12 Authentication

In this section you'll learn about authentication which basically speaking is the login process used to allow authorized users to access a system or set of data, while keeping unauthorized users out. You'll learn more about hashing, which was introduced in the Cryptology section, about different ways for users to prove their identify, about things like two factor authentication or 2FA, and about cracking passwords and password strength.

Objectives

At the end of this module, you will be able to:

1. Describe the difference between encryption and hashing.
2. Provide an example of a one-way function.
3. Describe the threat presented by insiders.
4. Describe the different types of authentication credentials.
5. List the steps in the password process.
6. Describe how network login systems such as Kerberos, TACACS+, or Internet login systems such as those offered by Facebook and Google function.
7. List the features of brute force attacks, dictionary attacks, and rainbow table attacks.
8. Crack passwords.
9. Evaluate passwords for strength.
10. List features of strong passwords.
11. Describe a method for creating strong passwords that are easy to remember.
12. Explain why users are the weak link in authentication.
13. Describe ways to train users to avoid phishing attacks.
14. Describe ways to implement password policies that ensure strong passwords are used.

Section Content

Introduction

In this chapter you will learn about authentication, which is the process used to ensure users or programs only have access to the resources they are authorized to use, and about hashing and password systems which are at the heart of authentication.

Like every other chapter the expectation is not that you become an expert on password and authentication systems, but that you start to learn the terminology. And my personal hope is that you're able to take what you learn more about the various types of authentication and use this knowledge to make your passwords stronger, and to choose the best method for protecting devices like your mobile phone.

The material in this chapter covers a lot of ground, but never gets into much detail. If it leaves you wanting more, you can take my cryptology and forensics classes where you'll learn all about how passwords are created and cracked. And if you take my Linux Administration class or some of the Windows Administration classes you'll learn about setting up the rules for ensuring that users choose strong passwords.

Authentication

Let's start by discussing authentication and why it's an important concept in Cyber Security. If your main experience as a computer user is with your home computer preventing users from doing things on a computer might seem a little foreign. That is, when you use your own computer, you're probably accustomed to doing anything and everything. You can install any program, and delete or edit any file, login whenever you want etc. It would make your life, at least your computing life, much more difficult if you had to get permission from someone else or from a government agency anytime you wanted to install a new game or delete a file. Maybe

you were in this situation when you were a child and had to ask your parents for permission anytime you wanted to load a new game and can remember how much time and effort that took.

But now you're an adult, or at least you're no longer a child, and you're starting to assume the parental responsibility of making sure things run smoothly, or at least don't devolve into complete chaos. As you start your career in cyber security you'll be moving into a world where you will be tasked with administering computers and systems that will be used by multiple people, and you'll be the one who decides what programs can be installed or run. In this world there needs to be some way to control what users do. Can you imagine the problems that would occur if there weren't some way to restrict what users could do? How would a system like Amazon.com or eBay even be possible if any user could modify any listing, or modify any account? Imagine if on Amazon a company could go to pages owned by their competitors and change the prices and/or ratings. Or imagine the chaos if a dishonest eBay or Amazon user could change all the account settings so that incoming payments were transferred to their bank account instead of the person who actually sold an item.

Simply trusting users to do the "right thing" kind of works with certain sites like Wikipedia. But even Wikipedia has had problems with users behaving poorly and doing things like defacing pages or entering incorrect information. In any case, simply trusting users isn't an option where sensitive data or finances are concerned. Which is why authentication systems were developed, and why you'll spend this chapter learning about authentication.

Authentication is the name given to the system or process for both checking to see that someone is who they claim to be, and to use their identity to secure devices and information from unauthorized access. The main way of doing this is by using a password system. But there are also other methods such as biometrics or physical devices such as key cards. This is a pretty big and important topic in both cyber security and in real life as it's the first line of defense used by most organizations for protecting your data.

Like all the other chapters, the main expectation from this section is that you'll start to learn the terminology. In addition to learning the terminology, there are a couple of interesting larger points that I think you should take away from this section.

The first point is that the people who can do the most damage to a company have always been the employees, or Insiders[1]. If you stop and think about it for a minute, it makes sense. An outside attacker needs to go through a lot of work just to figure out how to get into a single computer or device. They have to find a way to gain access, either by phishing and getting the credentials of an authorized user, or by taking advantage of a vulnerability and breaking in. Once they're in they have to do quite a bit more work to locate the data they want, and then find a way to gain access to it. After they have access to the data, they still have to find a way to sneak it out without anyone noticing. And they have to do all of these actions without being detected and without setting off any alarms.

