How effective are passwords as a means of authentication, how can they be attacked and how can they be made more resilient to attack?

Lucian James

March 13, 2022

Abstract

The aim of this project is to determine the strengths and weaknesses of using passwords as a means of authentication, primarily for online accounts. Protocols for password authentication will be introduced, at various levels of complexity and security. Attacks that can take place against these protocols will then be detailed, then the methods which can be used to secure systems against these attacks.

1 Introduction

Passwords have been used to verify identity since ancient times[citation needed]. In the modern world passwords are used primarily for login processes for computer devices and online services. A typical computer user will use passwords for many different purposes ranging from accessing their computer to performing online bank transactions. Due to the high importance that the confidentiality, integrity and availability of our data is maintained, it is of great importance that the procedures we use to verify identity in order to allow access to our data are highly secure. This project will assess the role that passwords play in modern authentication, and the issues surrounding the use of passwords, both the technical and human aspects will be considered. The primary issue with password-based authentication is that it relies on only one authentication factor; 'Something the user knows', this can be problematic as knowledge factors can be obtained by attackers sometimes with ease compared to other factors, such as "Something the user has" or "Something the user is". The fact that password authentication relies only on a knowledge factor does make it very convenient for users, as

there is no requirement for additional hardware or software to authenticate with a system (such as fingerprint scanners or smartcards). The primary cause of concern around the use of knowledge factors is that users may choose an easily guessed piece of knowledge, or fail to maintain the secrecy of the knowledge, which can cause great insecurity towards their accounts and thus their data. This project will focus primarily on authentication to online servers across a network assumed to be unsafe, but some mention of other use cases will (probably) be made.

2 Password protocols

Protocols are a system of rules and/or procedures that define how two entities interact. A password protocol is then to no surprise, a system of rules and/or procedures that define how authentication using a password takes place. The basic goal of almost all password protocols is simple; Allowing one party to prove that it knows some password, usually set in advance. Protocols which achieve this range from the trivial to the incredibly complex [10]. This section is important as developing an understanding of the various levels of security different protocols provide will be important for section 4.

2.1 A basic password authentication protocol (PAP)

In the simplest form of a password authentication protocol, the user/client sends to the host/server their plaintext username and password, then the server verifies the password either by comparing it directly to a stored plaintext password or applying a one-way hash function to the received password and comparing it to a stored hash. Since the users plaintext password is immediately exposed to being intercepted, this method is unacceptable for use on untrusted networks. Such a protocol is described in IETF RFC1334 [7]:

> "The Password Authentication Protocol (PAP) provides a simple method for the peer to establish its identity using a 2-way handshake. This is done only upon initial link establishment.
> After the Link Establishment phase is complete, an Id/Password pair is repeatedly sent by the peer to the authenticator until authentication is acknowledged or the connection is terminated.
> PAP is not a strong authentication method. Passwords are sent

over the circuit 'in the clear', and there is no protection from playback or repeated trial and error attacks. The peer is in control of the frequency and timing of the attempts"

2.2 Challenge handshake authentication protocol (CHAP)

IETF RFC1994 [8] details a protocol for authentication which provides protection against playback attacks, as the password is not communicated across the connection between the client and the server. The mechanism of the challenge handshake authentication protocol is described as so:

1. User sends their identity to the server.
2. The server uses the identity received from the user to fetch the required information, such as its copy of the users password (p_{server}).
3. The server sends the user a random message, known as a challenge (c).
4. The user uses some hashing function (h) to generate a response (r) to the challenge, using their password (p_{user}) and the random message received from the server. $r = h(p_{user}, c)$
5. The user then sends r to the server.
6. The server makes a comparison, if $r = h(p_{server}, c)$ then the user is authenticated, because $h(p_{user}, c) = h(p_{server}, c) \implies p_{user} = p_{server}$.
7. At random intervals after successful authentication has taken place, the server sends new challenges to the user, repeating the above steps.

Since $h(p_{user}, c)$ is sent across the network in this verification process instead of the plaintext password and c is unique for every authentication, interception is a less viable attack. Although if r and c are captured by an attacker and h is a known function, the attacker can attempt to find the value of p_{user} by calculating $h(x, c)$ and comparing it with r, where x is an arbitrary guess at what the password could be.

$$h(x, c) = r \implies x = p_{user}$$

The process can be repeated as many times as required with different values of x to find the value of p_{user}. As passwords are often considered low-entropy secrets [citation needed], the ability for an attacker to perform offline attacks in unacceptable. Another issue with CHAP is that passwords are stored as

plaintext on the server, irreversible encryption (hashing) cannot be used. If an attacker captures the password files they can use them to authenticate with the server with ease.

