The Authentication Dilemma

Michael Scott

MIRACL Labs
mike.scott@miracl.com

Abstract. The Internet community is up in a heap about Username/Password, and what to replace it with. Here we try to shed a little light.

1 What can Hackers actually do?

They can by sneaky methods plant viruses on your computer and, to varying extents, take control of it. Often this is done by fooling you into opening an email attachment. Sometimes they can exploit bugs in the software to remotely break into your computer without any recourse to your foolishness.

Using such methods they can break into servers and steal files from the hard disk.

Less well understood is what they cannot do. They cannot steal important data out of the memory of running processes, unless there is a particularly stupid software bug – the SSL Heartbleed exploit comes to mind [2] – that gives it to them on a plate. If they could, then it would be game over. Such a hacker could simply remotely reach into your computer and grab all of your secrets.

2 Username/Password – is it completely broken?

Well yes and no. First a really simple explanation of how Username/Password works. The User enters their Username and their password P. These are transmitted to a server who has stored against each user H(P), where H is what we call a Hash Function. A Hash function simply converts the password P into a meaningless looking number. So H("pass1234") would look something like 282742829472957297925725. When you log-in your password is hashed and compared with what is stored in the password file. The idea is that the server does not store your actual password, but rather a hash of it. In reality something called a Salt is involved as well, but we don't need to muddy the waters with that. The server implements a policy of only allowing a small number of incorrect password guesses before it locks down the account.

Problem: As explained above hackers have proven adept at grabbing the password files which contain everyone's Username and the hashes of their passwords. Then using fast computers to guess passwords, they can use the same hash function to look for matches. So unless your password is really obscure, they will figure it out.

Solution 1: Force users to use complicated password. This "solution" simply accepts that password files will be stolen. The server has basically given up on trying to secure itself and is telling users "look our password file will eventually be stolen, get over it, and to make sure the robbers can't figure out your password you had better choose one that they won't be able to guess using even the most powerful computers". WTF? The server might as well post its password file on its website!

Solution 2: The server accesses a Univeral Secret S, and now stores H(P,S) against each user. So now the Universal Secret is mixed in with the password prior to hashing it. This Universal Secret is really big, random, and unguessible. It is not stored on a file, but rather loaded directly into the server process memory from some other secure resource. Every time a user tries to enter the site using a password P', then H(P',S) is compared with the stored H(P,S) and only if they match is access allowed. A hacker who grabs the password file does not know S and so is stymied. This is a much better idea, and its how Facebook for example does it. However all of our eggs are in one basket - S better not leak out! But as pointed out above Hackers lack the capability to grab secrets out of process memory.

The big advantage of Username/Password is that its terribly convenient. You carry your password around with you in your head. The big problem with Username/Password is that its one-factor authentication. If some-one can steal or shoulder-surf your password, you are screwed. And of course (if you are like me) you are using the same password in many different server contexts, some of which may go to greater lengths than others to protect it.

3 Multi-Factor Authentication

So next up, whats all this about Multi-Factor authentication? This is about providing much stronger authentication. For maximum security you should ideally authenticate with something you know, something you have, and something you are. Typically that would be a password, a physical device of some sort, and a biometric like a fingerprint. For the moment the industry appears to be content with just two factor authentication, which could be any two out of these three.

This is potentially a lot more inconvenient. Certainly of the three factors the password is the simplest to use. The "what you have" factor, well you might lose that or forget to bring it with you. The "what you are" factor, well if this can be cloned you can't exactly change it like a password.

A really obvious idea is that the "what you have" factor is your mobile phone, something you value and are likely to protect. And you have it with you all the time anyway. And maybe it already supports fingerprint identification! So all of the components necessary for multi-factor authentication might already be right there in your pocket.

A common starting point is that you should actually authenticate using a cryptographic secret that is unique to you. But to access this secret you must be able to contribute all of your factors. When it comes to authenticating you use

this unique secret in some calculation that proves you have it (and that therefore you are who you say you are), without revealing it. These strong cryptographic techniques are outside of the scope of this article, but they are much stronger than the simple hashing used in Username/Password.

4 The FIDO approach

The FIDO alliance approach [1] is that you generate your private authentication secret yourself and then get the server to accept your credentials and an associated public key. These public keys are stored in a file somewhere on the server, next to your identity. Your private key is stored on your phone which becomes one of your factors, protected in some way by your password or biometric as a second factor.

For this to work your mobile phone must support a secure zone into which your biometric template and your authentication secret can be safely stored, in such a way that if your phone were stolen, these secrets would remain safe. Typically your biometric is passed into the secure zone, which attempts to match it with the template stored there, and if all is OK it allows the authentication secret to be retrieved and used to authenticate you.

However this is not truly two factor authentication, it merely simulates two factor authentication! An attacker who can break into the secure zone can grab the authentication secret without needing to have your fingerprint. Also what exactly is this secure zone? In reality there is little agreement on how this could be implemented. And how secure is the secure zone? Is it trap-doored? Can you trust it? How to simulate 3-factor authentication? And what if that file of user public keys on the server is stolen?

5 The M-Pin approach

In true multi-factor authentication you should be able to lose up to $n-1$ of your n factors, and still remain perfectly safe. Accepting that multi-factor authentication is a good idea, now lets now try and do it properly.

A really simple example of what is needed. Your authentication secret is 12069. You split this into 8234+1218+2617. You store 8234 on your phone, 1218 is your password, and 2617 is derived from your fingerprint. I steal your phone and shoulder-surf your password. I know your authentication secret is 8234+1218+? But I have no idea what ? is. Could be anything. The only way to test a guess is to try and use it to authenticate, but the server does not allow multiple incorrect guesses.

So instead of protecting your authentication secret with a second factor in order to merely simulate two factor authentication, lets simply split it into two parts and get the real thing. Or split it into three parts as above. Or more. So your password is one part of it, your biometric another, and whatever is left over is another. All these factors need to be added together to re-create the authentication secret. Pretty obvious really.

Typically the larger part of your secret (what we call the "token") resides on your phone. But its useless without the missing parts contributed by the password and the biometric. If your phone is stolen, nothing to worry about. And no secure zone required. However if you have a secure zone, you can pop the token in there, as it arguably offers extra protection for one of your factors. But its not necessary.

With M-Pin we take the approach that the authentication secret is issued to you by a trusted authority to whom you have proven your right to claim your identity. But we mitigate the potential power of such an authority by splitting it into two (or more) parts. On the server we maintain a single small in-memory hacker-proof secret, the loss of which is not catastrophic. Frustratingly for the file-stealing hacker, an M-Pin server has no files!

Simpler, cheaper and definitely the way to go! The MIRACL trick was simply to get this obvious approach to work in conjunction with methods of strong cryptography.

References

1. The FIDO alliance. `https://fidoalliance.org/`.
2. The Heartbleed Bug. `http://heartbleed.com/`.