attempt at unified ,quick collection of somewhat more developed ideas on the subject of authentication at a level relevant for the introsec course. This is not a complete document ,but it's hoped this first draft suitable as course material to get students thinking where we can go in terms of subject matter

What might be suitable improvements on simple password based authentication?

This is an early attempt at a unified, quick collection of somewhat more developed ideas on the subject of authentication at a level that is relevant for the IntroSec course. This is in no way a completed or honed pedagogical document, but it is hoped that even this first draft is suitable as course material to get students thinking in terms of where we can go with this subject matter.

Passwords are most widely used authentication method today, but they are far from most reliable of methods. Other basic classes of authentication that we look at are problematic too.

During the course we reach a point of insight that though passwords are possibly the most widely used authentication method today, we can see that they are far from the most reliable of methods. Unfortunately, we can also see that the other basic classes of authentication that we look at are not unproblematic. For example, “Something the entity has” is also problematic if the thing you have is easily lost or easily taken away from the user. “Something the entity is” such as biometric methods have many problems associated with them, so though we may gain a sense that they feel good and secure and easy to use we should no doubt check ourselves and ask where that sense comes from and if it really is justified. The subject of problems with biometric methods has been broached in the course videos.

So after looking at all of the basic and relatively obvious authentication methods, we seem to be left with the question: How can we go about improving these methods? If I really need strong authentication (one should not always assume that authentication does need to be strong – as ever it depends entirely on the situation) what can one do to improve things?

If part of the problem is users are required to choose and remember too many passwords to be able to cope with requirements put on them we may be able to reduce atleast burden of remembering with use of technology.

Solving password problems

If part of the problem is that users are required to choose and remember too many passwords to be able to cope with all the requirements that one puts on them, then we may be able to reduce at least the burden of remembering with the aid of technology. N.B. the following three categories of *password rememberers*, *password managers and keyrings*, and *single sign-on* are Alan’s terminology that attempts to establish a conceptual differentiation between more or less distinct ideas. Other sources may have different terms and classifications of these phenomena.

Browser based “password rememberers”

When you type a password into a system’s web interface your browser will commonly notice that the field is marked as a password field (i.e. the kind that should not show what you are typing as you type it) and offer to remember it for you, as well as what system it was that the password was for. That way, when you return to that website the browser can automatically insert your password into the appropriate field. As we saw during assignment 1 this is a weak strategy if anyone can ever access the browser that you use, since they will immediately gain access to a variety of services in your name. If the browser is not programmed to carefully handle password field it is even possible to reveal the passwords that you use, and thereby possibly learn what password strategies you use.

Plant a cookie with browser containing data the service provider thinks is sufficient to identify you.

Another somewhat similar phenomenon is that those who are running the server side service offer to “remember you” once you have logged in. This most likely means that the next time you return to this website the server will automatically authenticate you so that you avoid the password dialogue. The normal method of doing this is to plant a so called *cookie* with your browser containing whatever data the service provider thinks is sufficient to identify you. Servers are allowed to plant and retrieve such cookies silently in the background, unless you have deliberately restricted these rights in your browser configuration. In such situations the security of the authentication hinges on how the server provider chooses to implement it in your cookies. It has been known for sites to save your username and password in clear in a cookie in order of save you from re-typing them.

We can question the use of such functions for critical passwords since they can at worst be automatically writing the password down for you, so to speak. If your computer or your network communications are compromised then this may give others easy access to all the sites for which you have saved your password. It may sometimes be difficult to distinguish this class from the following one, so it you want to remember a password in this way and are not sure if it is the browser or some kind of password manager that is asking to remember it for you, you might do well to err on the side of caution.

Password managers and keyrings

A slight improvement on the previous class can be a carefully crafted system-wide password manager that can fill in remembered passwords over a variety of applications. The extra element of security is in that these tools are usually designed to be cryptographically well protected, unlocked by a single password, or other authentication mechanism. Other sensitive data, such as private keys, and certificates (i.e. the kind that contain private keys in this case) may also be kept by such a scheme. If the tool is well implemented then passwords and keys are decrypted and deliberately held in primary memory for the shortest possible time, so as to limit the opportunities for a malicious tool to search for sensitive data in memory.

This kind of tool will often be implemented to link to the authentication method that is used to identify you to the system, so once ‘logged in’ your system may be extra vulnerable. There are ways to limit the vulnerability, such as by invoking tools that time-out inactive sessions and require re-authentication, to guard against you leaving a system open and vulnerable as you leave your device unguarded for a moment.

Now that so many users tend to use a multitude of digital devices, cloud based password managers and keyrings are becoming more common. This adds the extra level of convenience that entering a password for a specific tool in one computing environment can allow that password to be stored to the cloud and then retrieved for any other linked device to be used for the same kind of service.