2.3 General encryption protocols

One option to ensure security of authentication is to encrypt all communications using some form of asymmetric encryption. An example of a protocol which allows for this kind of encryption is the Transport Layer Security (TLS) protocol, which is utilised by HTTPS. TLS is most likely the most commonly used method of securing communications, including of course communications during authentication. When all communications are encrypted very simplistic protocols such as PAP can be used, as interception is prevented by the encryption of the communications.

The almost universal method of authentication utilises TLS for security, and can be outlined as so:

1. A client-to-server TLS channel is established.
2. The client sends their identity and password p_{user} through the TLS channel.
3. The server runs the password received from the client through a hash function h.
4. The server compares $h(p_{user})$ with its stored hash of the correct password $h(p_{server})$.

$$h(p_{user}) = h(p_{server}) \Longrightarrow p_{user} = p_{server}$$

Using a secure communication protocol greatly reduces the need for a secure authentication protocol, but it does come with its disadvantages, these include:

1. The password appears in plaintext at the server during authentication. This can be a problem as this information could me mistakenly logged and stored on the server. This has occurred on the servers of even the biggest websites in the world. It has been reported that both twitter[1] and facebook[9] have mistakenly stored plaintext passwords in the past.

2. Public-key infrastructure failures can occur which causes the security to be compromised, these can include:

    - Theft of server private keys.
    - Software that does not verify certificates correctly.
    - Users that accept invalid or suspicious certificates.
    - Certificates issued by rogue certificate authorities.
    - Servers that share their TLS keys with others - e.g., CDN providers or security monitoring software.
    - Information (including passwords) that traverses networks in plaintext form after TLS termination
    - And more! :)

[4]

2.4 Password-authenticated key exchange (PAKE)

Password-authenticated key exchange protocols provide a method for two parties to establish a shared key based on their shared knowledge of a secret password, in a way which is immune to offline attacks [4]. PAKE can be used to provide secure authentication without the issues listed in section 2.3, as well as protection against other attacks such as man-in-the-middle attacks.

2.4.1 The secure remote password protocol

A PAKE protocol of note is the "Secure Remote Password Protocol" (SRP protocol), designed in 1998 [10]. The mechanism of the 1998 original version of the SRP protocol can be described as so:
Password establishment:

1. To establish a password P_1 with the server, the client picks a random salt s and computes $x_1 = H(s, P_1)$, where H is some hash function. As well as $v = g^{x_1}$, where g is a primitive root modulo n (often called a generator) where n is a large prime number. The values g and n and well-known and agreed beforehand.

2. The server then stores v and s as the clients password verifier and salt. x_1 is discarded because it is equivalent to the plaintext password P.

Authentication:

1. The client sends the server its username.

2. The server looks up the users password entry and fetches the users password verifier v and the users salt s. The server sends s to the client. The client then computes its long-term private key x_2 using s and the password P_2. $x_2 = H(s, P_2)$ where H is some hash function.

3. The client generates a random number a, $1 < a < n$, and computes its ephemeral public key $A = g^a$, and sends it to the server.

4. The server generates a random number b, $1 < b < n$, and computes its ephemeral public key $B = v + g^b$, and sends it back to the client, along with a randomly generated parameter u.

5. The client computes $S_1 = (B - g^{x_2})^{a+ux_2}$, and the server computes $S_2 = (Av^u)^b$. $P_1 = P_2 \implies S_1 = S_2$, which means that if both the client and the server get the same value of S then the client has the correct password.

Proof $P_1 = P_2 \implies S_1 = S_2$.

