Protected Login

Alexei Czeskis[1] and Dirk Balfanz[2] *

[1] University of Washington, Seattle, WA
[2] Google Inc., Mountain View, CA

Abstract. Despite known problems with their security and ease-of-use, passwords will likely continue to be the main form of web authentication for the foreseeable future. We define a certain class of password-based authentication protocols and call them *protected login*. Protected login mechanisms present reasonable security in the face of real-world threat models. We find that some websites already employ protected login mechanisms, but observe that they struggle to protect first logins from new devices – reducing usability and security. Armed with this insight, we make a recommendation for increasing the security of web authentication: reduce the number of unprotected logins, and in particular, offer opportunistic protection of first logins. We provide a sketch of a possible solution.

1 Introduction

The overwhelming majority of user authentication on the web today uses passwords. Perceiving passwords as weak and hard-to-use, the research community has in the past focused their efforts on replacing them with "better" authentication mechanisms [2, 5, 6]. Recently, Herley and van Oorschot [7] have suggested that such research is misguided because passwords, they argue, present a sweet spot in usability and security that is hard to match. Instead, they call for a research agenda that embraces passwords. We agree with much of the premise of Herley and van Oorschot's paper. In particular, we acknowledge that from a human-computer interaction point of view, passwords are hard to beat, and are likely to stay with us. We also agree that much confusion around password usage still exists that warrants further research.

We disagree, however, with the idea that "fixing" passwords is (probably) not necessary because the overall harm done by using them is (probably) small.[3] While the overall harm of account hijackings may be small, the blow to an individual's life when their account is hijacked can be devastating [4]. We take the opinion that every hijacked account is one hijacked account too many. Passwords should be fixed *now*, no matter how small the global harm done is due to their insecurity.

In fact, some websites have already begun to "protect" password-based logins in ways such that passwords alone are not sufficient to authenticate. Interestingly, this happens without affecting the user experience: users still simply enter a password, but their browser submits a cookie along with the password to protect the login.

In this paper, we study this phenomenon and suggest a generalization: we formally introduce the notion of "protected login", which is a class of authentication mechanisms that add credentials to the authentication flow that are invisible to the user. Because they are invisible, the user cannot be phished for them.

We have chosen the name "protected login" consciously: just like in physical human relationships, users can engage in "protected" or "unprotected" logins, and the two mechanisms are essentially identical: adding protection does not fundamentally change the way login is performed. When protection is unavailable, login still works, but the risks may be higher.

The rest of the paper is organized as follows: After reviewing real-world threat scenarios in Section 2, we formally introduce the notion of protected login in Section 3 and examine how websites use protected login today. We describe that the "Achilles' heel" of contemporary protected login mechanisms is the first login from a new device, both in terms of security and usability. In Section 4 we therefore explain how opportunistically adding protection to first logins can address this problem. We conclude in Section 5.

* The opinions expressed here are those of the authors and do not necessarily reflect the positions of Google.

[3] To be fair, their main point is that we can't currently quantify the harm done, but the implication is that if little harm is done then no corrective action is needed.

2 Practical Threats to Authentication on the Web

Based on our experience operating the authentication infrastructure of a site with hundreds of millions of users, we start by providing what we believe to be a real-world threat model for the majority of users authenticating on the web today. Most notably, we assume the attacker is capable of stealing user credentials (*i.e.,* passwords) through phishing or through compromising poorly protected web servers. Users are known to re-use (or share) passwords across websites, therefore a credential stolen from a poorly protected, unimportant web server may, for any given user, very well turn out to be also a credential for that user's banking or email provider [9].

Throughout this paper, we assume that the attacker is not controlling malware on the user's machine (although the careful reader will notice that some of the authentication mechanisms discussed here are secure against certain kinds of malware, such as keyloggers).

We'll furthermore assume that the authentication protocol runs over a secure connection, and the attacker is not able to steal cookies, passwords, or other credentials by eavesdropping on the messages between a user agent and a web server (*e.g.,* by compromising the Public Key Infrastructure [11] and becoming an "SSL man-in-the-middle"). It is possible to relax this assumption and design non-bearer-token-based authentication protocols that are secure against such attackers, but outlining this in detail goes beyond the scope of this position paper.[4] For the purposes of this paper, we'll assume that the attacker obtains possession of the user's credentials by phishing for them, or by breaking into poorly-protected websites.