It should be clear that the security of such services are entirely dependent on how well they are implemented, and if the source code of that implementation is not available for inspection, then we also rely on how much one can trust the implementor.

The wikipedia page https://en.wikipedia.org/wiki/Password_manager can give a sense of what such tools might look like, even though that page's emphasis differs slightly from this presentation. The wikipedia page on https://en.wikipedia.org/wiki/Keyring_(cryptography) is unfortunately barely any help. The page https://en.wikipedia.org/wiki/Keychain_(software) describes a specific propriety product that can be read as a typical example of this kind of phenomenon.

Single sign-on

Explanation yet to be completed.

e.g. Shibboleth and Kerberos

https://en.wikipedia.org/wiki/Single_sign-on initially gives a very broad definition of Single Sign-On systems though then narrows the description to more closely differentiate this phenomenon from the previous two.

In our view of the concept *single sign-on*, these are systems where there are no separate passwords to several different systems, but the user authenticates to a central authority who then verifies your authentication to other dependent services. This means that services have to be specifically adapted to allow any one of these models of cooperative authentication.

Note how single sign-on puts great demands on the security of the single authentication that gives us the right to access several diverse services. To propose one small example - Kerberos documentation is careful to never use the term *password*, preferring *pass phrase*, thereby encouraging users not to create the same kinds of problems one commonly sees in passwords for this one, critical authentication.

The demands put on single sign-on and federated ID systems for strong and safe authentication mean that they are often based on more complex authentication formats than just "something the entity knows", as discussed below.

Federated IDs

Explanation yet *to be* completed.

Federated ID is similar in function to single sign-on but where the identity may be trusted across several organisations. Things can become complex here, for example when different organisations have different policies on what are sufficient factors to identify an entity, and yet hope to collaborate on the authentication.

E.g. Swedish Bank ID, FaceBook Connect, Microsoft account or MSA (previously known under other proprietary names such as Microsoft Passport and Windows Live ID)

One-time passwords

A completely different take on securing passwords is *one-time passwords*. As the name implies, this involves using a password once, but once used that password is invalidated. The course book has a short section describing this in section 13.5.1 on page 436.

General Authentication Strategies and Protocols

The following two methods can be applied to all classes of authentication and provide additional security, though with some risk that the processes may become more cumbersome for the user.

Multi-factor Authentication

See the course book, section 13.9 p446 .

Challenge-response

See the course book, section 13.6 p438

Though classic examples of challenge-response generally involve “something the entity knows” it should be clear that it is not dependent on the class of authentication method.

- Something the entity knows (e.g. What is your mother’s maiden name?)
- Something the entity has (e.g. a token. A proffered code is entered into a token which then processes that code and in turn produces a code to return to the authenticator as proof that that individual token is being used.)
- Something the entity is (or can do) (e.g. shibboleth, i.e. not the web based single sign on system names Shibboleth, but in its original meaning. See Wikipedia on Shibboleth)
- Somewhere the entity is (uncommon, but quite feasible. Invent your own example?)

In its simplest and most trivial form one might even classify a password dialogue as challenge-response. Party A requests authentication to system S, S requests a password, and A supplies it. This trivial example gives us a clue as to what makes a weak versus a strong challenge-response protocol. The password scenario is very weak challenge-response in that an attacker knows that the challenge is to provide a password and can reuse that password in case they have managed to eavesdrop it. If however the attacker has a harder time predicting what the challenge will be, then they will have less opportunity to use any eavesdropped responses. It the authenticator can collect much information about a user they could choose among a set of possible challenges, such as: “what is your mother’s maiden name?”, or “what was the name of your first pet?” etc. This kind of challenge-response scenario is discussed in the course book under the name Security Questions, p 52.

To some degree, a challenge-response protocol can be used for mutual authentication, i.e. that both parties can be assured that the other is authentic in the one transaction. If the two parties secretly agree beforehand on exactly what constitutes a legitimate challenge and what the answers to those challenges should be, then an imposter may be revealed when they try to ask an invalid question. Note however that other less cumbersome protocols for mutual authentication also exist.

At its strongest, an attacker should have no reasonable opportunity to predict what the challenge will be, and therefore not have any clue what the response would be, nor be able to use earlier eavesdropped responses in a replay attack. One way to achieve this is for two parties to agree on a secret mathematical function. A unique key could be a part of this function. The challenge might be

a 6 digit decimal number chosen at random. That would mean that an attacker would have one chance in 10 million to guess what the challenge will be. The user then applies the function (if necessary with the use of that unique key) and returns the function output. The authenticator applies the same secret function to the random number and if the output matches that sent back from the user, the user is authenticated. You will notice that this is the method used with token device above. One can also see that this should be a cryptographic function where it is not reasonable to discover what the function is even after eavesdropping many challenges with corresponding responses.