But an Insider already has access to the computer systems, knows where the data is stored, and knows how to sneak the data out without anyone noticing or setting off any alarms. In fact, they may have been involved in setting up the security systems, so it's easy for them to find ways around it. Or they may have even left themselves a back door to make getting data in and out easier.

And here's one of the ironic things about cyber security. Who knows the most about the computer and network security at a company and presents the biggest risk? That's right, the cyber security specialist. If you get a job as the cyber security specialist at an organization, you'll have intimate knowledge of the company's weak spots and vulnerabilities. This privileged information will make it much easier for you to steal information than almost anyone else at the company.

[1] https://www.ekransystem.com/en/blog/insider-threat-statistics-facts-and-figures

In any case, the dangers of insider attacks are why some of the more extreme measures described in the book were developed. Things like mandatory vacation aren't put in place to ensure that you get away and enjoy yourself, they're in place because apparently no one can be trusted.

I'll close this discussion on insider attacks with a quote from the Roman poet Juvenal "Quis custodiet ipsos custodes". If you don't know what this means, run it through a translator and you'll see why it's appropriate.

The second point that I think may be lost in the details is regarding systems used for a single login. The book talks about a few of these systems, such as Kerberos and TACACS+, but they don't really explain how they're used or why they're important.

The first thing to understand is what's meant by single login. Hopefully you understand the importance of requiring users to login before allowing them to access a computer or other resources. Forcing users to prove their identity is the main way that we protect a computer, and it provides an excellent first line of protection if implemented correctly. However, the login process also comes with a cost, which is the effort required to remember your username and password and the time it takes to enter them. This cost is very minimal in relation to the protection it provides, but it's important that you recognize it as a cost for reasons that will soon be made clear.

The login process isn't bad if you only have to login once, but it can be annoying if you have to login repeatedly. There are two situations that can cause this:

1. If the system times you out too quickly. All Operating Systems have a setting to somehow protect an account if the user is idle for some period of time. On multi user systems this is done to ensure that you haven't logged in and then accidentally walked away from your computer without logging out. On your home computer, it's more likely done as part of the power savings scheme.

If you want to see how this could possibly be annoying, change your computer's Power Saving settings so that it goes to sleep after 3 minutes of idle time. Or change your phone settings to go to the lock screen after 5 seconds of being idle. Even though logging in any single time doesn't take long, I can pretty much guarantee you're not going to be happy after constantly getting logged off or locked out and having to repeatedly log back in.

2. If you have to login again to access different resources. In most cases, when you login to a network, such as the CBC network, you are granted access to multiple resources. For example, when you login to a computer connected to the CBC network you also gain access to several printers, probably a file server or two, and the campus network including access to the Internet. However, the network could be configured so that you would have to login again to gain access to any of those resources.

I've worked with one network administrator in the past who used to take care of CBC's Richland campus, and this is exactly what he did. Students and faculty had to login to Windows to gain access to the computer, and then login again to reach the network server where all the programs were stored. Even basic programs like an Internet browser or Microsoft Office were password protected so you had to login to the server to run a program. And you had to login again to access the printer. To make things worse he set the timeouts to 5 minutes. If you didn't use the printer for 5 minutes, you would have to login again when you did want to print. It was very secure, but it was also very, very, very annoying.

You still have to do a bit of this, when you access Canvas or when you access your CBC Cloud VM. That is, you have to login into your Windows computer, then login to the CBC Cloud to access your VM, then start the VM and login again.

Does it seem like you should only have to login once to access any resource on the CBC network? It would sure be a lot more convenient, but doing so would weaken security on those resources. So, in the case of the CBC network, a lot of thought has been given to what resources you're able to access with your primary login, and which resources will be protected by an additional login.

The point I'd like you to see is that at one extreme, you could require only one login to access everything on a network, or at the other extreme you could force the users to login to access every single resource. The single login is more convenient for the users, but less secure. While logging in to access every resource is more secure, it can be really annoying and inconvenient.

As an analogy imagine you are in charge of providing the physical security for a building and the rooms in a building. The building could be your home, or it could be an office building or laboratory. Door locks would provide an excellent first layer of protection for any type of building. They might be simple key locks, or you might use Cipher locks so that each person could be given their own access code and then you would know who accessed the building at any time.

Your next decision is whether or not you want to provide extra security on any room inside the building. You could have put locks on every door, the doors to each room, the doors to each closet, the doors on any cabinets or even add locks to individual drawers. This would make the overall security much stronger and make it much more difficult for unauthorized people to do anything in the building. But it would also make it much more difficult for actual authorized users to get anything done.