3 A New Paradigm: *Protected Logins*

A class of login mechanisms that we call "protected login" is well-positioned to mitigate against the threats outlined above. We begin with several definitions:

Definition 1. User-supplied credentials *are passwords or other secrets that users input into client devices in order to authenticate to web servers. A login to a web server that is authenticated only through user-supplied credentials is called an* **unprotected login**. *Any login that is not an unprotected login is a* **protected login**. *A login mechanism that allows the web server to distinguish between protected and unprotected logins is a* **protected login mechanism**.

Several observations follow:

- User-supplied credentials are subject to phishing attacks and theft from poorly managed servers.
- A protected login involves credentials beyond just user-supplied credentials. Because these additional credentials are never supplied by the user (and presumably not even known to the user), the user cannot be phished for them.
- A credential thief must always perform an unprotected login to gain access to a victim's account (because the attacker is only ever able to obtain – through phishing or server compromise – user-supplied credentials[5]).
- A legitimate user may or may not have to perform unprotected logins.

From the point of view of a web application, protected logins are less risky than unprotected logins, which can be initiated by an attacker after credential theft. As such, it seems natural to require additional user-supplied credentials (such as mother's maiden name, one-time-PIN, *etc.*) during unprotected logins, while requiring only passwords during protected logins. Observe that the additional user-supplied credentials are still phishable and subject to theft.

We often assume that more "secure" mechanisms must be less "usable". Note the seemingly counter-intuitive consequence of our definition: risky unprotected logins, which require manual entry of additional user-supplied credentials, tend to be, in practice, less user-friendly than safer protected logins, which use supplementary credentials such as special cookies, but require from the user at most a password. We will provide examples for this below.

[4] A hint as to how this might work: one way to make stolen credentials useless to eavesdroppers is to *channel-bind* [10] them to an *origin-bound client certificate* [1]. Coincidentally, this can also protect the credentials from malware theft, especially if the client's TLS private key is protected by hardware.

[5] This assumes a minimum of competence on behalf of the web server - they need to design their non-user-supplied credentials such that they are distinguishable from *other* servers' credentials. If they do this, then the credentials stolen from another web server won't be usable for a protected login.

3.1 Bootstrapping Protected Login: Current Best Practices

Although they may not use this terminology, some web applications are already using protected logins today. We will now examine a few examples. All of them *bootstrap* protected logins using different types of unprotected logins. This is a problem that we will later return to.

FACEBOOK LOGIN NOTIFICATIONS

Facebook allows users to opt into a mechanism called "Login Notifications" [3]. When users have Login Notifications turned on, the first time they log in from a new device they are asked to "name" that device. The user is then notified of the (unprotected) login via SMS or e-mail which contains the device's name. Facebook associates the user-agent's HTTP session with that user and device name (presumably by setting a cookie[6]).

Observe that the *first* login from a new device is an unprotected login because it uses only user-supplied credentials. Because this is risky, Facebook asks users to provide a device name and sends them a login notification. In certain circumstances, the user will have to answer even more challenges, such as identifying a known person from a given image. However, *subsequent* logins from the same device require the user to only supply his username and password and do not generate notifications. Facebook is able to give subsequent logins a higher trust rank because, under the hood, subsequent logins include the user's password and a browser cookie (for which the user could not have been phished, and which could not have been stolen from a non-Facebook web server). Therefore, subsequent logins are protected logins, and a credential thief will always cause at least one unprotected login when accessing a victim's account.

GOOGLE 2-STEP VERIFICATION

Google allows users to opt into a mechanism called "2-Step Verification" [8], which is a form of two-factor authentication. Users obtain a one-time code (OTC) through SMS or from a smartphone app, and must enter this short code during login (in addition to their password). Just as with Facebook Login Notifications, users must perform this step the *first* time they log in from a certain device. *Subsequent* logins don't require an OTC – they only require a password. Google can do this because, just as with Login Notifications, subsequent logins are protected by a cookie that is set during the two-factor login and is sent along with all subsequent logins.

Again, observe that the *first* login is unprotected according to our definition[7] because an attacker can phish the user for their password and OTC. As before, *subsequent* logins are protected. Similarly, after a credential theft, an attacker will always have to perform an unprotected login to access a victim's account.