$$\begin{aligned} S_2 &= (Av^u)^b \\ &= (g^a((g^{H(s,P_1)})^u))^b \\ &= (g^a(g^{u(H(s,P_1))}))^b \\ &= (g^{a+u(H(s,P_1))})^b \\ &= g^{b(a+u(H(s,P_1)))} \\ &= g^{ba+bu(H(S,P_1))} \end{aligned}$$

$$\begin{aligned} S_1 &= (B - g^{x_2})^{a+ux_2} \\ &= (g^{H(s,P_1)} + g^b - g^{H(s,P_2)})^{a+u(H(s,P_2))} \end{aligned}$$

$$\begin{aligned} P_1 = P_2 \implies g^{H(s,P_1)} - g^{H(s,P_2)} &= 0 \\ \therefore (g^{H(s,P_1)} + g^b - g^{H(s,P_2)})^{a+u(H(s,P_2))} &= (g^b)^{a+u(H(s,P_2))} \\ &= g^{b(a+u(H(S,P_2)))} \\ &= g^{ba+bu(H(S,P_2))} \end{aligned}$$

$$P_1 = P_2 \implies g^{ba+bu(H(S,P_2))} = g^{ba+bu(H(S,P_1))} \therefore S_1 = S_2$$

□

6. Using some hash function H, the client computes $K_1 = H(S_1)$ and the server computes $K_2 = H(S_2)$. K is a cryptographically strong session key.
7. The client uses some hash function H to calculate $M_1 = H(A, B, K_1)$.
8. The client sends the server M_1 as evidence that it has the correct session key. The server then computes M_1 itself and verifies that it matches what the client sent.
9. The server calculates $M_2 = H(A, M_1, K_2)$.
10. The server sends the client M_2 as evidence that it also has the correct session key, The client also verifies M_2 itself, accepting only if it matches the value the server provided.

The SRP protocol has the following advantages:

- If the hosts password file is captured and the intruder learns the value of v, it should still not allow the intruder to impersonate the user without an expensive dictionary search to find the value of P_1.
- Unlike earlier PAKE protocols, it does not require the passwords to be stored on the server in plaintext, instead, the server stores a "verifier" which is a one-way function of the the password hash. This means that a breach of the database does not immediately allow an attacker to impersonate users, they must first perform expensive dictionary attacks to obtain the raw passwords. The technical name for this is asymmetric password-authenticated key exchange.
- Public-key infrastructure is not required.
- Despite drawbacks that it may have, the SRP protocol is simple, there is working code in OpenSSL that even integrates with TLS, which makes it relatively easy to adopt.

The SRP protocol has the following disadvantages:

- Earlier versions of the SRP protocol have been broken several times, which is why the protocol is currently on revision 6a. Additionally the "security proof" in the original paper doesnt really prove anything meaningful.
- The SRP protocol leaks salt to unknown users by design, making it vulnerable to pre-computation attacks.

[3]

2.4.2 OPAQUE

A more recent PAKE protocol of note is "OPAQUE", which is an asymmetric PAKE protocol secure against pre-computation attacks [4]. A description of the full mechanism of this protocol is out of the scope of this project (The paper is 61 pages long!), but the benefits which it offers over a protocol such as the SRP protocol can be described easily thanks to [3]:

- OPAQUE does not reveal salts to potential attackers. This is done by combining the salt with the password, in a way which ensures the client does not learn the salt and the server does not learn the password.
- OPAQUE works with any password hashing function.
- All the hashing work is done on the client, which means OPAQUE can actually take load off the server.
- Unlike SRP, OPAQUE has a reasonable security proof.

[3]

Details about salt secrecy

"The main problem with earlier PAKEs is the need to transmit the salt from a server to a (so far unauthenticated) client. This enables an attacker to run pre-computation attacks, where they can build an offline dictionary based on this salt."

"The challenge here is that the salt is typically fed into a hash function (like scrypt) along with the password. Intuitively someone has to compute that function. If it's the server, then the server needs to see the password — which defeats the whole purpose. If it's the client, then the client needs the salt."

"OPAQUE gets around this with the following clever trick. They leave the password hash on the client's side, but they don't feed it the stored salt. Instead, they use a special two-party protocol called an oblivious PRF[1] to calculate a second salt (call it salt2) so that the client can use salt2 in the hash function — but does not learn the original salt. The basic idea of such a function is that the server and client can jointly compute a function PRF(salt, password), where the server knows 'salt' and the client knows

[1] Pseudo Random Function

'password'. Only the client learns the output of this function. Neither party learns anything about the other party's input."[3]

The implementation of oblivious PRF relies on the idea that the client has the password P and the server has the salt s. The output of the PRF function should be in the form $H(P)^s$ where H is a special hash function that hashes passwords into elements of a cyclic (prime-order) group.

$$H(P) \in \mathbb{Z} \bmod \mathbb{P}$$

To compute this, PRF requires a protocol between the client and server. In this protocol, the client first computes $H(P)$ and then 'blinds' this password by selecting a random scalar value r, and blinding the result to obtain $C = H(P)^r$. At this point, the client can send the blinded value C over the server, secure in the understanding that (in a prime-order group), the blinding by r hides all partial information about P.

