

Using ethical hacking to educate users about secure passwords by cracking insecure passwords using readily available software

Robin Snyder
Savannah State University
126 Jordan, P. O. Box 20359
Savannah, GA 31404
912-356-2716
snyderr@savstate.edu
<http://www.RobinSnyder.com>

Abstract

Security involves many tradeoffs. For example, passwords are often used for authentication purposes even though a secure password is not convenient while a convenient password is not secure. This paper (and talk) will discuss (and present) what everyone needs to know about passwords, from an end-user to a system/network administrator. Relevant password cracking techniques are covered that can be used, in certain situations, to determine which users do not have secure passwords. A web-based password cracking exercise used to educate students in a general purpose security course is presented. Other related topics covered include a review of password management systems, hash-coding principles, available biometric-based password systems, etc.

Introduction

On Valentines Day at the RSA Conference, Bill Gates said that "passwords don't cut it.". Although security has become increasingly important [7], in many organizations the password is the gatekeeper for user access to critical information systems within the organization. This paper discusses password security, password crackers, an example educational system, and the use of ethical hacking by system administrators to educate and improve system security for which they are responsible.

The fundamental assumption of security is that any adversary has access to all published information. Thus, one should try to keep the adversary from having access to as much information as possible. This goal is complicated by the goal of giving access to users who need such access to do their work. Exactly who is the adversary? "*We have met the enemy and he is us*" [1, p. 224]. Recognized security expert Bruce Schneier says over and over that security is not just a technical problem. Security involves the human factor and is only as strong as the weakest link [3].

Users can be divided into cooperating users and uncooperating users. A cooperating user is not trying to cause problems, but may inadvertently cause problems. An uncooperating user is a user that is deliberately trying to cause problems. In general, policies that rely on ignorance help against cooperating users but do not help against uncooperating users. For example, hiding files will inadvertently keep most cooperating users from accidentally erasing or changing that file. But, the specifications and access to hidden files is public knowledge. An uncooperating user bent on causing problems will not be stopped by hidden files. For scenario gaming purposes, the user will be the cooperating user while the attacker will be the uncooperating user. The goal then is to allow the user access to the system in order to do useful work while denying the attacker the ability to compromise the system.

Authentication

The authentication problem is that of determining if the user is who the user claims to be. Is the user who is attempting to access the system a cooperating user or an uncooperating attacker? This is not an easy question to answer. In real life, one might recognize the person. Or, the user might possess some attribute (e.g., biometric) or object, such as a driver's license, whose possession and matching attributes identifies that person. Again, there are forgeries and other factors that make the general problem very difficult to solve. For example, on the Georgia license application there is a question about whether the applicant is an identical twin. This information appears on the driver's license. There are many hi-tech ways of assuring that a user is who the user claims to be. Many of these methods involve biometrics such as fingerprints, eye retina pattern, DNA match, etc. If the user has certain information that only the user would have, then that information can be used for authentication. That is, as long as any adversary does not have access to the same information.

In a prison camp scenario, for example, one might ask a suspected impersonator questions from home that only a person from there would know. On the phone, for example, you might recognize the person's voice, but still ask the person a question. For example, a student calls and wants to know their current score. One might ask the student, "What did we do in class yesterday?". Of course, one might not have to give out too many scores using this approach (smile).

A typical tradeoff between complex methods and simple methods of authentication is the concept of a password, passcode, pass phrase, etc. A password is a code that is associated with a user name, often called a login name, and should not be easily guessed. For example, the user name might be "**smithj**" and the password might be "**tigers**".

How can the attacker's job be made more difficult? One way to make the attacker's job more difficult is to make it hard for an attacker to determine valid user names. In addition, one might make it hard to determine what user names go with what users. In general, this can be difficult as most organizations establish conventions for generating a given login name based on the corresponding user name. For example, the following are common login names for the user name "**John Mark Smith**".

jms jsmith jmsmith smithj smithjm JohnSmith JohnMarkSmith JohnMSmith

When duplicates arise, digits are often appended at the end of the user name. Once a few login names are public, many other login names can usually be guessed. Often, the email name for a site is used as the login name.

When user names are not easily determined by an attacker, the best strategy on a failed login attempt is to not let the attacker know whether the login name or the password was incorrect. This will sometimes confuse cooperating users, but makes an attack more difficult for attackers. Otherwise, the attacker can use a divide and conquer strategy to first guess a valid login name and then guess the appropriate password.

Password properties

Here is the game scenario that illustrates the problem, assuming that a user login name is known to the attacker. A user with a login name chooses a password. The attacker then tries to guess the password. If the attacker succeeds, the attacker gets access to the user's account and can impersonate that user. That is, the attacker can do whatever the user could do once logged into that account.

