Replacing passwords on the Internet
AKA post-Snowden Opportunistic Encryption

Ben Laurie
Google, Inc.

Ian Goldberg
University of Waterloo

January 18, 2014

1 Motivation

Just as DigiNotar gave us an opportunity to re-examine PKI as applied to the Web, Snowden gives us an opportunity to re-examine the more general question of encryption and authentication on the Internet.

In particular, it seems very clear that nothing should be plaintext anymore. The most obvious thing to do is to adopt opportunistic encryption (for the academics: unauthenticated forward-secret encryption). But this provides no defence against active attackers—which is not to say it is not useful: raising the bar to "must MitM" adds substantial cost for the attacker, and also exposes him to potential detection. Indeed, we encourage standards bodies to never again approve a protocol that sends plaintext over the Internet: unauthenticated encryption is straightforward to add to any online protocol, and is strictly safer than unencrypted data. Crucially, however, implementations of these protocols should *not* claim to users that such an unauthenticated connection is secure—it should be a drop-in replacement for plaintext, not for a fully secure authenticated connection.

In the TLS context, this amounts to encouraging the proliferation of self-signed certificates (or organization-signed, but not necessarily with a trust chain to a trusted root). Crucially, in this model, browsers must no longer display a scary warning message when they encounter self-signed certificates. Instead, they should, as above, simply claim the connection is *insecure*, presenting the page and chrome to the user identically to an unencrypted HTTP connection. This is reasonable because a self-signed TLS session is actually *more* secure than a plain HTTP session (it is secure against passive, though not active, attackers), so long as the user is not deceived into thinking she is using a (fully) secure connection. This means, additionally, that whenever the browser makes a decision depending on whether the page is secure or insecure (such as sending "secure" cookies), a self-signed TLS connection should be treated as insecure. In this way, we can encourage the proliferation of self-signed certificates on the Internet in a manner that replaces the current uses of plain HTTP, but not displacing the role of CA-signed certificates.

Can we do even better than pervasive unauthenticated encryption? We think we can. The important observation is that most traffic on the Internet of value is already authenticated in some way in one direction, most often by some kind of username/password. We can leverage that to provide authenticated encryption in a pretty general way, by using PAKEs. [2]

However, this leaves the problem that most passwords are weak, most passwords are used at multiple sites and phishing is easy. Therefore, we now need to improve the state of passwords. Once we have done that, we have a complete solution; further, it can be deployed *incrementally*.

2 Details

2.1 Passwords

First of all, we clearly need to replace weak, re-used, phishable passwords. The first component is a password store. This must be run by the OS because, contrary to popular belief, there is Internet outside the browser. For example, it would not be good to replace passwords on gmail only to find that I can no longer run Thunderbird+Enigmail to read my PGPed email.

The password store needs to operate like Plan 9's factotum [1]. That is, it does not provide passwords, keys, shared secrets, etc., to applications; instead, it provides the appropriate messages to them in order to authenticate to their peer (and authenticate their peer), keeping passwords safe within it. This allows us to later migrate the store to a TPM, another machine [4] or some other sanctuary from malware on the user's machine.

Furthermore, the password store needs to support sync. It must be possible to enroll all my devices, including a brand new device I purchased after I dropped all my existing ones in the ocean. Unless someone has a great new idea, this requirement means the password store must be unlocked with ... a password. [3] Moreover, to ensure the user does not forget this password, she must be required to use it frequently. Say, every time she unlocks a device, or the first time she authenticates after unlocking.

Since this One True Password (1TP) now carries great value, it must be carefully protected. It must be very clear to users when the 1TP is needed and where it should be entered, and it should not be possible for a phisher to fake that. This is undeniably a challenge—probably the biggest challenge in this entire plan.

2.2 Encryption and authentication protocol

The encryption protocol need not be anything novel—it can be as simple as unauthenticated Diffie-Hellman followed by a stream cipher. Once the unauthenticated connection is established, the protocol should be able to optionally support additional authentication using a PAKE (keyed on a combination of the Diffie-Hellman shared secret, an identifier for the server, and the client's authentication information) so that when a client authenticates to a server with which it has an existing relationship, it also authenticates that server.

Again in the TLS context, a possible option is TLS-SRP. [5] Something TLS-like can be desirable, because we want to be able to upgrade opportunistic encryption to authenticated encryption when possible. In this way, the protocol has to allow the use of something certificate-like to authenticate the endpoints, and something CA-like to authenticate the certificates.

2.3 Migration

Where a client and server have an existing relationship secured by username/password, it would be nice to be able to move seamlessly to the new scheme. If the server has appropriate hygiene, this means standard PAKEs need to be slightly modified in order to allow the server to construct the shared secret (for a symmetric PAKE), or the authenticator (for an asymmetric PAKE), and also to bind the PAKE to the unauthenticated encryption channel.

For example, SRP uses $g^{H(p\|u)}$, but you might instead use $g^{H(s\|H_2(p)\|u)}$, where p is the password, u is the URL of the server, s is a value derived from the Diffie-Hellman shared secret, H is a hash function, and H_2 is the hash function the server already uses to check passwords.

A challenge here is that many servers use ad hoc hashing mechanisms to store password authenticators, so we may need to enumerate all the possible schemes (required because the client needs to know which scheme to use).

The password store should also upgrade passwords to strongly random ones—or move away from passwords altogether and start using public/private keypairs for authentication.

All of this can clearly be done incrementally—old clients can continue to authenticate directly with their passwords. New clients can use Diffie-Hellman and a PAKE to get a connection that is both encrypted and authenticated.

3 Conclusion

As is often the case, improving security is not a matter of inventing new crypto or new protocols. Instead it is a matter of user interface and user experience. If only we can finally stop giving users the utterly impractical advice that they should use different, strong passwords at each site and never write them down, and instead provide them with powerful and usable tools for authentication, we can fix phishing *and* make it much harder to intercept the plaintext of users' interactions on the Internet.

References

[1] Russ Cox, Eric Grosse, Rob Pike, Dave Presotto, and Sean Quinlan. Security in Plan 9. In *11th USENIX Security Symposium*, pages 3–16, August 2002.

[2] David P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):5–26, October 1996.

[3] Ben Laurie. Nigori: Storing Secrets in the Cloud. http://www.links.org/files/nigori-overview.pdf, May 2010.

[4] Ben Laurie and Abe Singer. Choose the red pill *and* the blue pill: a position paper. In *NSPW*, pages 127–133, September 2008.

[5] David Taylor, Tom Wu, Nikos Mavrogiannopoulos, and Trevor Perrin. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054, http://tools.ietf.org/html/rfc5054, November 2007.