Password (In)security
Attacks & Prevention

Jon Meyer and Jared Zook

July 18, 2016
Version 2.1

University of Idaho

CS 539: Applied Security Concepts

Summary

A good password security policy requires, at a minimum, the use of complex, unique passwords for different services. People often view this practice as cumbersome or difficult and continue to use one easy password for all their needs. This tutorial demonstrates how trivial it is for an attacker to crack simple passwords. This should help the reader develop an intuitive understanding of the importance of choosing strong passwords. The tutorial includes walk-throughs of several common methods that attackers use to crack passwords, including dictionary attacks and word-mangling attacks. The reader's understanding will be further cemented by a series of challenges relating to the attacks. A discussion of safe password practices is also included.

Prerequisites

Basic familiarity with the bash command line.

Contents

1 Problem Statement 1

2 Related News 2

3 Proposed Solution 3

4 Password Cracking Attacks 4

5 Password Security Guidelines 6

6 Tutorial I: John the Ripper (john) 7

7 Linux Passwords 8

8 Unshadowing the Password File 9

9 Cracking MD5 Hashes by Default 10

10 Cracking MD5 Hashes with a Wordlist 11

11 Writing Our Own Cracking Rules with john 12

12 Cracking MD5 Hashes by Specifying Rules 13

13 Challenge 1 14

14 Challenge 2 15

15 Questions 16

16 Alternative Solution: Rainbow Tables 17

17 Conclusion 18

18 Appendix: Setting Up the VM, Solutions, and Change-log 19

1 Problem Statement

In today's digital systems, how do we make sure users can access their data and authorized system functions while protecting against unauthorized access by other users?

Nearly all sufficiently advanced applications require identity authentication of some sort, either to keep information private or to prevent unauthorized people from gaining control of a system or application. Passwords prevent unauthorized logins while not inconveniencing legitimate users too much. Because password cracking is such a widely visible attack vector for hackers, it is very important that passwords are sufficiently complex as to not be cracked in a reasonable amount of time.

2 Related News <>

1. Untrained reporter cracks thousands of hashed passwords in just a few hours (2013) [3]
2. 11 million passwords cracked in 10 days following Ashley Madison leak (2015) [4]
3. Flaw in Windows 8 allowed passwords to be stored in plaintext (2012) [5]

1. In 2013, Ars Technica studied password cracking by providing a file with 16,000 password hashes from a web service. Then, it invited an unseasoned reporter and three password cracking experts to decipher the hashes. Over the course of a few hours, the reporter was able to crack 47% of the passwords by using a tool. The top expert was able to crack 90% of the passwords in less than 24 hours. Simple passwords (e.g. *password1*) were broken almost instantaneously (along with variations such as *p@ssw0rd*). More complex passwords (e.g. *@Oceancity12* and *momof4g8kids*) were compromised as well by advanced techniques. The biggest contributing factors to the insecurity were, on the user side, not using complex, random passwords and, on the client side, using MD5 to hash the passwords. MD5 was not designed for password safety, it is fast, but weak, and should never be used for this purpose.

2. In September, hackers leaked 100GB of sensitive data from the Ashley Madison "dating" website. Among the data were files containing hashed user passwords for all 38 million users. The passwords were encrypted with bcrypt, a very slow hashing algorithm. The bcrypt hashes would have taken hypothetically centuries to break. However, crackers from "CynoSure Prime" discovered a database of hashed passwords that also included login key tokens hashed with MD5. They were able to crack the tokens, which made it a trivial feat to crack the passwords. In only 10 days, they were able to crack over 11 million passwords.

3. In the early days of Windows 8, researchers found a flaw in the method Microsoft used to store passwords. Windows 8 allowed for alternative authentication methods. To enable these, one first had to create a login with a passphrase. That passphrase was copied, and stored in an AES-encrypted "vault". This vault, however, was accessible to anyone designated an administrator on the system. By simply entering the vault, admins could view the hex representation of the password which is easy to convert to plaintext. This example shows that reversible encryption should never be used to store passwords.