Adding all the additional locks might make sense if you were trying to protect a building that contains weapons or biological hazards, but it wouldn't make much sense in your home. Imagine how happy your significant other and children or parents would be if you forced them to lock and unlock every single drawer and cabinet in your home. It would be much more secure on the off chance that a burglar broke into your house, but this extra security probably isn't worth the inconvenience to the people living in your house.

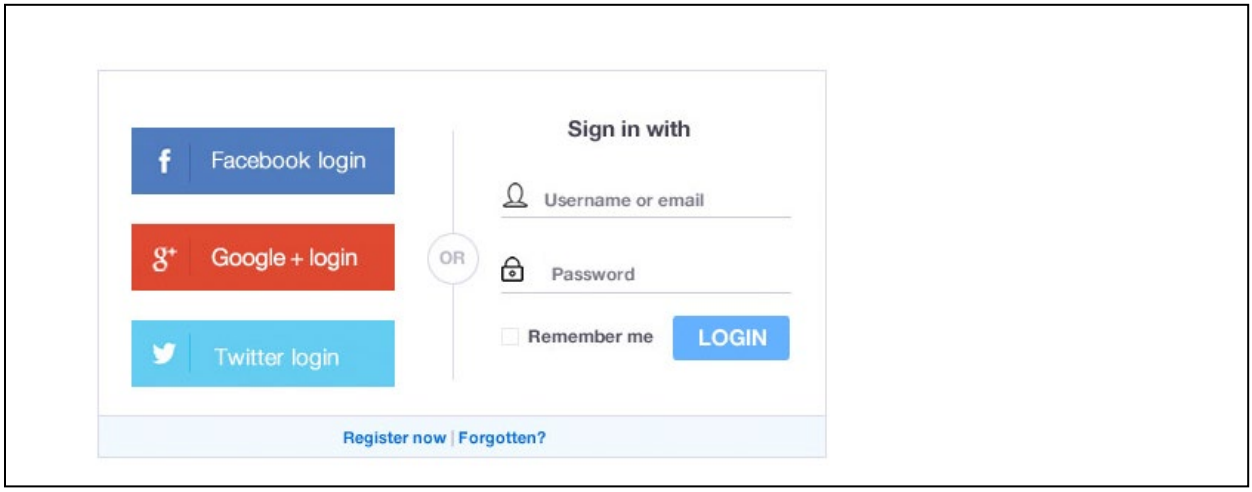
Just as we typically use just a single key to unlock a house, most smaller networks use a single login to access most of the resources on the network. This is what is meant by single login. In a single login system, when you login you're given a set of credentials. You can think of these credentials like a driver's license or passport, or any set of credentials that have been issued by some central governing authority. These government credentials are only issued after some

type of thorough check to ensure that you are who you say you are. This background check takes a certain amount of time, up to a few months for a passport. But in most cases, we agree it's worth the time and effort to get a set of government issued credentials. The great thing about having a driver's license or passport is it's not only trusted by the organization that issued the credentials, they're also trusted by other individuals or other organizations as proof of your identity. While accepting outside credentials is a bit of a risk, it saves these other organizations the time and effort it would take them to do the same thorough background check.

Issuing network credentials is what systems like Kerberos and TACACS+ do. Once you login to a central server, it provides you with a set of credentials that can be used by other network resources to verify your identity. This is also what Active Directory does for Microsoft based networks. In these systems you login once, get your set of network credentials, and then your network credentials are used any time you want access another resource on the network. For example, say you've logged in once, and now want to print a document using a network printer. Without network credentials you'd have to log in to the printer server before your document could be printer. But with network credentials, your computer will send the network credentials you were issued when you logged in to the print server. The print server will authenticate your network credentials automatically, and if you're allowed to use the print server, you'll be able to print without logging in.

Another aspect of this that's become increasingly more relevant is the use of single logins for the Internet. In some sense, the Internet is much like the CBC network, only much, much bigger. That is, it's a big network with many resources. However, even though you've logged in to your computer there are many sites on the Internet that require you to login again before they will grant you access. Of course, it would be ridiculous for any website to trust you just because you logged into Windows on your computer. But the obvious problem with having to prove your identity anew at every website is the number of login credentials you're forced to remember. Think about how many different website usernames and passwords you have, and have to keep track of. Wouldn't it be nice, from a user's perspective at least, if you only had to login once, and then have access to any site on the Internet?

