Chapter 2 Authentication, Authorization, and Accounting

"When there is no enemy within, the enemies outside cannot hurt"
African proverb

Security is not a new problem, but has existed and evolved since the origin of man. Implicitly, we have also learned that a system can be envisioned as a boundary with an interface. If the integrity of the boundary is secured, security then relies on the interface and information flow through it.

In this chapter, we will focus on the interface of the system. In particular, we will examine the three components critical in securing the interface. These components are: Authentication, Authorization, and Accounting.

2.1 Introduction

In order to construct a secure system, we must ensure that *only authorized persons can perform certain functions*. In particular we need to secure our system by controlling the subjects, objects and actions allowed on our system. Achieving this goal requires a good combination of Authentication, Authorization, and Auditing. The components are sometime referred to as ***the AAA of security*** [cite] or the three-headed guard dog of the security world.

Authentication is the process of identifying the subject. Authorization defines a relation among subjects, actions, and objects. Accounting is the act of monitoring or auditing the actions of subjects on objects. Together these three components allow us to control and trace every activity that occurs in our system. This view of security can be summed up as "Who can do what when?" Authentication allows us to identify "Who". Authorization controls the action ("do what when"), and Auditing records all the action. All three components are necessary. Missing one component is likely to result in an insecure system. Compromising one component is likely to compromise the whole system. To illustrate these concepts, we will provide a few examples.

Example 1: A safe.
The policy of the safe is that the person with the key is allowed to access the safe. The authentication is the validation of the key. In this case, the mechanism for validating the key is the lock. Accounting could be a log of people who inserted a key into the lock.

Example 2: Automatic Teller Machine.
An automatic teller machine (ATM) is a computerized system (device) that provides a secure method of performing financial transactions without the need for a bank teller. It is a perfect example of well-constructed secure system. The system uses a combination of token (ATM card) and password (Personal Identification Number---PIN) for authentication. Customers are authorized to perform certain transactions with respect to their financial status. In addition to the video camera equipped with the machine for monitoring the user, each transaction is also recorded in a log file.

Example 3: Encryption.

Encryption is a useful tool that can provide both authentication and authorization (in certain applications). The cipher key together with the encryption algorithm forms the authentication mechanism; the policy implicit in every encryption is that only an authenticated subject can access the encrypted object.

It is worth nothing that a subject is not limited to a person. Indeed, subjects can vary in both time and space. For example, a subject may be a packet in network communication. One may argue that there always is a person behind the machine that generates the packet. However, in such a case, we do not control the person, but the packet.

This chapter is organized around these three A's of security. Throughout the chapter, readers will be exposed to a wide variety of authentication methods and their limitations. In the end, readers should be able to select an appropriate authentication method for his or her application. With respect to authorization, the material in this chapter focuses on the concept. This is due to the fact that authorization varies so much from application to application. However, these concepts should serve as a framework for creating strong policies specific to a particular application. Finally, we illustrate the importance of reliable accounting systems which can provide invaluable feedback for solving several security issues.

2.2 Authentication

"It's easy to know men's faces, not their hearts."
Chinese Proverb

In a computer system, authentication is the process of verifying identity of a user. In a communication system, authentication is the process of verifying the stated source of a message [dictionary.com].

The definitions of authentication may vary in different contexts, they all, however, share similarity of validating the quality or condition of being trustworthy, genuine, or creditable.

Intuitively, validating a subject would require examination of a token or investigation of some property of the subject itself. For example, validating the authenticity of a student's transcript can be done by validating the university seal on the document itself. In this example, the token is the university seal. Similarly, the identity of a person can be validated by examining the person himself (or herself) or validating some identifying token carried by the person. For example, asking a question that only Alice can answer (e.g. prearranged question) can be use to authenticate Alice. This concept is not new, but has existed for (thousands of) years. A signal used by pilots during the Vietnam War[1] best epitomized this.

Implicitly, we have to trust a token or a property of the subject otherwise the validation process is impossible. The obvious drawback of basing the authentication on a token is the integrity of the token. For example, we may need to trust that the name of a person printed on a driving license is correct even though forgeries are a possibility. Nonetheless, trusting is necessary to the authentication. Therefore, an improvement is to base the authentication process on multiple tokens or properties. In addition, we also have to validate the integrity of the tokens.