QUORA LOGIN

Previous examples showed protected logins that required only passwords, and unprotected logins that required additional user action. Quora makes a different trade-off between protected and unprotected logins. The first time users log into Quora from a new device, the (unprotected) login requires a username and password. This is an unprotected login because a phisher can perform the same login once he obtains the user's credentials.

On the login page, Quora shows the user a checkbox that says "let me login without a password on this browser". Once checked, subsequent logins will not require a password – instead, the login page shows the user their profile picture, and a single click on that picture logs the user back in. Note that this login page with the user's profile picture is shown *after* the user has specifically clicked the logout button. Even though this login page doesn't require a password at all, this is a protected login (presumably affected by a cookie that was saved on the user's machine), because a phisher cannot cause this kind of login simply by stealing user-supplied credentials (*i.e.,* the user's Quora password) – he would first have to go through an unprotected login.

Some threat models would consider the protected Quora login less secure than the unprotected login (*e.g.,* if the threat is that of an attacker walking up to the user's terminal). In our threat model however, which we argue reflects real-world threats for the majority of internet users, this attack is not an issue.

[6] We haven't identified the particular cookie responsible for maintaining this state, but verified that cookies remain set after logout, and that removing all cookies results in the user having to "name" their device again.

[7] This doesn't mean that 2-Step Verification is a bad idea. In fact, it in practice affords vastly improved security to Google account holders, in part because it protects against password sharing.

3.2 Problems with Current Login Mechanisms

In the above examples attackers can bootstrap protected logins by first performing an unprotected login with stolen user-supplied credentials. Web sites have naturally sought to increase the security of such unprotected logins. The techniques we examined above (sending login notifications and using two-factor authentications) are examples of this.

Other popular methods of bootstrapping protected login include questions such as "what's your mother's maiden name?", "when was your father born", and "did you recently take out a mortgage?". These approaches are common and are used by many banks (*e.g.,* Bank of America and ING Direct) and credit history agencies (*e.g.,* Equifax).

The general approach of bootstrapping an easy-to-use, protected login with a more onerous, unprotected login, unfortunately has several problems:

- First, unprotected first logins lack *security* because they are fundamentally phishable. Even if the users are not phished, answers to security questions can often be found by determined attackers through publicly available records or social networks – Sarah Palin's personal Yahoo! e-mail account was "hacked" in this way [12].
- Second, current login flows also do not provide *availability* as users can easily be locked out of their accounts. For example, users often forget the answers to secret questions (what was the name of my favorite song 3 years ago?) As another example, second factor authentication users are at the mercy of the second device's availability. If they misplace it or let the device's battery die, users may be locked out of their account.[8]
- Most notably, all of the existing approaches to strengthen unprotected login sacrifice *usability* by introducing changes that slow down, inhibit, or otherwise worsen the user experience (UX). The degraded UX may, in turn, lead to user discontent and accordingly, companies like Google and Facebook are cautious to make many of the login protections mandatory – rather leaving them in the "opt-in" arena for more tech-savvy and security conscious users. This, in turn, leaves the vast majority of users without the protection provided by the disabled-by-default additional security measures.

Given that strengthening authentication for unprotected logins seems to severely degrade the user experience, how can we improve on the current best practices around web authentication?

4 Reducing Unprotected Logins

We believe the key to improving web authentication without sacrificing usability is to significantly reduce the number of unprotected logins. This would increase *security* because unprotected logins are by definition phishable, and hence less of them means less opportunity for passwords and other credentials to get compromised. If unprotected logins are very rare, websites can afford to "raise alarms" whenever an unprotected login occurs and legitimately treat those sessions as less secure. Websites would also be able to notify the users with more conspicuous messages, perhaps require them to take action, taint sessions initiated with unprotected logins, or even have infrastructure to revert changes caused by a potential attacker. Additionally, having less unprotected logins would allow immense *usability* benefits for many users. As we showed, unprotected logins tend to be onerous; therefore, fewer unprotected logins means fewer onerous logins.

Clearly, reducing unprotected logins would be beneficial from a security and usability perspective. However, can it be done without affecting availability? If so, how and where in the web authentication flow this be done?

4.1 Protecting Subsequent Logins

We noted earlier that some websites already offer protected logins to their users. They do this by setting a special cookie during the first, unprotected, login from a new device, and then checking that cookie on subsequent logins.