This type of single login for the Internet is what some of the big Internet players like Microsoft, Google and Facebook have been working on. Take Google for example. Once you create one Google account, you'll also have access to every other site and Google service such as Gmail, Google Docs, Google Calendar, YouTube, etc. If you use different Google tools it's super convenient to only have to login once, and not each time you want to access a different service. On top of that, you may also be granted access to other websites outside of the Google ecosystem. You may notice the growing number of websites that give you an option of logging in using your Facebook or Google credentials. The way this works is that Google, Facebook, Microsoft etc. publish something called an Application Programmable Interface which is commonly known as an API. The API allows other sites to write programs that make use of the main sites login service and user database. This reduces the amount of work required for the other sites, as they don't have to maintain their own database of user login credentials. And it makes it simpler for users, as they won't have as many usernames and passwords to remember.



In any case, this is an interesting aspect of Access Control that is fairly pertinent to any user. Hopefully it helps add some context to the technical discussion on Kerberos and TACACS+.

Just in case you're an experienced programmer and want to know a little more, here are a couple of sites that contain information on using the Google and Facebook APIs to authenticate users on your website. This information gets technical pretty quickly, and will only make sense if you're a programmer. There's also a link describing how Facebook went from using Kerberos internally, to using something called TLS.

https://developers.google.com/identity/sign-in/web/sign-in

https://developers.facebook.com/docs/facebook-login/

https://engineering.fb.com/security/service-encryption/

Passwords & Hashing

In addition to the material in the book, here's some information that I think you should learn about passwords and password systems. There's a bit of technical background to wade through, but I think that understanding how most password systems work will help you create stronger passwords for yourself.

The first thing to understand is that passwords are hashed and not encrypted. Hashing and encryption are two completely different processes, and it's critical that you understand how they both work. You need to understand the difference to understand how modern password systems function.

Most people are familiar with encryption, and it might seem like passwords should be protected by encryption, but they aren't. Passwords are protected by hashing.

It’s actually possible to set up a password system that uses encryption but it would have several holes. In this type of system when a user sets their password it would be encrypted and stored. When the user tries to log in the system decrypts the stored password and checks to see if it matches the password supplied by the user. However, using encryption to protect passwords has a couple of huge weaknesses. The main problem is that if an attacker had access to an encrypted password file it would be relatively easy for them to decrypt every password on the system.

You might know enough about encryption and decryption to know that decrypting the password file requires the key. But as you’ll learn in later classes it's hard, almost impossible, to keep the keys secret. Or another way to think of it is to say you had some information in your house that you want to keep secret, so you lock it in a box. Only people with access to the key will be able to read the information. The problem is you need to hide the key somewhere in the house. If someone has access to your house, they can just look around until they find the key. Anyone who finds the key will have access to all the information in the box. With computer systems, it’s pretty easy to figure out where the key is hidden. It’s relatively simple for an experienced individual to run the encryption program, and by monitoring which files are modified, figure out where it stores the keys. In fact, most encryption systems don’t even try to hide their keys. Instead, they treat the keys like passwords and protect them by hashing them instead of trying to hide them.

Which brings us back to hashing. The technical difference between encryption and hashing was explained in a previous chapter on cryptography, but this concept can be a little hard to understand until you actually see it applied, like with passwords. Remember that with encryption anything that is encrypted can be decrypted. But hashing is much different, as the hashing process is one-way. Being one-way means that there’s a formula for hashing information, but there is no formula reversing the process and de-hashing information.

It's easy to memorize the fact that hashing is a one way function, but what does this really mean? What is a one way function? Here are a few examples that should help explain the concept.

The first is the clock analogy. To start the function or process we need to be given a time in hours, minutes and seconds, that time is our function input. The function will take the input and drop everything except for the seconds. For example, if the time is 6:05 PM and 29 seconds, we'll drop the hour and minutes, and just keep the 29 seconds. Or, if we start with 9:42 AM and 17 seconds, we drop the 9:42 AM and end up with 17 seconds. If you try and reverse this, that is if you're given a number of seconds, there's no way to get back to the original time. For example, if I tell you that the result of the process is 11 seconds, can you tell me what the original starting time was? It's impossible to do because there are several starting times (24x60 to be exact) that result in 11 seconds.

The second example of a one way function is water in a measuring cup. The function or process that we'll use this time is to take a container of water and pour it into a 1 liter cup that measures fluid by the milliliter. Or for those of you who are still stuck in the 18th century (LOL) you can assume the cup holds 1 cup total and it measures by fluid ounces. As we pour water from the original container, each time the measuring cup fills up we'll stop pouring, dump out the water in the measuring cup, and then start pouring again. We continue until the original container is empty, at which point we determine how many ml of water are in the measuring cup. For example, if the original container held 4.579 liters, there will be 579 ml of water left at the end. Or, if you start with 7 cups and 6 ounces, when you're done the measuring cup will have 6 ounces. No matter how much water you start with in your original container, the end result will be a number between 0 and 1000 if you use milliliters, or 0 and 16 if you use ounces. But if you're only given a number of milliliters or fluid ounces left at the end of the process, is there any way to determine how much water was in the original container?