If authentication relies on tokens or properties of the subject, studying protocols and methods for validating properties or token is necessary. In fact, they are the heart of this section. In addition to the legacy token, the material will also cover the concept of ***smart tokens*** (e.g. smart card). In general, the smart tokens are tokens that can authenticate themselves. For example, a smart card consists of an encrypted key and algorithm. We can validate the smart card by a simple challenge and response protocol.

The protocols and methods discussed in this section primarily focus on computer-related applications. However, the same concepts should be applicable to every security related system in general.

This section starts with the tokens, properties and the methods of using them. Later, we will focus on more complex protocols where a combination of tokens, methods and interactive protocol are used to provide a stronger authentication system. The end of this section is dedicated to case studies, where we will guide readers through real-world authentication systems.

---

[1] The July 2004 issue of Air Force Magazine Online featuring a story, "The Courage of Lance Sijan", by Correll showed that pilots used several prearranged questions for authenticating Sijan. In this article, one of the questions chosen by Sijan was "Who is the greatest football team in the world?", and the answer was "The Geen Bay Packers." (Find full story at http://www.afa.org/magazine/july2004/0704sijan.asp) .

2.2.1 Methods

So far we have learned that tokens and properties are critical for validating the authenticity of a subject. This infers that a key to strong authentication is to choose the appropriate tokens or properties for that application. Please note that every authentication method has its own strength and weakness, and there is no such thing as a perfect authentication method.

Rather than focus on a token or a property in particular, we divide the methods into three main categories, and analyze each category at a high level. For each category, we will learn the concepts and its limitations. Those three categories are:

- What do you ***know***?
- What do you ***have***?
- Who do you ***trust***?

In general, we can authenticate a person by validating something that he or she knows, checking a token that he or she has, or trust that a person is whom he or she claims to be. In practice, the trust concept also includes trust of a third party. For example, we may believe that the person is Alice if she is endorsed by Bob.

Another aspect that should be taken into account when choosing an authentication method is the transferability. While some tokens can easily be handed to another person, some cannot. There are two sides to transferability: it is both desirable and undesirable.. For example, a person can ask someone to do his or her job when he or she is incapable of doing it. However, transferability also means that the token can be used if stolen. A good analogy is a key to a house. Using the physical key as a token, we can easily allow someone to get into our house by simply passing the key to them. The drawback is that we may accidentally lose our key and have it fall into the wrong hands. In contrast, imagine that a door to an house is operated only by the owner's fingerprint, it is impossible to ask a (trusted) friend to tend to your house when you are away. However, there is no need to managing and tracking the token. For example, it is possible to allow a friend for getting into a house, but there is no guarantee that he will not duplicate the key for later (malicious) use.

2.2.2 What do you know?

A secret between two is God's secret, a secret between three is everybody's.
Spanish Proverb

Perhaps, the simplest way to verify that a person is whom they claim to be is to ask a question that only the system and the person know the answer. The beauty of this method is that there is no need for a physical token so it is easy to implement. The most common example of this method is the ***password*** or ***passphase***; however, the idea is not limited to that. The more complex version is known as a form of the challenge and response protocol where the correct password depends on the context of the communication.

One concern is how long your secret will be safe. To raise the awareness of this concern, we will provide a guideline for selecting a good password. This idea will also serve as a fundamental concept in managing the key in encryption.

2.2.2.1 Password or secret

Password (n)
1: something that enables one to pass or gain admission: as
a : a spoken word or phrase required to pass by a guard
b : a sequence of characters required for access to a computer system
Merriam-Webster Online Dictionary

Passwords are known to be useful and are a widely-used authentication method. Rather than spending time defining the password, this section is dedicated to the measurement of the quality of a password. The quality of password can be measured in several aspects. Given enough resources (e.g. computational power), it is just a matter of time before the password will be broken. From this perspective, the longer the time it remains secret, the more secure that password is. Economically, the security of a password can be reasoned as the cost taken to break it. However, there are certain properties that make one password system more secure than the other.

Uniqueness
A unique password should be difficult to guess. One known method for attacking a password protected system is the dictionary attack. In this method, the attacker will use a list of words such as a dictionary as a source of guesses. Hence, one property of the good password is the uniqueness—if it can be guessed from a small list of words, one can say that it is not sufficiently unique. Therefore, from the point of view of uniqueness is advisable that a good password should be a combination of numeric, alphabetic and special characters (e.g. punctuation). This combination will enhance the uniqueness of the password requiring attacks to use "brute force."

