM*, pp.686-693.

13. L. Gong, T. Lomas, R. Needham, J. Saltzer, "Protecting poorly chosen secrets from guessing attacks", *IEEE J. Selected Areas Comm.* vol.11 (1993), pp.648-656.

14. D. Jablon, "Strong password-only authenticated key exchange", *ACM Computer Communcations Review*, Oct.1996.

15. A. Juels, J. Brainard, "Client puzzles: A cryptographic defense against connection depletion attacks", *Proceedings of the 1999 ISOC Network and Distributed System Security Symposium*, pp.151-165, 1999.

16. C. Kaufman, R. Perlman, M. Speciner, *Network Security: Private Communication in a Public World*, Second Edition, Prentice Hall, 2002.

17. T. Lomas, L. Gong, J. Saltzer, R. Needham, "Reducing risks from poorly chosen keys", *Operating Systems Review* vol.13, pp.14-18 (presented at 1989 ACM Symp. on Operating Systems Principles).

18. A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

19. M. Naor, "Verification of a human in the loop or Identification via the Turing Test", unpublished manuscript, 1997. Online version available at: http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human.ps

20. B. Pinkas, T. Sander, "Securing Passwords Against Dictionary Attacks", *2002 ACM Conf. on Computer and Communications Security*, Wash. D.C.

21. L. von Ahn, Eurocrypt'03 presentation of [22], 6 May 2003, Warsaw, Poland.

22. L. von Ahn, M. Blum, N. Hopper, J. Langford, "CAPTCHA: Using Hard AI Problems for Security", *Eurocrypt'03* proceedings, Springer-Verlag, LNCS 2656 (2003).

23. T. Wu, "The secure remote password protocol", Internet Society *1998 Network and Distributed System Security* symposium (NDSS'98).

24. T. Wolverton, Hackers find new way to bilk eBay users, CNET news.com 03/25/02.

25. J. Yan, "A Note on Proactive Password Checking", *Proc. 2001 ACM New Security Paradigms Workshop*, New Mexico, USA, Sept.2001.

26. J. Yan, A. Blackwell, R. Anderson, A. Grant, "The Memorability and Security of Passwords – Some Empirical Results", Tech. Report 500, Computer Lab, Cambridge, 2000. http://www.ftp.cl.cam.ac.uk/ftp/rja14/tr500.pdf.

27. P. Zimmermann, *The Official PGP User's Guide*, MIT Press, 1995.