The third example of a one way function uses something from math called the remainder function. This is also known as the Modulus function, but most programmers refer to it by its common name which is the Mod function. The remainder function is pretty simple, you give it two numbers, divide the first number by the last, and return the whole number remainder.

For example, if you divide 12 by 5 the whole number remainder is 2. That is, 5 will go into 12 two times since 2*5=10, but we don't really care about that. What we want is the remainder that's left after we take the most 5's we can out of 12, which can be found by subtracting 10 from 12. For the remainder function all we care about is that the whole number remainder after dividing 12 by 5 is 2.

Or if you divide 42 by 5, the whole number remainder will also be 2. Or in this case, 5 will go into 42 eight times since 8*5=40, but once again we don't really care about that. For the remainder function all we care about is that the remainder after the division is 2.

To turn this into a one way function you would keep the second number fixed, but vary the first number. Say for example we wanted to always mod numbers by 7. Whatever number you provide will be divided by 7, and then the remainder will be calculated. The following table shows the remainder that results from calculating the remainder after dividing all of the numbers between 0 and 32 by 7. As you can see this is a one way function, as there are many different numbers you can start with that have the same remainder after you divide them by 7. That is, if I told you the result was 3, would you be able to tell me what the starting number was? Looking at the table you can see that 3 Mod(7) results in 3, but so does 10 Mod(7), and so does 17 Mod(7), etc.

Number	Result
1	1
2	2
3	3
4	4
5	5
6	6
7	0
8	1

Number	Result
9	2
10	3
11	4
12	5
13	6
14	0
15	1
16	2

Number	Result
17	3
18	4
19	5
20	6
21	0
22	1
23	2
24	3

Number	Result
25	4
26	5
27	6
28	0
29	1
30	2
31	3
32	4

Calculating the remainder is really just what the clock and measuring cup examples were doing, but in this case, we're just doing it with pure math instead of some real world construct. For example, with the clock example we get rid of the full number of hours and minutes, and only pay attention to the remaining seconds. This is the same as modding things by 60. Or with the measuring cup example we end up ignoring the number of full liters or full cups and only look at the remaining water. This would be the same as modding by 1000 for milliliters, or modding by 16 for fluid ounces.

The last example of a one way function uses playing cards. The description for this one is a little more complicated, and might be hard to follow by just reading the description. But it's pretty simple if you can see it visually, so I made a video to demonstrate how it works.

https://tonysako.com/home/cs150-introduction-to-computer-security/intro-to-computer-security-authentication-hashing-vs-encryption-cards/

The process can be done with any size set of playing cards, but without any face cards. That is, the cards are normal playing cards, but there can be any number of cards. The function or process works by taking the first four cards off the top of the stack of the cards to be hashed and adding their values. After adding the four card values together, you divide the total by 10 and the hash result is the remainder. For example, if the first four cards are an Ace, 7, 3 and 6

the total is 1+7+3+6=17. Dividing this by 10 and taking the remainder results is 7 as 17=(1*10)+7. The next step is to remember the 7, discard the original four cards and draw the next four cards off the top of the stack of cards to be hashed. Add the 7 to the values of the four cards, divide by 10 and take the remainder. Repeat this process until all of the cards to be hashed are used. When you're done, the last remainder you calculated will be the result of the function.

Here is a simple example. Say you start with 3 cards in your deck. The top card is a 5, the second card is a Queen, and the last card is a 2. You start by adding the 5 and the Queen together, resulting in 17, which is larger than 13 so you subtract 13 and get 4. You discard the 5 and the Queen, but get a 5 card. You now discard the 5 and the Queen, take the next card from the deck, which is a 2, and add it to the 5 in your hand, which results in 7. You now discard the 2, leaving you with the 7 card. Knowing the result is a 7, is there any way for you to determine what the original cards in the deck were, or how many cards were originally in the deck? Even if you knew there were originally 3 cards would you be able to determine what they were?

The clock, measuring cup, and card examples are all simple hash functions, and obviously way too simple for securing passwords. But they do provide good illustrations of how one way functions work. The important thing to comprehend at this point is that in each case, there are an infinite number of inputs that end up with the same result. So even if you know the end result, there's no way to work backwards and to get the starting values.

In the Cryptology chapter you learned about several hash functions such as SHA-512, RIPEMD or MD5. One of the things that these cryptologic hash functions do is produce output in a way that can't be reversed, just like the simple hash functions you just learned about. But the cryptologic hash functions all use some math algorithms to scramble the bits from the input and produce their output.