Length of password
A longer password should be harder to guess using "brute force" increasing the chance that it will be more secure. Probabilistically, a long password would require more combinations of guesses. However, the length of a password is limited by the ability of a human to remember it. Long passwords can end up encouraging users to write down their passwords in order to remember them, which can reduce its security. It has been suggested that password should be at least 6 characters, but that may be too small as computational power has increased. Historically, UNIX systems use 8-character passwords. On average, one study indicated that a person is capable of remember up to 12-14 numbers, but it may be shorter for random characters. We will quantitatively analyze the details in next section.

Age
To strengthen its security, a password should be changed regularly. In fact, if it is expected that a password can be broken (possibly by brute force) in some time period, a system should force a user to change the password within that time frame. The benefits of aging are: (a) it forces attackers to work within a limited time frame—limiting damage; (b) it allows us to keep track of the active users.

Password History
Most of the time people will reuse their password for convenience. This convenience, however, assists attackers in that once a password is circumvented; it is expected to be circumvented again in the future. With password history, a system can prevent the user from reusing recent passwords. Consider a situation where 10 passwords previously using by a

user for accessing a system were differences, attackers would have a hard time guessing the current password (since there is no hint for the current password).

Invalid attempts

One method to defend against a brute force attack is to check the number of recent invalid attempts. For example, if a particular user enters a wrong password several times in a row, it is likely that this account is being attacked by brute force. If we only allow a few attempts of invalid password and temporarily disable the target account, attackers will not be able to use brute force to guess a password (through this interface). However, the drawback of this method is that it may deny a legal user from accessing the system after their account was attacked. This undeireable situation can be partially solved by resetting the number of invalid attempts within a particular time frame (e.g. every midnight). This way, attackers are limited to only a few trials in a period, which limits a brute-force attack.

Time between attempts

There are two concepts related to time between attempts. First, time taken for processing valid account and invalid account should be comparable. Second, sleeping time before users are allowed to try again after an invalid attempt.

The first idea is to prevent attackers from guessing a valid login name. To ease understanding, consider the following programs in Figure 1. Algorithmically, three programs yield the same result. However, the execution times are fundamentally different. Among the three programs, Prog 1 is the only program that assists finding a valid login name. By simply entering a login name and checking weather the password is prompted, it is trivial to identify a valid login name. Though Prog 2 does not explicitly assist the guessing of login name by skipping the password prompt, timing the process still unveils such information. From the security point of view, Prog 3 is better in that it does not provide any useful feedback to attackers.

Prog 1.	***Prog 2.***	***Prog 3.***

1. Input [login name]
2. Fetch [saved password]
3. If no entry then
  exit
4. Input [password]
5. Compare passwords.
6. If valid then
  start session
  else
  exit
  End if
	1. Input [login name]
	2. Input [password]
	3. Fetch [saved password]
	4. If no entry then
	  exit
	5. Compare passwords.
	6. If valid then
	  start session
	  else
	  exit
	  End if
		1. Input [login name]
		2. Input [password]
		3. Fetch [saved password]
		4. If no entry then
		  [saved password]←Null
		5. Compare passwords.
		6. If valid then
		  start session
		  else
		  exit
		  End if

Figure 1 Examples of login programs

Another idea is to limit the number of attempts in a period of time by sleeping between each attempt. Though the method does not make a system more secure, attackers are forced to take longer physical time.

One-time password

Considering the conditions listed above indicates that a good password should be changed regularly and valid only in a limited time frame. An extreme case is the one-time password—a password that can only be used once. After successfully logging in to a system, a new password is then assigned. This method guarantees the uniqueness and aging of the password. However, users may find it difficult to manage (remember) their passwords.

2.2.2.1.1 How secure is a password?

Given that a password is unique enough so that it cannot be found in any dictionary, the only possible vector of attack would be randomly guessing or brute force. In this section, we will analyze the quality of a password system. Indirectly, this analyzes quantitatively supports the properties of password system mentioned previously.

Assume that:

- ***n*** is the length of the password (e.g. digits or characters).
- ***k*** is the number of characters in the set of possible characters.
- ***C*** is the constant amount of time requires for testing a password (e.g. seconds).
- ***t*** is the number of times allowed to guess the password before locking the account.

Given *n* characters in a password, each character is taken from the *k* characters in the set,
The total combinations of possible passwords are k^n.

