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

April 17, 2022

Abstract

This project aims to discover the major vulnerabilities with password-based authentication, along with solutions to these vulnerabilities. Some protocols that can be used to secure password-based authentication are detailed in this report to show how secure password authentication is in terms of technical cryptographic security. As well as citing information about vulnerabilities, an analysis of offline password cracking is made along with an analysis of the password dictionaries used for this task.

1 Introduction

Passwords are used very often 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 how the security of password-based authentication is ensured, considering both the technical and human aspects.

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 on authentication to online servers across a network assumed to be unsafe, but the sections discussing choice of password are still especially relevant to the use of passwords for offline access (for example, file encryption), as brute-force attacks can be performed much easier in these scenarios.

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 [12]. This section is important as developing an understanding of the various levels of security different protocols provide will be important for section 3.

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 [9]:

> "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 [10] 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 $h(p_{user}, c) = h(p_{server}, c) \Longrightarrow p_{user} = p_{server}$.
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.

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

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 $h(p_{server})$.

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

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[11] 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! :)

[5]

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 [5]. 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 [12]. 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 $P_1 = P_2 \implies S_1 = S_2$.

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

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

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

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

□

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.

[4]

2.4.2 OPAQUE

A more recent PAKE protocol of note is "OPAQUE", which is an asymmetric PAKE protocol secure against pre-computation attacks [5]. 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 [4]:

- 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.

[4]

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

[1] Pseudo Random Function

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 'password'. Only the client learns the output of this function. Neither party learns anything about the other party's input."[4]

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.

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 [3].

3 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.

3.1 Human(user) vulnerabilities

3.1.1 Weak passwords and password behaviour

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.

"Addressing Misconceptions About Password Security Effectively" [7] conducted a systematic literature review to determine the misconceptions the general public may have about password security, as well as how to address them. This work identified 23 misconceptions about password security, which where broken down into four categories, composition, handling, attacks and miscellaneous. The first two of these categories are most relevant to this section.

- **Composition:**
  Users misunderstanding what a secure password is composed of is very common. There are many ways users may misunderstand how to compose a secure password, but the five outlined by [7] are:
  - Adding numbers makes passwords automatically more difficult to guess.
  - Adding symbols makes passwords automatically more difficult to guess.
  - Adding uppercase letters makes passwords automatically more difficult to guess.
  - Replacing lowercase letters in the password with numbers, symbols, or uppercase letters makes the password more difficult to guess.
  - A word from another language than the your own mother tongue is a secure password.

- **Handling:**
  Users misunderstanding what behaviour puts the security of their accounts at risk is also common. There are many ways users may misunderstand how to securely protect their passwords, but the six outlined by [7] are:
  - Reusing passwords is ok for secure passwords, but should be avoided for weak passwords.
  - Reusing is ok for more frequently used passwords.
  - Reusing passwords is better than writing them down.
  - Notes of passwords do not need to be particularly protected.
  - Passwords have to be changed frequently.
  - Storing passwords in the browser does not mean one is using a password manager.

3.1.2 Phishing

Phishing attacks are a form of social engineering which attempt to trick victims into handing sensitive information over to an attacker. Attackers generally impersonate some other trusted entity when sending attacks to victims, quite often in the form of a link inside an email.

Phishing is a major attack technique used to capture victim passwords, this is done by creating a fake log-in page for some service and then making efforts to redirect users to the fake log-in page instead of the legitimate log-in page for that service. This is often done by sending the target an email prompting them to log in to their account for some reason or another, providing them with a malicious link which appears to be the legitimate page.

Phishing can be incredibly effective at obtaining the passwords of unsuspecting targets, these passwords can then be used by the attacker to falsely authenticate themselves and perform actions inside the victims account. [2]

3.2 Technical vulnerabilities

3.2.1 SQL injection

SQL injection is a type of attack which targets web database files, it takes advantage of websites which send user input into an SQL query without

proper validation and cleaning. SQL injection can be used to expose hashed passwords stored on a database, among many other things. The attacker can attempt to decrypt these hashed passwords to gain access to user accounts on the service.

3.2.2 MITM

Man-**I**n-**T**he-**M**iddle attacks exploit systems by capturing and/or modifying communications between two target parties. A man-in-the-middle attack is performed how it sounds, a malicious party sits in-between two targets, intercepting their communications.