What is the easiest way to get someone's password? The easiest way to get someone's password is to ask them for it. This is a social engineering attack and is the method of choice for most attackers. Why work harder than necessary?

Another popular method is to get the user to download and run a program or ActiveX control which acts as a key logger, recording all keystrokes of the user and sending promising keystroke (i.e., with user names and passwords) to another site. Such techniques are beyond the scope of this paper.

An attacker should not be able to easily guess the password of a user. Of course, there is a finite probability that any password can be guessed. But, if that probability is kept very low, the attacker will not have much success, may be discovered, may give up and try another attack method, etc. A relevant story in this context is about the two campers who were approached by a hungry bear. The one camper started to put on his running shoes. The other camper asked the first camper, "You can't outrun a bear.". The first camper said, "I don't have to outrun the bear. I just have to outrun you!". This is the case in security. If a security system is more difficult to attack than another site, the other site will, more often than not, tend to get successfully attacked.

The next tradeoff is that easy-to-remember passwords tend to be easy-to-guess while hard-to-guess passwords tend to be hard-to-remember. A dictionary may contain 10,000 words which are fairly easy to remember. On average, an attacker would need to try 5,000 words from the dictionary in order to succeed. If an attack can be automated, and no one is monitoring to respond to the attack, there is nothing to stop the attack eventually succeeding in a reasonable amount of time. And, attackers have lists of words that are used as passwords more often than other words.

A randomly generated password, such as 3tX2\$&q!nsP# is hard-to-guess but also hard-to-remember. If users are forced to use hard-to-remember passwords, most users will write the password down and then keep it in an easy-to-find place. Many users will write their password down on paper and put it under the keyboard, in a desk drawer, or just out-in-the-open. This is similar to putting a key to a door under the door mat. In running races, runners will, not wanting to carry their keys during a race, often put the key somewhere on the car (e.g., on a tire). Thieves who know this will look for where the runner puts the key. Then, while the race is going on, get the key and make an attack.

There are ways to take an easy-to-remember password and make it harder-to-guess but not necessarily harder-to-remember. The strategy of changing, say, "i" to "1", "o" to "0", "E" to "3", etc., is also known to attackers, and automated password guessing sometimes uses this strategy. Another way is to create a phrase, such as "If only I could remember this password" and take the first letter of each word, to get IOICRTP. Again, the phrase should not spell an easy-to-guess word.

In a targeted attack, an attacker has a specific target (i.e., user) in mind. In such cases, knowledge about the user can help in the attack. Many users may use the name of their spouse, kids, favorite car, etc., as a password. This can help in targeting an attack. In military history, successful generals often use knowledge of the opposing general to advantage. A well-known example is that of Robert E. Lee in the American Civil War. Lee, as Superintendent of the United States Military Academy at West Point, of which he was a graduate, had occasion to observe the cadets under his leadership. When opposing them in the war, Lee used personal knowledge of what the opposing general would do and would not do to advantage.

If users need to remember multiple passwords, there is little choice but to write them down, unless, of course, all of the passwords are the same. Many organizations try to get a unified password system. This helps users, but then one key unlocks everything. A related issue is that when creating login names and passwords on web sites, many users will use the same user name and password. Thus, a rogue site could allow users to create accounts, and then see if the user has used the same login name and password on other sites.

Understanding lockout

To make it harder for the attacker, a lockout may be used. A lockout happens when a user fails to successfully login after a given number of attempts, and is then locked out from being allowed to login for a given number of minutes. Most ATM's will "eat" a bank card after three unsuccessful login attempts using a PIN (Personal Identification Number) that is one of 10,000 possible 4-digit sequences. Some attackers have tried ATM skimmer technique, whereby an attachment placed on the ATM gets the account number as the card passes through, a small camera watches the PIN being entered, both are sent remotely to the attacker, who can then generate a plastic card with the bank account number.

An important point here is to understand the purpose of the lockout time. If the lockout is set for too long a period, for example, mischievous students could, before class, attempt to login as the teacher and lock the teacher out until the situation can be remedied. If the lockout is for one hour, and a IT support person cannot be located, well, no useful class for today, assuming the teacher needs access to a computer account to use technology in the classroom.