3 Proposed Solution <>

1. Traditionally, the use of user IDs and passwords help us carry out these functions
2. This practice is not without its weaknesses and challenges
3. Let's look at some of the challenges presented with the use of passwords and find out ways to maximize their benefits...

1. The use of user IDs and passwords remains the most common form of identity authentication in digital applications. While other forms of authentication have arisen, such as biometric authentication and text messaging authentication, the use of passwords is dominant.

2. User IDs and passwords are not without their shortcomings. Passwords can be guessed, stolen, or cracked. Common passwords are very simple and often have as a component an English word or series of numbers that holds personal significance to the user. We rely on passwords to provide assurance of authentication into the services we use on a daily basis. Though people are admonished to use "strong passwords", they choose not to because they find it difficult. Unfortunately, this leaves many open to password cracking attacks that compromise the security of user accounts. [7]

3. Passwords are stored on a computer in the form of a hash. If a user has access to these hashes, he can attempt to crack them by guessing various passwords, applying the hash function to them, and checking if the resultant hash matches the hash of the password of the user he wants to impersonate. In order for passwords to be hard to crack, they must be complex. To make a password as complex as possible, one must make it sufficiently long and select characters randomly from a large character set.

4 Password Cracking Attacks <>

Passwords can be cracked using:

1. Brute-force attacks
2. Dictionary attacks
3. Precomputed tables
4. "Intelligent" brute-force attacks (e.g. Markov chains)

Passwords generally are (and should be) stored as a hashed message digest instead of plaintext [3]. As such, the attempts for each of these attacks will generally need to be hashed using the same algorithm to match the corresponding stored password hash. This is because cracking attacks are often run against password files that the attacker apprehended [6].

1. Brute-force attacks aim to crack passwords by trying every possible permutation of a password. For example, a brute-force attack against a four character password would begin with "aaaa" and attempt every possible iteration up until "zzzz". This attack is simple and effective, so long as there is enough time to run through the permutations. As such, the drawback of brute-force methods is that, with each additional character added to the password, the time it takes to crack it grows expontentially. [3]

2. Dictionary attacks attempt to match words from a given wordlist against a set of passwords [3]. This method is more sophisticated than a brute-force search because, instead of generating the passwords character-by-character, a list of likely character combinations can be used (e.g. "password").

3. Rainbow tables represent one type of pre-computed dictionary attack. Tables store large amounts of passwords and their corresponding hashes. A rainbow table attack uses a hash value taken from the list and reduces it using a *reduction function* to match against passwords in the table. If a reduction matches a hash, the password may be found by generating a chain of hashes to find a match with one of the pre-computed hashes. Since rainbow tables incorporate the same hashing algorithm as the password storage, they can be defeated if the administrator includes a "salt" or a random value added to the password before hashing. This makes it so identical passwords can have different hashes, frustrating the attack.

4. An example of an intelligent brute-force attack is an attack using a Markov chain. By this method, probabilities for each position of the password can be determined by using previously-cracked passwords [3]. The rules we write in this tutorial for john the ripper are another example.

5 Password Security Guidelines <>

Administrators should:

- Never store plaintext passwords
- Use a slow hashing algorithm to hash *salted* passwords
- Incorporate complexity requirements for password creation

Users should:

- Use strong, unique passwords
- Consider using a password manager and/or two-factor authentication

Secure password storage requires the use of a one-way hash. The contents of the password file need to be obscured in case they make their way into the wrong hands. Prior to hashing passwords, the administrator should add a *salt*, a randomly-generated string of characters, to the password to slow cracking attempts and to frustrate rainbow table attacks. It is also important to be mindful of which hashing algorithm to put the salted passwords through. Common hashing algorithms such as MD5 were not designed for password hashing, but for low computational cost. For example, in 2013 a single GPU cracking MD5 passwords could guess 8 billion combinations per second. When used against a SHA512crypt hash, it could only achieve 2,000 guesses per second. [3] An example of a good, slow, computationally-expensive hashing algorithm is **bcrypt** [4].