On very insecure systems attackers could simply capture plaintext or hashed passwords being communicated across the network, but generally more complex techniques are required to capture sensitive information such as passwords. A man in the middle who has the ability to modify traffic could in theory redirect targets to a phishing webpage when they try and visit certain legitimate sites. Therefore, man in the middle attacks are classified in two ways:

1. Passive attacks, when an attacker only listens.
2. Active attacks, when an attacker modifies packets.

4 Password dataset and cracking analysis

One of the major issues with passwords is the fact that users often put little effort into choosing and memorising secure passwords. Section 3.1.1 mentions this, but this whole section is dedicated to a more thorough analysis of how the prevalence of weak passwords can be exploited through the use of password lists and specialised hash cracking software. This section can in some ways, be considered an extension of section 3.1.1.

4.1 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.

4.2 Software used

- C++ (g++ compiler)
- Python (Not actually used for anything that made it into this report though!)
- Hashcat

4.3 Hardware used

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

4.4 Dataset analysis

[8] 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 4.1 provides some indication of how realistic these dictionaries are.

Dataset	Avg length	Avg n uppercase	Avg n lowercase	Avg n digits	Avg n special
Top2Billion-probable	10.01	0.969	5.8	3.14	0.174
hk-hlm-founds	9.56	0.715	5.49	3.26	0.386
CHHC	9.84	0.677	5.58	3.41	0.597
weakpass-3p	9.69	0.769	5.57	3.35	0.00
ASLM	11.1	0.23	5.47	4.34	5.40

The below table shows the result of $\sum |x_i - y_i|$ performed using the data about the password datasets, and the data about the real-world passwords. This gives a number quantifying how close overall each dataset is to real-world passwords in terms of character frequency.

Dataset	Sum of distances
Top2Billion-probable	1.055
hk-hlm-founds	1.539
CHHC	1.42
weakpass-3p	1.751
ASLM	8.81

The Top2Billion-probable dataset is overall has the closest characteristics to real passwords and the ASLM dataset is by far the most unrealistic password dataset, mostly due to the fact it contains far too many special characters.

4.5 Password cracking analysis

4.5.1 Introduction

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".

Being able to decrypt hashed passwords is valuable to an attacker who has successfully captured the user database of their target, as it allows them to log into user accounts on their target.

The "correct" password is the password which when sent through some hashing algorithm, produces an identical hash to the one which we are trying to decrypt.

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.

Modification rules can be used to augment a password dictionary by doing simple operations such as appending numbers, capitalising the first letter, etc. 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 (people born in 1998 may append "98" onto their passwords for example)
- Appending recent years to the end of passwords
- Appending special chars

For my experiment, I produced a simple shell script to run over all my datasets. This script is pictured below. (fig 1)

```
#!/bin/bash

# Some helpful paths
path=$HOME/Downloads/passwordStuff
hashes=$path/pwned-passwords-ntlm_first1M_cleaned
rules=$path/rules1.txt

# Top2Billion probable no rules
hashcat --potfile-disable -m 1000 $hashes $path/Top2Billion-probable-v2.txt -o top2billion_norules | tee top2billion_norules_term
# Top2Billion probable with rules
hashcat --potfile-disable -m 1000 $hashes $path/Top2Billion-probable-v2.txt -r $rules -o top2billion_withrules | tee top2billion_withrules_term

# hk_hlm_founds no rules
hashcat --potfile-disable -m 1000 $hashes $path/hk_hlm_founds.txt -o hk-hlm_norules | tee hk-hlm_norules_term
# hk_hlm_founds with rules
hashcat --potfile-disable -m 1000 $hashes $path/hk_hlm_founds.txt -r $rules -o hk-hlm_withrules | tee hk-hlm_withrules_term

# weakpass_3p no rules
hashcat --potfile-disable -m 1000 $hashes $path/weakpass_3p -o wp_3p_norules | tee wp_3p_norules_term
# weakpass_3p with rules
hashcat --potfile-disable -m 1000 $hashes $path/weakpass_3p -r $rules -o wp_3p_withrules | tee wp_3p_withrules_term

# ASLM no rules
hashcat --potfile-disable -m 1000 $hashes $path/ASLM.txt -o ASLM_norules | tee ASLM_norules_term
# ASLM with rules
hashcat --potfile-disable -m 1000 $hashes $path/ASLM.txt -r $rules -o ASLM_withrules | tee ASLM_withrules_term

# cyclone.hashesorg.hashkiller.combined.txt no rules
hashcat --potfile-disable -m 1000 $hashes $path/cyclone.hashesorg.hashkiller.combined.txt -o CHHC_norules | tee CHHC_norules_term
# cyclone.hashesorg.hashkiller.combined.txt with rules
hashcat --potfile-disable -m 1000 $hashes $path/cyclone.hashesorg.hashkiller.combined.txt -r $rules -o CHHC_withrules | tee CHHC_withrules_term