The main purpose of a lock out is not to keep an attacker from guessing, but to slow down an attacker and provide warning to the system administrator that an attack may be in progress. Most attacks are not made by an attacker sitting at a computer guessing passwords. Instead, most attacks are automated by software. A simple example is in order. Assume that passwords are only words in a dictionary of 10,000 words. On average, the attacker must try 5,000 words to succeed on any given account. If the software can try 10 passwords per second, then it would take 500 seconds, or just over 8 minutes, to successfully break into an account. If, instead, a lockout is for 1 minute after 3 unsuccessful attempts, then the same attack would take about $5,000/3 = 1,667$ minutes = 27.8 hours. After the first few lockouts, the system administrator should be notified of a potential attack (e.g., via a portable communication device that supports, say, email messaging). With more secure passwords, the scenario becomes less attractive to the attacker. The lockout thus serves not as a barrier to the attack, but as an alarm system that notifies the owner of a possible break-in attempt and that an appropriate response make be needed.

On the web, if a web server is not monitored, an attacker might make many attacks until one succeeds. One example involved weekend access to the web server by an attacker. The attacker attempted to make more than 10,000 purchases, each for \$0.05 (i.e., one nickel), using made-up credit card numbers. Most failed. A few succeeded. Some credit card users had their credit card numbers compromised. Those customers had never visited this web site. The owners of the web server ended up with a large bill for 10,000 credit card authentications, each of which was on the order of, say, \$0.50. This amounted to about \$5,000. Any site that allows unmonitored access can encounter this problem.

Hashing

A hash, or digest, function converts a value into another value such that, given the second value, it is not easy to determine the first value. For example, here are some hashes of the name "**Savannah**" expressed in hexadecimal.

```
hash value
----
none Savannah
MD4  b08f57ef4e1609cf3cf0b00d1fa4a41f
MD5  cef52e4251b1b52e1da23854efbb0849
SHA1 c31ca1b500b60069c3255a5e1607f5f7f2d01020
```

The SHA (Secure Hash Algorithm), version 1 (i.e., SHA1 or SHA-1) is a very popular hash method. Older methods include MD4 and MD5.

Suppose, for example, that the password were "**Savannah**" (a poor choice as it could be easily guessed). Instead of storing "**Savannah**" as the password, the hash is stored. Then, when the user attempts a login, the hash of the password typed by the user is computed. If this computed hash matches the hash value in the database, access is granted (i.e., the user is authenticated. This is why a network administrator can provide you with a new password, but should not be able to tell you your existing password. There is no easy way to reverse the hash without going through every possible password until a match is made.

Hashes can be made of files to insure that a file is not changed (i.e., tampered). For example, if four CD images required for Fedora Core 4 of Linux from Red Hat are downloaded from some site, the SHA-1 hashes are provided by Red Hat. Checking these hashes ensures both that the download worked without problems and that the image had not been changed (i.e., with a virus, etc.). Here is an example.

```
For x86-compatible (32-bit):
FC4-i386-disc1.iso (shasum: 3fb2924c8fb8098dbc8260f69824e9c437d28c68)
FC4-i386-disc2.iso (shasum: 31fdc2d7a1f1709aa02c9ea5854015645bd69504)
FC4-i386-disc3.iso (shasum: 032455cdf457179916be3a739ca16add75b768b7)
FC4-i386-disc4.iso (shasum: f560f26a32820143e8286afb188f7c36d905a735)
```

Note that `shasum.exe` is a popular and readily available command line program for determining the SHA1 hash of a file. Here is the command line and response for the first file above.

```
[4nt] shasum.exe FC4-i386-disc1.iso
3fb2924c8fb8098dbc8260f69824e9c437d28c68 FC4-i386-disc1.iso
```

The checks can be automated with appropriate software.

Off-line guessing of passwords

A password cracker works via a brute force method using the encrypted password passwords. A brute force method goes through every possibility in an attempt to achieve a goal. A brute force password guessing method uses long lists of possible passwords, computing the hash of each, to try to guess the password. This method does require access to the encrypted passwords.

Here is how off-line guessing of passwords works.

- There is a known algorithm for encryption of passwords, $f(x)$.
- The attacker has access to the encrypted password file, list of y values for each userid.
- The attacker keeps guessing, via software, a password x until $f(x)$ is equal to y
- Once a match is found, the password works on the first try. There is no lockout.

If passwords are guessed online, most systems would detect this and perform a lockout. But, with access to the encrypted password file, one can guess passwords off-line and then successfully login the very first time. Protecting the encrypted password file is thus essential, because if an attacker has access to the encrypted password file, the system can be susceptible to off-line guessing of passwords. Standard threat monitoring techniques such as lockout will not easily detect on off-line guessing attack that uses the encrypted password file.

Most systems will add a "salt" value to the typed password when hashing. The "salt" value is put in the database which makes it more computationally difficult to guess passwords if the attacker get access to the encrypted passwords.

There are password programs online that can be used to recover passwords for programs such as Word, Excel, Access, etc.

