Hashing at Home

My daughter called me over for permission to register at a kid’s multiplayer game site. This naturally piques my interest, and I give these sites more than a cursory look. This site seemed to be well designed: no personal information about the child is even stored on the site. Instead, they store a hash of the child’s registration information, and compare logins with the hash (the login is done by a cookie, of course).

In the event of a lost password, the child has to reenter the personal information, including an email address, and a new hash is computed. If this matches the stored hash, a new password ticket is sent by email.

Review & Preview

We discussed the analogy between a ticket and a bus transfer. A bus transfer is valid over a whole transit system (with exceptions, although in Sydney a bus transfer gets you onto a commuter boat, offering a lovely free tour of the harbour), and is a tangible indication that you, as a rider, are “validated” (that is, paid up). Transit systems have developed simple but effective schemes for ensuring transfer validity and for recording the time stamp. Some systems time stamp the transfer as you pick it up; others punch holes indicating the end of the valid time; others use a scheme by which part of the transfer is torn off, so the stub indicated the expiration time.

This reminds me of a story, from before 2001 sometime. My wife and I were flying back from a vacation, and had a very long layover in Salt Lake City. We took the bus into town, had a nice lunch, did some shopping, and came back to the airport in plenty of time. When we presented ourselves to security, we were waved (or should I say “waived”?) right through, because “… you’ve already been checked.” This was true, but we had ridden on a city bus full of crazy people. the security authentication had no time stamp and no expiration (I suppose it expired at midnight).

When this happened more recently, I took a taxi to Red Iguana and got takeout, which I hand-carried through the same security line.

You probably already have some experience with the authentication-by-ticket process from recovering a lost password. Typically, you click on “Forgot My Password”, and enter an email address. In the bad old days, the site would send an email with your password in the clear. The probability of a sniffing attack detecting your password in this case is very small, but not zero. This was the problem with the first authentication scheme we looked at on Thursday.
A slightly more sophisticated approach is to send you a password hint, which you left with
the server at the time of your last password creation. But a hint, is a hint, is a hint, and
an attacker who knows you, or who has access to some information about you, might be
able to guess the password.

An even more sophisticated approach is the authorization email that includes a link to the
password reset page. The link might look something like

\[ http://www.phishing.com/foo.html?tick=a1b2c3d4e5f6 \]

The question mark in the URL is the HTML method for passing a variable. In this case,
the value passed is the ticket. Presumably, the site is only willing to accept this ticket
from one network address (IP address, in this case).

### Ticket Granting Server

The ticket granting server (TGS) is distinct from the service granting server (SGS). The
TGS has a password $P_C$ (public or private) for each client $C$. The authentication process
involves two phases: in the first, the client asks the authentication server AS for a ticket,
which has a finite lifetime, like a bus transfer.

\[
C \text{ to } AS: \quad ID_C \| ID_{TGS}
\]

\[
AS \text{ to } C: \quad T_{TGS},
\]

where $T_{TGS}$ is sent encrypted with the client’s key. This means that only the client (or
someone who has stolen the client’s key) can use the ticket. The ticket $T_A$ is encrypted
so that only the TGS can read it. It contains the client ID, client address, TGS ID, a
timestamp, and a lifetime.

Now, when the client needs a service, she sends the ticket to the TGS along with an
identification for the requested service; the TGS sends the client a ticket $T_V$.

\[
C \text{ to } TGS: \quad ID_C \| ID_V \| T_{TGS}
\]

\[
TS \text{ to } C: \quad T_V
\]

THE TGS examines the ticket to see if it is still valid. The ticket $T_V$ is encrypted with
the service’s key, preventing a nefarious client from using an authorization for one service
to use it for another. It consists of the client ID, client address, service ID, a timestamp,
and a lifetime.

Finally, the client sends the ticket to the server, and the server grants the service.

\[
C \text{ to } V: \quad ID_C \| T_V
\]

Everything has a timestamp and a lifetime. There is a tradeoff here: a long lifetime for
the TGS ticket means that the client does not need to authenticate very often, but it
means that the TGS ticket and the TGS key will be used more, leading to more plaintext-
ciphertext, or at least partial plaintext-ciphertext, pairs.
This protocol is still subject to replay attacks. Furthermore, there is no authentication of the servers, so an opponent could hijack a server request and gain access to various tickets. The Kerberos system addresses these issues.

**Kerberos, Version 5**

Kerberos Version 5 is a sophisticated and complex system; its advocates would argue that this is an example of necessary complexity.

\[
\begin{align*}
C \text{ to } AS : & \quad \text{Options } || ID_C || Realm_C || ID_{TGS} \| Times \| nonce \\
AS \text{ to } C : & \quad Realm_C \| ID_C \| T_{TGS} \| Auth
\end{align*}
\]

The client now has the ability to request several options. The most intriguing of these is “FORWARDABLE”, allowing the client to specify that the ticket may be forwarded to another server for associated services (for example, a combined print server – SQL server request to generate and print a report). The *nonce* is a random number that identifies this pair and prevents replays.

The user can use the *Times* field to request a lifetime, or a renewal.

The ticket is encrypted with the TGS key, and includes the flags (allowed options), the session key, the realm, the client ID, the client address, and the times.

The “Auth” field is encrypted. The contents are the flags, a session key for both the TGS and the client, the Realm, the client ID, the client address, and the Times field. This is encrypted with the client’s key, so is of no use to an eavesdropper.

While a service request is broadly similar, its main innovation is the inclusion of a session key which is sent to both the client and the server. The granting of the service request is encrypted using the session key, which is sent to the server encrypted with the server’s key. Thus, a spoof server who does not have the server’s key cannot send a valid grant of service.