This much is simple: one way to reduce the number of unprotected logins is to implement this pattern in more places around the Internet: set a cookie when users first log in, and use the presence of that cookie as an additional signal to increase the trust rank of (subsequent) logins.

This leaves us with the problem of unprotected first logins, the "Achilles' heel" of current web authentication, both in terms of security and usability. Achieving a protected login during the *first* authentication from a new device has been elusive. Why? What makes protecting the first authentication so difficult? Is there really nothing we can do?

[8] Google 2-step verification users are asked to print out a set of "backup" codes that can be used in this scenario. Users are urged to carry these codes with them. Users can also provide backup telephone numbers where codes can be sent in case of a lockout.

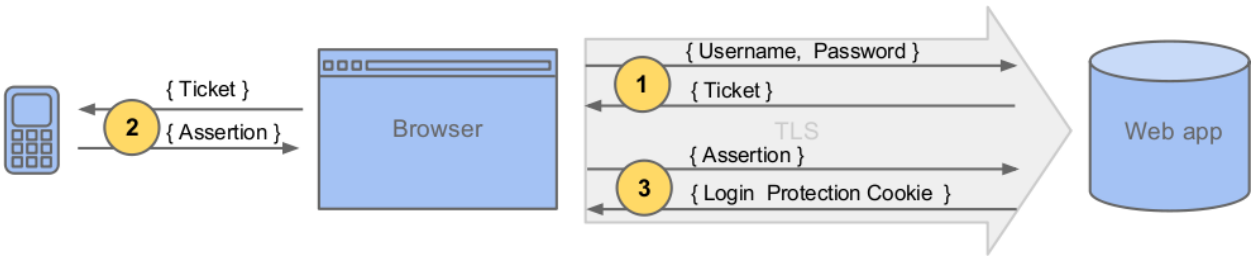

Fig. 1. Possible protocol for a protecting *first* login from new devices.

4.2 Protecting First Logins

To understand why protected login during first authentication is hard to achieve, let's consider what that process must involve. By definition, during a protected login, the user's client must transmit to the server a piece of data, for which the user cannot be phished, but which will convince the web server that a legitimate login is in progress. Where can this data come from? For secondary logins, the browser can send secret cookies, but on first login, the user's browser hasn't yet established user-specific secrets with the server and is unable to do so. Could the data come from the user? Unfortunately, any secret the user knows is fundamentally phishable. This indicates that for a first login to be protected, it must involve some second factor device. However, all second factor authentication mechanisms seem to degrade users' experience. Even smartcards, which can be used in ways that don't alter the authentication user experience, require users to carry additional hardware and are henceforth seldom used outside of the corporate and government settings. Therefore, we believe the biggest impediment to providing protected login during the first authentication is the lack of a second-factor based protected login mechanism that is largely transparent for users.

We believe that asking users to carry additional devices is unacceptable, so we are focusing our attention on mobile phones – the only additional piece of hardware that many users consistently carry with them. But even phones are often unavailable or non-existent, and relying on them is a recipe for failure.

We now offer a key insight and propose a security compromise: what if protected login is provided opportunistically? That is, the website will always ask the user for his password, but if it's possible and all of the "stars align", the authentication flow will involve an additional operation (that's transparent to the user), which will result in a protected login. However, if the protected login mechanism fails to successfully complete, the authentication will result in an unprotected login. In either case, the user will be logged in, and would have done no work beyond entering his password. This compromise will allow the system to maintain good *availability*. [9]

Of course, several questions immediately arise: How does one make the protected login mechanism transparent to users? How reliable will the protected login mechanism be and how often will users be forced into an unprotected session? What protocol should the protected login mechanism use and what type of data should it send – a certificate, a token? Finally, is it possible to provide a framework that's usable by any site on the web and does not involve significant developer effort?

These questions are complicated and have non-trivial tensions between availability, security, usability, accessibility, and privacy. We are in the beginning stages of building a system that attempts to navigate these various constraints.

Sketch of a Possible Solution

We now provide a high-level glimpse of our work-in-progress – an authentication system that opportunistically allows users to achieve a protected login from new devices without altering the user experience. We omit the majority of details as well as the discussion of why certain tradeoffs were made – those issues are complicated, non-obvious, and are still in slight fluctuation.