For a hash function to be useful in cryptology and for storing passwords it needs to meet a few other criteria besides just being one way. These criteria include:

1. Every output is the same size, regardless of the size of the input. This makes it harder to guess how big the input is, and removes any clues about the input size.
2. No two inputs will produce the same output. This is a big problem with our simple examples as they all produce output that will be from a very limited set of numbers. But if we want to use hashing to store and verify passwords, we need each password to produce a unique hash value.
3. Small changes to the input value will result in large and unpredictable changes to the output value. This is required to remove any clues that may help an attacker reverse engineer what a hash is doing.

Modern password systems use one of the cryptologic hash functions to store and verify passwords using these steps:

1. When the user creates or changes their password, the password is hashed before it's stored.
2. During the login process, the user will provide their password. The provided password will be hashed, and the hash will be compared with the stored hash. If the hashes match, the user will be logged in. If the hashes don't match the user will not be logged in.

Notice that in the password system there's no way to recover the original password from the stored hash. Even the system administrator cannot de-hash passwords, since it's impossible to run the hashing process in reverse. When a user logs in and supplies their password, the only way to know if it's the correct password is to hash the supplied password and compare it to the stored password hash. And this is why when you forget your password on most systems, you will have to create a new one. No one can look at the stored hash and tell you what your

original password was. The only thing that can happen is to allow you to enter a new password, which will create a new password hash.

Using hashes provides much better protection for the stored passwords as compared to encryption. Once again, this is because anything that's encrypted can be decrypted, but anything that's hashed cannot be un-hashed. Plus, when a piece of data, like a password, is encrypted, the result will be the same length as the original data. For example, if you encrypt the 5-character string **12345** the result will be something that's also 5 characters long, like **C45B9**. But, if you encrypt a 12 character string like CryptoisK00L the result will also be something 12 characters in length, like **5E5D625E00B6**. Knowing the length of a password provides a valuable clue to anyone trying to crack the password, and will greatly reduce the amount of time it takes to find the correct result.

So, hashes are better for storing passwords than encryption, but they don't provide perfect protection. One of the weaknesses of using hashed passwords is that there are only a few good cryptologic hash algorithms available and in use, and it's common knowledge as to which hash function is used for the password system with every OS. This means that if an attacker can get a copy of the stored password hashes, they can try to crack it by using the following steps:

1. Guess a password.
2. Hash the guessed password.
3. Compare the hash from the guess to the original password hash.
4. If the hashes match, then the guessed password is correct.
5. If the hashes don't match, go back to step 1 and repeat the process.

The book describes several different variations of cracking attacks, such as brute force attacks, dictionary attacks, and rainbow table attacks. You'll actually use these different attacks to crack passwords in the Forensics class, at which point they'll probably make a lot more sense. But for now, just try and learn the basic terminology.

One of the things that the book doesn't mention is that one of these attack types will *always* succeed. Yes, there's one type of attack that's able to crack any hash. This is the brute force attack. Because the brute force attack tries every possible character combination, given enough time it will crack every password. The book mentions that the brute force attack is the most "thorough", but what this really means is that it can be used to crack any password eventually.

Don't panic though. When I say that brute force attacks will succeed "given enough time" or "eventually", the time frame on this can be thousands or even billions of years, or longer! The actual amount of time it takes will depend on the strength of the password. When we refer to password strength we're actually talking about three different parameters:

1. The length, or number of characters in the password.
2. The complexity, or types of different characters used in the password. For example, it can be a requirement that your password contains at least 1 character from upper and lower case letters, numbers, and special characters. Using different types of characters makes it more difficult for brute force attacks as it increases the number of characters that must be tested.
3. Avoiding common or dictionary words. For example, don't use **password** or **12345678**. This doesn't help protect against brute force attacks at all, but it does help protect against dictionary attacks and rainbow table attacks.

Applying What You've Learned to Your Home System(s)

My hope is that now you've been armed with some information on passwords, hashing, and password cracking, you'll be able to use determine whether you're using strong passwords, or if you need to step up your game when you select passwords. And rather than just making this a thought experiment, you'll be testing your passwords in one of the assignments for this section.