Traditional advice for password complexity requires that the password be at least 12 characters long, consisting of letters (upper and lowercase), numbers, and special characters. In addition, it should not be a dictionary word and shouldn't have obvious substitutions. To be even more secure, users should consider using a *password manager*. Password managers allow you to store strong, randomized, unique passwords for every system you connect to. [7] To avoid having a single point of failure due to the master password, methods such as two-factor authentication can add an extra layer of security [8]. Additionally, many password managers which store encrypted passwords locally can be configured to require the use of a key file in conjunction with a master password to unlock all of a user's saved passwords. This is a common way to prevent a single point of failure when storing passwords on a local machine.

6 Tutorial I: John the Ripper (john) <>

john:

- a fast, free, multi-platform, open-source password cracker
- will allow us to try out brute-force, dictionary, and rule-based attacks

Default modes:

- single
- wordlist with rules
- incremental

John the Ripper (hereafter referred to as "john") is a free, open source password cracking utility. It was initially designed to uncover weak passwords on Unix systems, but is also used in a wider scope for general password cracking. It is available for both *nix and Windows-based systems and can crack password hashes from different hashing algorithms (e.g. crypt(3) for Unix, LM for Windows). [9] john is typically run from the command line, but GUI interfaces exist for it (e.g. "Johnny" on Kali Linux).

In lieu of running any additional command line options, executing **`john <pass.txt>`** will run john in the following order of cracking modes: single crack, wordlist with rules, and incremental. Single crack mode will try passwords based off user names, word-mangling, and previously cracked passwords. Wordlist mode incorporates rules specified in john's configuration file (or by the user) and runs a wordlist against password hashes in the list. A wordlist is just a text file with each line representing a different word that can be a password. This is akin to the dictionary attack discussed previously. If a wordlist is not specified, john will use its default wordlist. Finally, incremental mode tries every different possible combination of characters for the password. It is essentially a brute-force attack. [9]

7 Linux Passwords <>

- Traditionally, UNIX account information and passwords were stored in **/etc/passwd**
- This creates a security concern because **/etc/passwd** is a world-readable file used by many command line tools
- To mitigate this risk, many distros store this information in a shadow password file (**/etc/shadow**) which can only be read by the root account
- With root access, it is possible to decrypt all system passwords

Traditional Unix systems store user names, one-way encrypted passwords, and user attributes (e.g. User ID, home directory, login shell) in **/etc/passwd**. Many Unix utilities use the information in the file to execute properly, therefore **/etc/passwd** needs to be *world-readable*. Since it is world-readable, this leaves the password hashes in a vulnerable position. To address this vulnerability, many Linux distributions store the password hash portion of **/etc/passwd** in a *shadow password* file, **/etc/shadow**. When a distro has shadow passwords enabled, **/etc/passwd** is used as before, except the password field is replaced by an 'x'. **/etc/shadow** also may include useful information regarding user passwords such as number of days since a password change and number of days after expiry before an account is disabled. [1]

8 Unshadowing the Password File <>

- Enter terminal in Ubuntu and enter the following:
  `$ unshadow /etc/passwd /etc/shadow > mypasswd`
- We've unshadowed the password file, now we can run john:
  `$ john mypasswd`
- Wait for it...
- To see the cracked passwords, we now type:
  `$ john --show mypasswd`

john, being originally designed to detect insecure Unix passwords, includes a utility called **unshadow** that combines **`/etc/passwd`** and **`/etc/shadow`**, effectively unshadowing the passwords and matching them up with their associated user and user attributes. Once this file is cracked, the attacker has a lot of useful information at her fingertips. [9]

Cracked passwords are stored in **`$JOHN/john.pot`**, a non-human readable file. the **`--show`** tag is the best way to view cracked passwords [9].