The server, which has salt value s, now further exponentiates this value to obtain $R = C^s$ and sends R back to the client. Written out in detail, the result can be expressed as $R = H(P)^{rs}$. The client now computes the inverse of its own blinding value r and exponentiates one more time as follows: $R' = R^{r^{-1}} = H(P)^s$. This element R', which consists of the hash of the password exponentiated by the salt, is the output of the desired PRF function.

A nice feature of this protocol is that if the client enters the wrong password into the protocol, it should obtain a value which is very different from the actual value it wants. This guarantee comes from the fact the hash function is very likely to produce wildly different outputs for distinct passwords.

Theoretically, the salt can be found. The client knows the value of R' which is equal to $H(P)^s$, and the value of $H(P)$, so theoretically it is possible to find the value of s by computing $log_{H(S)}R'$. But this is only a theoretical possibility, as under the condition that the value produced by $H(P)$ is $2q+1$ where q is a large prime number, this calculation falls into the discrete logarithm problem, which is well known to be impractical to compute [2].

3 Password dataset and cracking analysis

Might make sense for this to be its own section, despite the fact some individual parts fit may better into the vulnerabilities section.

3.1 Datasets used

- **Top2Billion-probable-v2**
  20.6gb file containing 1973218846 "passwords", they are ordered in the file starting with the most common to the least common. All items in this dataset are sourced from other datasets.
  Sourced from the torrent listed on github.com/berzerk0/Probable-Wordlists.

- **hk-hlm-founds.txt**
  389.4mb file containing 38647798 "passwords", quick examination of the file shows some reasonably likely passwords and some gibberish.
  Sourced from weakpass.

- **cyclone.hashesorg.hashkiller.combined**
  15.0gb file containing 1469156570 "passwords", quick examination of the file shows some reasonably likely passwords and some gibberish.
  Sourced from weakpass.

- **weakpass-3p**
  14.5gb file containing 1454086314 "passwords", the file appears to be sorted alphabetically. fair amount of gibberish.
  Sourced from weakpass.

- **ASLM**
  479.8mb file containing 41591035 "passwords".
  Sourced from weakpass.

- **pwned-passwords-ntlm-ordered-by-hash-v7**
  20.6gb file containing 613584246 NTLM hashes of "passwords", along with the frequency that they are found in "data breaches".
  Sourced from haveibeenpwned.

- **Rules1.txt**
  A set of rules for use during rule-based password cracking. It consists of 229 rules which modify input password guesses. Sourced from weakpass with minor modification.

3.2 Software used

- C++ (g++ compiler)
- Python
- Hashcat

3.3 Hardware used

- CPU: AMD Ryzen 5 2600
- RAM: 16GB
- GPU: Nvidia Quadro P5000

3.4 Dataset analysis

[6] contains information about 4057 passwords (1522 unique) from 154 participants across 2077 web domains. This paper provides a reliable real-world analysis to compare the password dictionaries I have collected to. The characteristics of the data collected for this paper are as follows:

> "On average, participants submitted a password 1.40 times per day. Including all passwords, participants had an average password length of 9.92 characters with their average password composed from 2.77 character classes including 2.70 digits, 5.91 lowercase letters, 0.84 uppercase letters, and 0.46 special characters."

Making a comparison between this information, and a similar analysis of the password dictionaries listed in 3.1 provides some indication of how realistic these dictionaries are.

3.4.1 Top2Billion-probable

- Average length ≈ 10.01
- Average number of uppercase characters ≈ 0.969
- Average number of lowercase characters ≈ 5.8
- Average number of digits ≈ 3.14
- Average number of special characters ≈ 0.174

3.4.2 hk-hlm-founds

- Average length ≈ 9.56
- Average number of uppercase characters ≈ 0.715
- Average number of lowercase characters ≈ 5.49
- Average number of digits ≈ 3.26
- Average number of special characters ≈ 0.386

3.4.3 cyclone.hashesorg.hashkiller.combined

- Average length ≈ 9.84
- Average number of uppercase characters ≈ 0.677
- Average number of lowercase characters ≈ 5.58
- Average number of digits ≈ 3.41
- Average number of special characters ≈ 0.597

3.4.4 weakpass-3p

- Average length ≈ 9.69
- Average number of uppercase characters ≈ 0.769
- Average number of lowercase characters ≈ 5.57
- Average number of digits ≈ 3.35
- Average number of special characters = 0.00

Interestingly, this dictionary contains no special characters at all, whoever made it must have chosen to exclude any entries that contain special characters for some reason.