The estimated time for testing all possibilities (brute force) is Ck^n. Algorithmically, the complexity of this function is $O(k^n)$. Thus, time taken for breaking the password with brute force ranges between ***C*** (best case) and Ck^n (worst case).

One way of constructing a secure password is to maximize the worst case. To do so, we may choose to maximize one or both variables: ***k*** and ***n***. Given this condition, increasing ***k*** and ***n*** can be done by increasing the number of possible letters used in the password and the length of the password respectively. Table 1 shows the number of possible letters in each case. Note that ***C*** is a constant value on a system, but varying from system to system.

Types of characters (English)	**Number (*k*)**
Lower-case alphabetic	26
Numeric and lower-case alphabetic	36
Upper and lower-case alphabetic	52
Numeric, lower-case alphabetic, with symbols and punctuation	68
All displayable characters	94

Table 1 Number of possible letters

From the attacker's point of view, constant value ***C*** is perhaps an only parameter that can be lessen. Since variables ***k*** and ***n*** are specified by the system, attackers are indirectly forced to try the large combinations. However, attackers may choose to parallelize the attacking processes by running it simultaneously from a cluster of computers. Parallel processing allows the attacker to speed up the system, which obliquely reduces constant time ***C***. For example, 50 nodes cluster system may reduce the worst case from Ck^n seconds to $\frac{Ck^n}{50}$ seconds in theory. If the cost for constructing a cluster is $ XXX, it is suggested that this password system can only protect an asset whose value is less than $ XXX.

Maybe a picture of cluster machine used for password hacking or a real story.

A naïve solution to counter the advance of brute force method is limiting the number of times for invalid entries (***t***). This, however, only applies to active (online) attack. To establish that

this limitation can reduce the probability of successful attack, we first formalize the probability of successfully guessing the password. Assuming that a correction combination of password is a red ball in a box with k^n balls, the probability of picking a red ball from this box is $\frac{1}{k^n}$. If one can pick a ball indefinitely, this probability is then $\frac{\infty}{k^n}$ or 1. Given the time function earlier, it is just a matter of time before the attacker gets the correct combination. If one can pick only t balls, the probability of guessing the password is $\frac{t}{k^n}$.

Maybe a picture of balls explains such probability.

As described, the more the attacker can guess, the higher the value of t which increases the probability of success. However, the combination of password aging, password history, and the number of invalid attempts directly limits the value of this variable.

2.2.2.1.2 Guidelines

"Treat your password like your toothbrush.
Don't let anybody else use it,
and get a new one every six months."
Clifford Stoll

There are several guidelines and suggestions on how to handle passwords. The noticeable one is ISO17799 which states password quality guidelines. We will briefly describe this standard as follows.

Overview of ISO17799 password quality guidelines [CITE]
- Passwords should be at least six characters long
- Passwords should be free of consecutive identical characters
- Do not use all numbers or all letters
- Avoid reusing or recycling old passwords
- Require that passwords be changed at regular intervals
- Force users to change temporary passwords at the next log-on
- Maintain a record of previous user passwords and prevent their reuse
- Change all vendor default passwords
- Eliminate or lock shared-user account

Mnemonic Phrase Password

In addition to the ISO17799 guidelines, Anderson et al. (http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/tr500.pdf) performed an empirical study that showed that passwords based on mnemonic phrases are just as hard to crack as random passwords yet just as easy to remember as naive user selections.

Their mnemonic phrase for passwords technique is as follows:
1. Choose a phrase which is familiar to you: "My favorite car is a pearl-blue Porsche 911 Turbo"
2. Extract the first characters from each word: Mfciap-bP911T

The resulting password contains a set of characters which is nearly impossible to guess, yet is unforgettable as long as you remember the phrase. One should choose a phrase which results

in a password which adheres to the ISO standard. In this case, the password is 13-characters long and is made of upper-case and lower-case characters, digits, and punctuation. Of course, if you actually own a pearl-blue Porsche 911, that phrase might be guessable.

Substitution
If your password has insufficient numbers, you can substitute numbers for some letters. One could choose to substitute similar representations: e.g. replace the letter '***e***' with the number '***3***' because one can view the number '***3***' as the letter '***E***' written backward. Under substitution the word "***pencil***" becomes "***P3nc1l***". Alternatively, the substitution can be based on the sound of that word (a.k.a. soundex). For example, if the password is "***For you, and me too***", the substitution version may be "***4U&m32***". We can also combine techniques so the statement "***You are not alone, I'll be with you***" may be transformed to "***UR~1lb3wU***". (The character '~' sometimes means ***"not"*** in mathematical logic.)