```

Figure 1: Shell script for hashcat

4.5.2 Results table

Dataset	Raw	Percent
Top2Billion-probable without rules	245137/1000000	24.51%
Top2Billion-probable with rules	369683/1000000	36.97%
hk-hlm-founds without rules	28399/1000000	2.84%
hk-hlm-founds with rules	99867/1000000	9.99%
cyclone.hashesorg.hashkiller.combined without rules	934087/1000000	93.41%
cyclone.hashesorg.hashkiller.combined with rules	934344/1000000	93.43%
weakpass-3p without rules	55876/1000000	85.59%
weakpass-3p with rules	863702/1000000	86.37%
ASLM without rules	52045/1000000	5.20%
ASLM with rules	150402/1000000	15.04%

4.5.3 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. Pictured below (fig 2) are some of the "passwords" that the CHHC dataset managed to recover (hashes left of the **:**, "passwords" on the right), further solidifying the idea that the recovery rate is high not because of dataset realism, but because they overlap.

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.

Figure 2: Junk data seen in output of hashcat

5 Mitigations

5.1 Addressing password security misconceptions

Mitigating the risk of bad password security practices from users is mostly a matter of education. [7] provides explanations of the truth behind each of the identified misconceptions, they referred to these texts as "intervention texts". The authors then performed a user study, this evaluation of the intervention texts consisted of:

1. A pre-treatment questionnaire measuring the prevalence of the different misconceptions in their participant sample.

2. the treatment using the intervention texts alongside information describing attacks on passwords and user accounts as well as respective defences.

3. a post-treatment questionnaire measuring the prevalence of the different misconceptions in their participant sample after having been exposed to the interventions as well as collecting basic demographics data.

They concluded that their interventions could be used to effectively decrease the prevalence of many of the identified misconceptions.

5.2 Phishing

[2] describes a variety of ways in which phishing can be detected:

5.2.1 Machine learning

Machine learning and data mining techniques can be utilised for detecting phishing, specifically:

Classification:
When using classification techniques, a model will be created that takes some input email/text and produces an output classifying it as either legitimate or phishing mail.

Clustering:
Clustering is used very similarly to classification, it partitions a set of instances into two clusters, phishing and legitimate. The objective of clustering is to group objects based on their similarities.

Anomaly detection:
An anomaly is a pattern in data which is not consistent with expected normal behaviour. Anomaly detection can be applied to the detection of phishing, treating phishing as an outlier. Phishing websites and mail typically demonstrate abnormal behaviour compared to their legitimate counterparts, this abnormal behaviour can be detected.

Text mining:
Text mining refers to utilising data mining and machine learning techniques to discover trends, patterns, and any other useful knowledge from a piece of text. Text mining can be used to detect phishing attempts by analysing patterns in the contents of emails, instant messages, URLs, websites, and more.

5.2.2 Human detection

Increasing user awareness:
Since phishing primarily exploits a lack of security awareness in targets, increasing this awareness should help reduce the amount of successful phishing attacks taking place significantly. A variety of factors affect a users security awareness, including:

- Security knowledge
- Web experience
- Computer self-efficacy
- Disposition to trust
- Perceived risk
- Suspicion of humanity

The primary technique used to increase the likelihood of human users successfully detecting phishing attempts is training and education. One form of this is sending users security notices embedded in emails with graphics and text explaining what phishing attacks are and how to detect them, along with any other relevant important information.