9 Cracking MD5 Hashes by Default <>

- Run the following command:
  `$ john pass.txt`
- Let john run for 1 minute and then stop execution with Ctrl+C.
- Run
  `$ john --show pass.txt`
- Note how many passwords were cracked
- Delete **`john.pot`** with
  `$ rm ~/.john/john.pot`

This demonstration shows john operating in the default cracking mode order as discussed previously: single crack mode, wordlist with rules, and incremental.

10 Cracking MD5 Hashes with a Wordlist <>

- Run the following command:
  `$ john pass.txt -wordlist:rockyou.txt -rules`
- Let john run for 1 minute
- Stop execution with Ctrl+C
- Run
  `$ john --show pass.txt`
- Note how many passwords were cracked

The **`rockyou.txt`** wordlist is an actual password file leaked from the RockYou website. Download it from the Skull Security wiki. [10]

11 Writing Our Own Cracking Rules with john<>

Writing our own rules can prove to be very useful (e.g. if we know the general form of a password). Rules take the basic form:

<Az |A0>"[<set(s)>]"

- **Az** means "append"
- **A0** means "prepend"
- Sets can be ranges or single characters
- You can add "c" to begin search with capital letters, "l" for lowercase, "r" for reverse, etc. [11]

A good john usage and rule-writing "cheat sheet" can be found at [11].

12 Cracking MD5 Hashes by Specifying Rules <>

We know that a password begins with a capital letter and ends with a number between 0 and 9. Let's crack it!

- Open **`/etc/john/john.conf`** in a text editor
- Add the following in a new line after the line where [List.Rules:Wordlist] appears:
  `cAz"[0-9]"`
- Return to the Desktop and execute the following command:
  `$ john -wordlist:length4.txt -rules rule1.txt`

13 Challenge 1 <>

1. Create an MD5 password hash that is easy to crack by using john. Use john to verify.

   - **Hint:**
     use the following command to create an MD5 hash:
     `$ mkpasswd --method=md5`

2. Now make a password that is difficult to crack. Use john to verify.

14 Challenge 2 <>

The password in **`rule2.txt`** has the following properties:

- It contains a four-letter dictionary word
- It begins with a "+"
- It ends with the number 8

1. Write a rule to crack this password.
2. Verify with john.

15 Questions <>

1. What is an example of one cryptographic hashing algorithm *besides MD5* that *should not* be used to hash passwords? What should be used in their place?
2. Is the default cracking mode or the wordlist mode more effective at cracking passwords? Why is this the case?
3. Can you crack any possible password with a brute-force attack? If so, what would this require?

Answers are in the appendix.

16 Alternative Solution: Rainbow Tables <>

- Another way to crack passwords is to calculate all possible hashes for passwords of a certain length and hash function and to store the passwords and their associated hashes in a table for future lookup.
- This trades speed for storage capacity. To have rainbow tables for the most common hash functions and for passwords of moderate length, a couple terabytes of storage is needed.
- The downside to cracking with rainbow tables is that they are completely ineffective if a salt is added to a password before it is hashed.

[13]

Specialized software is available for cracking passwords with rainbow tables. Rainbow tables and the required software to use them can be downloaded from many places, but one reputable place is Project RainbowCrack. [13]

17 Conclusion <

- Passwords are fragile. Passwords that are easy for people to remember are even easier for computers to guess.
- Even "secure" passwords can be vulnerable to more sophisticated attacks.
- It seems likely that passwords will ultimately be replaced by a newer more secure mechanism, possibly bio-metric in nature that will combine better security with greater ease of use.