Our design assumes that users have a smartphone, which various web services can leverage to protect user logins. We also assume that at some point, the phone and web service were able to perform a key exchange. The core idea of our design is described in Figure 1 and works as follows. First, the user navigates to a website of his choice and is

[9] Nevertheless, opportunistic protected login may be insufficient to meet the security needs of some organizations; they may choose to deploy a policy that will enforce mandatory protected login for some or all transactions.

presented with a login page (as usual). He enters his username and password. These user-supplied credentials are sent to the server, which authenticates the user and responds with a *login ticket*.

Next, the web browser sends the ticket to the user's phone over a wireless distance-bound protocol. The phone verifies the ticket, generates an assertion and hands it back to the browser. The browser then sends the assertion to the server, which can verify it based on the previous key exchange with the phone. The server then responds with a cookie[10] and marks the HTTP session as protected. If the user's phone is not reachable within a small delta of time, the browser cannot send the phone's assertion. Instead, it sends an error message to the server, which will mark the session as unprotected, but still responds with a cookie.

We leave all further discussion of this proposal as future work, but would like to note that a successful protected login under this protocol protects users against phishing and password theft. Further refinements that utilize channel-bound assertions and cookies also protect against SSL men-in-the-middle attacks and cookie theft [1, 10].

5 Conclusion

Passwords have been the primary authentication mechanism since before the web was born, and they don't seem to be going away. At the same time, attackers have become adept at stealing passwords through a variety of attacks. In this paper, we made several key contributions. First, we presented a real-world threat model for web authentication and password use. Second, we introduced a new paradigm, *protected login*, that is useful for analyzing various web authentication techniques. Third, we used the protected login paradigm to examine current web login flows and find that many web applications already deploy protected login, but still make themselves vulnerable by forcing users to bootstrap protected login through unprotected login from new devices. Fourth, we gave an agenda for improving web authentication: reduce unprotected logins by opportunistically protecting *first logins* with a protocol like the one outlined above and by protecting *subsequent logins* with mechanisms similar to what we already see in the wild today.

References

1. D. Balfanz, D. Smetters, M. Upadhyay, and A. Barth. TLS Origin-Bound Certificates (Working Draft), July 2011. http://tools.ietf.org/html/draft-balfanz-tls-obc.
2. Katherine M. Everitt, Tanya Bragin, James Fogarty, and Tadayoshi Kohno. A comprehensive study of frequency, interference, and training of multiple graphical passwords. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 889–898, New York, NY, USA, 2009. ACM.
3. Facebook. What are Login Notifications?, 2011. https://www.facebook.com/help/?faq=162968940433354.
4. James Fallows. Hacked!, 2011. http://www.theatlantic.com/magazine/archive/2011/11/hacked/8673/.
5. Alain Forget, Sonia Chiasson, and Robert Biddle. Shoulder-surfing resistance with eye-gaze entry in cued-recall graphical passwords. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1107–1110, New York, NY, USA, 2010. ACM.
6. Sebastian Gajek, Jrg Schwenk, Michael Steiner, and Chen Xuan. Risks of the cardspace protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Ardagna, editors, *Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 278–293. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-04474-8_23.
7. Cormac Herley and Paul van Oorschot. A Research Agenda Acknowledging the Persistence of Passwords. *IEEE Security & Privacy Magazine*, 2011.
8. Google Inc. Getting started with 2-step verification, 2011. http://goo.gl/5r8Za.
9. John Leyden. Anonymous hack showed password re-use becoming endemic, 2011. http://www.theregister.co.uk/2011/02/10/password_re_use_study/.
10. N. Williams. On the Use of Channel Bindings to Secure Channels. RFC 5056, RFC Editor, November 2007. http://www.ietf.org/rfc/rfc5056.txt.
11. Kim Zetter. Diginotar files for bankruptcy in wake of devastating hack, 2011. http://www.wired.com/threatlevel/2011/09/diginotar-bankruptcy/.
12. Kim Zetter. Sarah Palin E-mail Hacker Sentenced to 1 Year in Custody, 2011. http://www.wired.com/threatlevel/2010/11/palin-hacker-sentenced/.

[10] This cookie is what protects subsequent logins, for which presence of the phone is no longer required.