Password cracking

Next to asking for the password, the next best way to get a password might be to find the encrypted password file posted on the Internet. Suppose that one does the following Google search.

```
ext:pwd inurl:(service | authors | administrators | users)
```

Here is one such hit that was found on Tue, Aug 23, 2005.

```
# -FrontPage-  
ekendall:bYld1Sr73NLKo  
louisa:5zm94d7cdDFiQ
```

Here is another hit that was found on Tue, Aug 23, 2005.

```
# -FrontPage-  
cannon:QcbaEg14h4ub.
```

In this case, the site had removed this information. But the information was still in the Google cache, saved from a previous visit.

Note that Google can be used to find search terms that can be used to look for various vulnerabilities on the Internet. To keep such files from being accessed via the Internet one might use the `urlscan.exe` program intended for Microsoft IIS (Internet Information Server) or the `.htaccess` file intended for the Apache web server. In both cases, one would disallow a file with extension `pwd` from being returned by the web server (i.e., being accessible directly from the Internet).

What might be done with an encrypted password file? A password is usually encrypted. But the password can often be recovered. A password cracker is a program that allows you to recover or crack a password.

One such password cracking program is "**John the Ripper**", at <http://www.openwall.com/john/> [as of Tue, Aug 23, 2005]. The "**John the Ripper**" program is a command line program. You start the program and type commands at the command line. In the following examples, "[4nt]" is the command prompt. Here is the help information for the program.

```
[4nt] john.exe

John the Ripper  Version 1.6  Copyright (c) 1996-98 by Solar Designer

Usage: john [OPTIONS] [PASSWORD-FILES]
-single                "single crack" mode
-wordfile:FILE -stdin  wordlist mode, read words from FILE or stdin
-rules                enable rules for wordlist mode
-incremental[:MODE]   incremental mode [using section MODE]
-external:MODE        external mode or word filter
-stdout[:LENGTH]      no cracking, just write words to stdout
-restore[:FILE]       restore an interrupted session [from FILE]
-session:FILE         set session file name to FILE
-status[:FILE]        print status of a session [from FILE]
-makechars:FILE       make a charset, FILE will be overwritten
-show                 show cracked passwords
-test                 perform a benchmark
-users:[-]LOGIN|UID[,..] load this (these) user(s) only
-groups:[-]GID[,..]   load users of this (these) group(s) only
-shells:[-]SHELL[,..] load users with this (these) shell(s) only
-salts:[-]COUNT     load salts with at least COUNT passwords only
-format:NAME          force ciphertext format NAME (DES/BSDI/MD5/...)
-savemem:LEVEL       enable memory saving, at LEVEL 1..3

[4nt]
```

Here is the result of running "**John the Ripper**" on the above information that was saved in file `pwd3.dat`.

```
[4nt] john.exe pwd3.dat
Loaded 1 password (Standard DES [24/32 4K])
6496 (cannon)
guesses: 1  time: 0:00:00:23 (3)  c/s: 67959  trying: 6cco - 63ri
[4nt]
```

It took 23 seconds to crack the password for user `cannon`. The password is `6496`. Note that some passwords may take much longer. It depends on the password used.

When restarted, "**John the Ripper**" keeps working on unsolved passwords and does not restart the previous work. Here is one way to confirm what work has been done to this point.

```
[4nt] john -show pwd3.dat
cannon:6496

1 password cracked, 0 left
[4nt]
```

If a password is not cracked quickly, one can let the program run longer, or use additional options to provide additional guidance to the program. One should always make your password hard to crack. One can always run a password cracker on your encrypted password to see how hard it is to crack. Or, one might run the password cracker on the encrypted password file for local users to see who does not have a very safe password. But, do not post the encrypted password file on the Internet where an attacker (e.g., using Google) can find it.

Though not the most efficient, the popular web server processing language PHP (note that it can be used for other things) can be used both to verify the results of John the Ripper and to better see how passwords are encrypted and how a password cracker works.

```
<?
function encode2($pw, $hash2) {
    $salt = substr($hash2, 0, 2);
    return crypt($pw, $salt);
}

function encode1($user1, $hash1, $guess1) {
    $hash2 = encode2($pw1, $hash1);
    echo "<br>";
    echo " user=[" . $user1 . "].";
    echo " hash=[" . $hash1 . "].";
    echo " guess=[" . $guess1 . "].";
    if ($hash1 == $hash2) {
        echo " [ok].";
    }
}

encode1("louisa", "5zm94d7cdDFiQ", "trumpet");
encode1("cannon", "QcbaEg14h4ub.", "6496.");
?>
```

Here is the output.

```
user=[louisa] hash=[5zm94d7cdDFiQ] guess=[trumpet] [ok]
user=[cannon] hash=[QcbaEg14h4ub.] guess=[6496.] [ok]
```

Note the following for FrontPage password files, which potentially provide access to web site authorship.