18 Appendix: Setting Up the VM, Solutions, and Change-log <

1. Steps for setting up the virtual machine
2. Solutions to the challenges and questions
3. Change-log

Setting Up the VM

1. Start a virtual machine based on Ubuntu 16.04 LTS
2. Install John the Ripper (version 1.8.0-2) with the following commands in a bash terminal:

   ```
   $ sudo apt-get update
   $ sudo apt-get install john-data=1.8.0-2
   $ sudo apt-get install john=1.8.0-2
   ```

   For this tutorial to be guaranteed to work, use only the operating system and software versions listed above. John may behave differently than documented in this tutorial if you use a different version released in a future update.
3. Download the files associated with this tutorial from the same website from which you downloaded or are viewing this tutorial. Extract them to your Desktop or the home directory.

Challenge 1

1. Use, for example:

   ```
   $ mkpasswd --method=md5 > test
   $ john test
   ```

2. Use something like ";lkjdaf09823471092jlj23" instead of "hello".

Challenge 2

1. Rule to add in **/etc/john/john.conf**:

   ```
   A0"[+]"Az"[8]"
   ```

   Command:
   ```
   $ john -wordlist:length4.txt -rules rule2.txt
   ```

Questions

1. SHA1 is another hashing algorithm poorly suited to passwords. bcrypt is a better solution.

2. This is kind of a trick question. Either can really be faster in a minute. In password cracking, a minute doesn't really matter. You have to find the best solution that fits your time and computational resource needs. For example, a good wordlist can quickly find common passwords, but incremental mode can be more effective at solving a random password, given enough time and resources.

3. Yes. For the really tough ones, all you need is massive amounts of time and computing resources.

Changelog:

Password (In)security: Attacks & Prevention			
Ver.	Date	Authors	Changes
v1	Feb. 2nd 2016	Jon Meyer and Jared Zook	First draft of tutorial.
v2	May 24th 2016	Adam Odell	Moderate additions, corrected instructions and commands, and fixed grammar mistakes.
v2.1	July 18th 2016	Adam Odell	Added appendix.

References

[1] Frampton, S. *Linux Password & Shadow File Formats.* Linux Administration Made Easy. http://www.tldp.org/LDP/lame/LAME/linux-admin-made-easy/shadow-file-formats.html.

[2] Openwall. *John the Ripper usage examples.* http://www.openwall.com/john/doc/EXAMPLES.shtml

[3] Goodin, D. *How Crackers Make Minced Meat Out of your Passwords.* Ars Technica. http://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your-passwords/1/. .

[4] Kumar, M. *11 Million Ashley Madison Passwords Cracked in Just 10 Days.* The Hacker News. http://thehackernews.com/2015/09/ashley-madison-password-cracked.html. .

[5] Botezatu, L. *Windows 8 Stores Logon Passwords in Plain Text.* Hot for Security. http://www.hotforsecurity.com/blog/windows-8-stores-logon-passwords-in-plain-text-3914.html. .

[6] Microsoft Corporation *The Importance of Using Strong Passwords.* Hot for Security. https://msdn.microsoft.com/en-us/library/ms851492(v=winembedded.11).aspx. .

[7] Hoffman, C. *How to Create a Strong Password (and Remember It).* How-To Geek http://www.howtogeek.com/195430/how-to-create-a-strong-password-and-remember-it/. .

[8] Johnson, D. *How secure are password managers?.* http://www.cbsnews.com/news/in-wake-of-lastpass-hack-how-safe-are-password-managers/. .

[9] John the Ripper *John the Ripper Documentation.* http://www.openwall.com/.

[10] Skull Security. *Passwords.* rockyou.txt. https://wiki.skullsecurity.org/Passwords.

[11] Count Upon Security *JTR CHEAT SHEET.* https://countuponsecurity.files.wordpress.com/2015/06/jtr-cheat-sheet.pdf.

[12] *Microsoft LAN Manager Hash.* https://en.wikipedia.org/wiki/LM_hash.

[13] *Project RainbowCrack* http://project-rainbowcrack.com/index.htm