Involving users in identifying phishing material:
Users are expected to be able to manually identify new phishing attempts through participation in identifying legitimate and phishing material. There are two approaches:

1. Manual authentication. This approach notifies users to identify suspicious behaviour linked to phishing themselves, reminding them that they must also make an effort to prevent phishing attacks from successfully taking place.

2. User voting. This is an interesting approach where a community database is created and users can submit suspected phishes, other users can then vote for whether these submissions are phishing or legitimate. This can also be used to create labelled datasets for the training of machine-learned detection systems.

5.2.3 Other phishing countermeasures

Honeypots:
Honeypots are traps created to capture information about attackers. These traps do this by convincing attackers that they are real victims, then observing exactly how the attacker decides to proceed with their phishing attempts. The data collected from observing the attacker can be used when building attacker blacklist databases, and/or to discover and block suspicious domains.

Search engines:
Search engines can be combined with other techniques to detect phishing. Typically legitimate pages are indexed and assigned a rank by search engines, this rank will generally be reasonably high, especially for major websites. Phishing domains will generally rank very low on search engines, or not be indexed at all, this means search engines can be used to detect suspicious links.

5.3 Injection/database compromise

5.3.1 Input validation

Simple input validation can be used to prevent injection attacks from being successfully used. This validation would involve removing characters or strings of characters which are used

5.3.2 Honeypot accounts

The concept of honeypot accounts is simple, fake account entries are added to the user database and given relatively weak passwords and a system is set in place to alert administrators when those accounts are logged into. This means that malicious use of user accounts will hopefully be detected quickly.

Honeypot accounts are not guaranteed to alert administrators though, an attacker may target specific real accounts or be able to identify the honeypot accounts by their usernames.

5.3.3 Honeywords

[6] describes the concept of "honeywords". this concept extends on honeypot accounts, aiming to address the issues identified with honeypot accounts. This is done by extending the basic concept to every single account, by having multiple passwords stored for each and every account, of which only one is the real and usable password. The fake passwords are known as the "honeywords". The attempted use of a honeyword to authenticate will alert administrators, as it should only happen if the database has been compromised and decrypted.

The fact that honeywords make every single account inaccessible without a high chance of alerting administrators is a huge advantage over the similar method of creating honeypot accounts.

5.4 MITM

The prevention of man-in-the-middle attacks is a matter of implementing complex cryptography into communication and authentication protocols. Preventing passive attacks which aim to capture passwords was discussed in great detail in section 2. Preventing active attacks requires much more additional cryptographic nonsense used to validate the identity of each party during communications, this additional cryptography is unfortunately not within the scope of this project.

6 Conclusion

The security of password-based systems can be ensured on a technical level well as discussed in section 2, although active attacks were not discussed and the solutions to active attacks were deemed out of the scope of this project in section 5.4.

The major issue with password-based systems is most certainly the fact it is well open to social-engineering based attacks such as phishing. Section 5.2 details some solutions to this, but educating all users on the threat of phishing can be a difficult and time consuming task.

The primary flaw with password-based authentication is almost solely

the end user, they often do not behave in a way which can be considered secure. Issues with user behaviour ranges from the inability to create and remember secure passwords, to great susceptibility to social engineering attacks.

While the analysis of password cracking (section 4.5) in the scenario of a breached database had some flaws, the most realistic dictionary (according to character set count) achieved a 36.97% recovery rate when using modification rules. Although the realism of the set of 1 million hashed passwords used for the password cracking analysis cannot be confirmed, it is still alarming that around 30-40% of passwords may in theory be possible to crack with relative ease (not even taking into consideration the fact that many passwords will be used many times by many different users, the statistic of 36.97% was with a set of completely unique hashes). This is further indication that user knowledge is of great concern to the security of password-based authentication.

When paired with well educated users, password authentication can be considered very secure. The effectiveness of password-based authentication systems depends greatly on those using the systems, great responsibility is placed on the users to keep their own accounts safe.

References

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

[2] Ahmed Aleroud and Lina Zhou. Phishing environments, techniques, and countermeasures: A survey. *Computers & Security*, 68:160–196, 2017.

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

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

[5] 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.

[6] Ari Juels and Ronald L Rivest. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 145–160, 2013.

[7] 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.

[8] 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.

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

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

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

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