- The encrypted password file contains the user name.
- The encrypted password file contains the encrypted user password.

- The encryption algorithm is known, the DES (Data Encryption Standard).

One security problem here is that the encrypted password file is not itself encrypted.

In the case of Microsoft IIS, IIS was designed as a single developer server. Providing any user with rights to create ASP scripts allows that user to see anything on the web server that is accessible from the web server, including databases, other user scripts, etc. [6]. This feature is not easy to remove without crippling the functionality of the web server. Thus, cracking one FrontPage password of a user with ASP create rights provides access to the entire web server, the files it can access, and the databases that it can access. For this and other security reasons, many web servers will not enable FrontPage server extensions.

An actual password cracker would make guesses until the guess hash matches the actual hash. Besides using algorithmic methods, here are some common passwords near the top of the list.

```
12345  
abc123  
password  
passwd  
123456  
newpass  
notused  
Hockey  
internet  
Maddock  
12345678  
newuser  
computer  
Internet  
qwerty
```

If your password is on this list, it is probably not very secure.

Ethical hacking

When teaching password cracking techniques, it must be stressed to the students that the techniques are for educational purposes and should not be used to attack systems. Password cracking should be used ethically.

In ethical hacking, for example, a network administrator might use the encrypted password file and a "cracking" program to determine who has not picked a good password. The human part is in letting the users whose passwords have been cracked that they should pick another password (e.g., by having their passwords expire). Some systems run a cracker program on a user's selected password and will not let the user pick a password that is easily cracked. This happened to the author in 1988 at the IBM R&D Center. It took several attempts to get an acceptable password. The password "qwerty" did not work. But another "pattern" on the keyboard did work. Apparently that pattern was not forbidden.

Educational system

The author has developed a web-based educational system for password cracking that uses John the Ripper.

The system works as follows from the student point of view. The student starts the exercise with the FrontPage user name and encrypted password. The student uses John the Ripper to crack the password. Once cracked, the student is presented with another password to crack. After 5 successful attempts, the student answers a short answer question about the exercise.

The system works as follows from the teacher point of view. First, an XML file containing the user names, passwords, and encrypted passwords, and actual passwords is created. Here is an example fragment of the XML for a two-step sequence (for just the teacher, for example purposes).

```
<?xml version="1.0"?>
<rmsSecures class="CISM3300" term="Fall 2005">
<account type="teacher" login="snyderr" name="Dr. Robin Snyder">
<challenge index="0" done="0">
# -FrontPage-
mary:Ds81AR.h3maHY
</challenge>
<challenge index="1" done="0" name="mary" code="password">
# -FrontPage-
jenna:DsKvHJgx2b7DI
</challenge>
<challenge index="2" done="1" name="jenna" code="amber6"/>
</account>
</rmsSecures>
```

Each student receives a different sequence of user names and passwords. The author has been using this approach because while students may not mind showing other students their work or answers, they may mind actually doing the work for the other student. Each password is checked to insure that it will actually be cracked in a short period of time. When the student completes the sequence and answers the question, the result is submitted into the author's submission and annotation system, making the submissions easy to grade.

Summary

This paper has provided an introduction to password security and cracking of passwords using readily available software. Hackers have such software and use it when necessary. System administrators should have such software and use it ethically to both educate users and improve the security of the systems for which they are responsible.

References

- [1] Kelly, Mrs. W., & Couch Jr., B. (eds) (1982). The best of Pogo. Simon & Shuster.
- [2] McClure, S., Scambray, J., & Kurtz, G. (2005). Hacking Exposed. New York: McGraw-Hill Osborne Media
- [3] Schneier, B. (2000). Secrets & lies: digital security in a networked world. New York: John Wiley & Sons, Inc.
- [4] Simpson, M. (2005). Hands-On Ethical Hacking and Network Defense. Boston, MA: Course Technology

[5] Smith, R. (2001). Authentication: From Passwords to Public Keys. Boston, MA: Addison-Wesley Professional

[6] Snyder, R. (2003). IIS security considerations of the FileSystemObject for cooperating and uncooperating users. Proceedings of the 31st Annual Conference of the International Business Schools Computing Association. Daytona Beach, FL.

[7] Snyder, R. (1994). Proactive approaches to information systems and computer security. Proceedings of the 27th Annual Conference of the Association of Small Computer Users in Education. Myrtle Beach, SC.