3.4.5 ASLM

- Average length ≈ 11.1
- Average number of uppercase characters ≈ 0.230
- Average number of lowercase characters ≈ 5.47
- Average number of digits ≈ 4.34
- Average number of special characters ≈ 5.40

3.5 Password cracking analysis

Hashcat is a GPU-accelerated tool for cracking password hashes, it supports a wide range of hash types and many different attack modes. This analysis will focus on dictionary attacks, with and without modification rules. A dictionary attack is a method of discovering the correct password by systematically trying all entries in a "dictionary". Modification rules can be

used to augment a password dictionary by doing simple operations such as appending numbers, capitalising the first letter, etc.

Time taken is not considered in this analysis, as it varies greatly with the hardware being used, and the hashing algorithm used. Instead, this analysis focuses on the number of passwords cracked from a set of hashes.

For this analysis, the "rules" file consisted of 229 rules which perform simple modifications to every item in the input password dictionary, these modifications include, among others:

- Capitalising the first letter of passwords
- Appending numbers which represent "years" to the end of passwords
- Appending recent years to the end of passwords
- Appending special chars

Below is how many hashes each dictionary managed to crack out of the first 1 million hashes in pwned-passwords-ntlm-ordered-by-hash-v7.

3.5.1 Top2Billion-probable

Without rules
245137/1000000 (24.51%)
With rules
369683/1000000 (36.97%)

3.5.2 hk-hlm-founds

Without rules
28399/1000000 (2.84%)
With rules
99867/1000000 (9.99%)

3.5.3 cyclone.hashesorg.hashkiller.combined

Without rules
934087/1000000 (93.41%)
With rules
934344/1000000 (93.43%)

3.5.4 weakpass-3p

Without rules
55876/1000000 (85.59%)
With rules
863702/1000000 (86.37%)

3.5.5 ASLM

Without rules
52045/1000000 (5.20%)
With rules
150402/1000000 (15.04%)

3.5.6 Password cracking analysis conclusion

The cyclone.hashesorg.hashkiller.combined dataset achieved an incredibly high recovery rate, this indicates that pwned-passwords-ntlm-ordered-by-hash-v7 was probably made using this dataset, as 93% recovery is not very realistic. This demonstrates that my analysis of password cracking is in some sense flawed due to the datasets I am using. Due to this apparent flaw in my methods, I cannot reliably come to any conclusions about the rate at which real-world passwords can be cracked using dictionary attacks. If the time constraints of this project were not so strict, I could attempt to source a more reliable dataset of password hashes and retry this experiment.

4 Vulnerabilities

A variety of vulnerabilities can be exploited to gain unauthorised access to a system which utilises passwords for authentication, these attacks can exploit technical flaws, human behaviour, or both.

4.1 Human(user) vulnerabilities

4.1.1 Weak passwords

One of the major problems with password-based authentication systems is that users are often quite lazy with their choice of passwords. Choosing easily guessed passwords and re-using passwords are incredibly common. This is often due to the large amount of misconceptions that users may have about password security. something something [5].

References

[1] Abrar Al-Heeti. Twitter advises all users to change passwords after glitch exposed some in plain text, May 2018.

[2] Changyu Dong. Math in network security: A crash course, 2016.

[3] Matthew Green. Let's talk about pake, Oct 2018.

[4] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque: an asymmetric pake protocol secure against pre-computation attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 456–486. Springer, 2018.

[5] Peter Mayer and Melanie Volkamer. Addressing misconceptions about password security effectively. In *Proceedings of the 7th Workshop on Socio-Technical Aspects in Security and Trust*, pages 16–27, 2018.

[6] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 295–310, 2017.

[7] William Allen Simpson. Ppp authentication protocols, Oct 1992.

[8] William Allen Simpson. Ppp challenge handshake authentication protocol (chap), Aug 1996.

[9] Zack Whittaker. Facebook admits it stored 'hundreds of millions' of account passwords in plaintext, Mar 2019.

[10] Thomas D Wu et al. The secure remote password protocol. In *NDSS*, volume 98, pages 97–111. Citeseer, 1998.