Here's another tip I can give you regarding passwords. It's some advice that I think is pretty valuable as it's saved me a lot of time and effort. One of the problems with modern life, if

you're not a Luddite[2], is that you're going to have to create a lot of online accounts, which means creating lots of passwords. And since you don't want to use the same password at more than one site, this means creating and remembering lots of different passwords. I don't know about you, but I have dozens of passwords for things I think are important, like my email or online bank, and hundreds of other passwords at sites that force you to create an account before you can access any of their material. (As you probably know, remembering passwords is a big enough problem that there are many different companies selling password management services.) Trying to come up with unique, but strong passwords; and then remembering those passwords used to be a real challenge for me. But somewhere along the way I came up with a system for selecting passwords that not only makes it easier to remember, but also creates very strong passwords.

My system consists of creating passwords with three pieces. The first piece is a phrase that includes characters from all of the various sets. I'm not going to tell you the actual phrase I use, LOL, but it's something like: **#8Ball@**. Next, I add a portion of the name of the site where the account is being used. For example, for the Columbia Basin College website I would add the characters: **CBC**. Then I finish by adding another stock phrase, to ensure that the password has plenty of length. For example, I might use: **!ceCream**. In this case the resulting password is: **#8Ball@CBC!ceCream**, which is pretty darn strong. But even though it's long and complex, making it difficult to crack, it's also easy to remember as the only portion that changes from any of my other passwords is **CBC**.

[2] [1] https://www.history.com/news/who-were-the-luddites - Just in case you don't know what a Luddite is, and you thought I was making fun of some obscure religion.

Ways to Check Your Comprehension

The test over this chapter won't be for a few weeks, so if you want check your comprehension you can use the review questions at the end of the chapter or use the Practice Test. I don't have the review questions loaded in Canvas, so you'll have to just read them and figure out your answers on your own. Or you could try and connect with some other students in the class and drill each other using these questions.

If you want to use the Practice Test, look in the Test 2 Canvas Module for the link to a Practice Test. You can take the Practice Test as few or as many times as you want. You're not required to take the practice test, but it's also a good way to check your comprehension and prepare for the "real" test.

The Activities for This Section

There are two sets of activities for this section, Hands-On Projects 12-1 and 12-6, and Case Project 12-1. All these assignments were designed to help reinforce what you have learned about passwords, authentication, and identifying users. To get the most out of the exercises for this chapter, I'm going to suggest that you do Case Project 12-1 first. Although it's a writing assignment, it's different than most writing assignments as it will show you how to select passwords to make them stronger and more impervious to cracking. Next, do Hands-On Project 12-1, which will show do more than give you an estimate of how long it takes to crack passwords, it will show you that it is possible to crack password hashes, especially if a user selects a weak password.

1. Required Case Project Homework (Writing Assignment)

The writing assignment for this section is **Case Project 12-1 Testing Password Strength**. This project is a little different than the previous writing assignments, in that you need to do some hands-on work testing passwords before you do any writing. The book says to write a 1 paragraph summary of your findings, but you can write more than that if you wish.

The book says to test 3 passwords that are very similar, but I want you to test 4 passwords and make sure that the passwords you test meet the following requirements:

1. The first password must be 8 characters in length, but contain only upper and lower case letters. For example: **HelpDesk**

2. The second password must also be 8 characters in length, but it must also contain at least 1 lower case character, 1 upper case character, 1 number, and 1 special character. For example: **H3lpDe$k**

3. The third password must be 10 characters in length, and can contain any mix of characters. For example: **HelpDesk22**

4. The third password must be 12 characters in length, and can contain any mix of characters. For example: **HelpDesk@CBC**

The big concept that I think you should be able to discover from this assignment is what makes passwords harder to crack, adding more characters, or using characters from different character sets? You can test more passwords if you like, but make sure and answer this question in your paper.

The websites listed in the book may no longer exist, but if you do an Internet search for something like “online password strength test” you should be able to find plenty of alternatives.

Another thing you should think about before starting this assignment is “Do you trust the sites you’re using for testing?” These web sites know what IP address you’re coming from, and if you use your real passwords for testing, they may be able to tie that information back and link it to

your accounts on the sites you frequently use. (If you want to know how this works, do some research on 3rd party cookies. And no, it's not the chocolate chip kind of cookies, or the birthday party kind of party. But a macadamia nut cookie does sound good right about now.) My point is, do be safe when you do this assignment. The passwords you use for testing should be something similar to your real passwords in that they have the same number of characters and maybe similar characters, but you shouldn't test your real passwords.

Hopefully you remember how your paper must be formatted, and the other guidelines for writing papers. Even though this paper is a little different, you still need to include all of the required information, such as the proper Header, and a References section. If you don't remember what's required you can refer to the Written Project Guidelines document for details on how your paper/report will be graded. You can find the document at:

https://tonysako.com/writingprojectguidelines2021/

2. Hands-On Project 12-1 Using an Online Password Cracker

In this exercise you'll gain some experience using a password cracker that utilizes a rainbow table attack. During the exercise you'll hash some strings using the MD5 hash, and then see if the online tool can find matching hashes in its rainbow table. If the online tool finds a match, then it will be able to tell you the original string, essentially cracking the hash and the password.

As you do the steps in this exercise you'll create some MD5 hashes. The book instructs you to use a web site to create the hashes, but you can also create the MD5 hashes using the HashCalc tool you previously installed.

The web site you'll use it able to crack passwords created using most of the common hash functions, but only those built from words and strings from a specific dictionary. The dictionary

is very large, 190 GB for the MD5 and SHA-1 hashes, as it's built from every password dictionary they could find, plus every word in the Wikipedia database. But, once again, this will only crack passwords made up from the strings in the dictionary, which you should take into account when you create your own passwords.

Oh, and it might seem that this site can crack every password quickly, but in practice that's not the case. The people designing password systems know how quick and easy it is to run a dictionary or rainbow table attack and crack passwords. To mitigate this, they've made password hashes much harder to crack by adding something called a salt. A salt is a random looking string that's added to the beginning of every password before it's hashed and stored, and this salt will be different on every system. For example, let's assume a salt for a system is **g*y53B)**. If you set your password to the terrible choice **password**, the salt will appended changing the string to **g*y53B)password** before it's hashed. Adding the salt will make it much more difficult for an attacker to crack the password. You'll demonstrate this to yourself in the exercise.

Password cracking is an interesting skill to learn and if you're interested in password cracking you'll learn how to do crack passwords using a better, easier to use tool, if you take my Forensics class.

What to submit for Project 12-1:

- Make screenshots after steps 9 and 17, and answer questions 10 and 18.
- Do step 19, ensuring that you find a string that the web site cannot crack. Create a screenshot showing the result of entering your uncrackable string, after you press the **Crack Hashes** button. That is, I want you to show me that the tool was not able to crack your password string.
- Answer question 20.

3. Hands-On Project 12-6 Using Windows Picture Password

In this exercise you'll gain some experience using a system that could be used to replace passwords. You can do this on your own system or you can do it on one of the computers in the CS Labs, but you may not be able to do this on the VM. If you cannot do complete this assignment, there's an alternate assignment that you must do instead. The instructions for this assignment, Picture Password Training Simulation, can be found below in #4 [Alternate] Hands-On Project.

What to submit for Project 12-6:

- Make screenshots after step 9, and answer question 18.

Before submitting your work, add all of the information for both Hands-On Projects into a single document. Make sure that this document has the proper header information (your name, project number, date) and well as the project and step number for each item in the document.

4. [Alternate] Hands-On Project - Picture Password Training Simulation

Note that this assignment is an alternate for the previous assignment. If you completed the previous assignment you don't need to do this one. In this exercise you'll gain some experience using a system that could be used to replace passwords. The instructions for this assignment are not in the book. To complete this assignment, do the following:

- Go to the following URL and complete the simulation: https://play.stmath.com/raft/resources/interactive_activities/pp_simulation/

  The simulation is pretty straightforward, but there are a few places where the steps you need to take may not be obvious. If you get stuck or need help, I've made a video that

demonstrates running the simulation. You can find a link to the video in Canvas, or use the following link:

https://tonysako.com/home/cs150-introduction-to-computer-security/intro-to-computer-security-authentication-threats-attacks-picture-password-alternate-assignment-demonstration/

- Create a screenshot of the last page in the training.

Optional Activities

Have you been pwned?

One of the problems with passwords for online sites is that you have to rely on the site owner to secure their password database, and many of them haven't been too good at doing this. There are thousands of websites that have been compromised lost their password database to hackers. This probably isn't a problem if you always use strong passwords, as the hackers will still have to crack the passwords. But it leaves you with the problem of wondering if you've been pwned (pronounced poned), which is slang for being "owned" or in this case having your personal data compromised in one of these attacks. Luckily a few individuals and groups have written programs that will check sites that publish the database dumps from these attacks to see if your account is among the ones that have been leaked. In this exercise you'll check your accounts to see if they're included in any of these database dumps. You'll use a couple of sites that I've found, but as always, they may no longer exist. If this is the case, you can use the general information from this exercise and search for sites on you own.

1. One of the most famous of these sites is **Have I Been Pwned?** Which was created by security expert Troy Hunt in 2013. Connect to the site https://haveibeenpwned.com/ and follow the instructions. You can check any email address or phone number with one easy button push.

2. You can also check the Avast site at https://www.avast.com/hackcheck. Avast's main claim to fame is as a company that provides free anti-virus solutions.