Avoid Patterns
While the use of mnemonics and substitution are encouraged, patterns should be avoided. Examples of patterns are time sequences and keyboard patterns (including backward spelling). In some cases, users may be forced to change their password. A common pitfall is appending a sequential number to the password, leaving the remainder unchanged: e.g. "***password1***", "***password2***". Another example of a guessable pattern uses letters as they are laid out on the keyboard (e.g. "***qwerty***", "***q1w2e3r4t5y***").

Typing
Last but not least, a password should be able to be typed relatively quickly so someone nearby cannot determine the password. This issue may depend on the typing skill of each user. However, a common technique is to obtain a password by simply looking over the user's shoulder, a.k.a. "shoulder surfing."

2.2.2.2 Storing a password

If a password is difficult to remember, a user my write it down. However, writing a password in plain text makes it available to anyone who gains physical access to the writing. If it is necessary to write a password down, encode it in a way that nobody can understand it. Also, if it must be written down, some suggest that one's wallet may be more secure than leaving the password where one uses it. If someone gains access to your workplace, and your password is written there, you are compromised. However, if the password is with you, you are safe from that attack.

How does a computer store your password? Modern computer systems use a one-way hash function to store the password in an encrypted form. (We will describe the details of one-way hash function later when we get to encryption.) This technique prevents a direct attack on the password file while allowing one to check the password. The check is performed by passing the user-entered password through the same one-way hash function and then comparing the hashed values. The password is secure because a well-formed one-way hash function does not allow one to determine the preimage of the hash, i.e. one cannot look at the hashed (encrypted) password and determine the password.

2.2.2.3 Challenge and Response

This method is particularly useful when subject is not a person. For example, machine may use this method to authenticate the token. The idea is that rather sending plain password, the client have to response with an encrypted version of server challenged word. In most case, this challenged word is simple a random number (N). However, a shared key (k) is necessary. Challenging between Alice (A) and Bob (B), the protocol may look like:

$A \rightarrow B : N$
$B \rightarrow A: \{N, B\}_k$
Since each message is unique, this scheme can indirectly prevent replay attacks. The strength of this method however relies on the predictability of the random sequences. In real application, the randomness is sometime deterministic. Hence it is possible to guess the next sequence by listening to the previous message.

2.2.3 What do you have? (tokens)

Tokens come in variety of forms. They can be simple paper cards, smart tokens with processing power, or even a part of human body. Though they are different, they all serve the same function "identification". The idea of using tokens has been with us for more than 4,000 years. At least, we know that every king (and institute) has a unique seal for validating the authenticity of his royal documents.

2.2.3.1 ID

This is one form of authentication used widely in our daily life. ID comes in the form of formal document issued by trustworthy institutes such as school, government, company. A good example is state ID or passport. To vote, a state ID is required for identification. A drawback of this method is that an ID can be duplicated. Thus, a more secure type of ID may include seals or microchips for validating the ID itself.

2.2.3.2 Seal

Seal is an old but trustworthy form for validating the document. It can also be used for validating the genuine of the token. For example, a student document is not valid without a university seal. Another form of seal is watermark where a figure is embedded in the background of a document.

In some country, a person seal is required for a legal document. A good example is Japanese HANKO stamp [http://www.boingboing.net/2005/06/02/japanese_hanko_stamp.html]. Once a HANKO stamp is registered with a government (called JI-TSU-IN), any document stamped with it is a legal document regardless of who stamped it. Figure 2 is an example of the HANKO. Similarly, some countries require that a person stamp his/her thumb as a signature in every legal documents.

Figure 2 Example of Japanese HANKO

However, the seal can easily be broken. In Japan, where HANKO stamp must legally be registered, there exits several cases that a thief HANKO stamp withdrew money from an account using a copy. Hence a good seal is one that is difficult to make a copy. For example, Singapore bank uses hologram seal to validate the banknote.

2.2.3.3 Smart Tokens

Smart cards are tokens with processing power. In several cases, it is capable of storing and computing a complex encryption algorithm. This is usually used to provide a credential for tokens. In fact, it introduces difficulty to reproduce a token.

2.2.3.4 Biometrics

Biometrics are automated methods of recognizing a person based on a physiological or behavioral characteristic. The features measured are:

- Fingerprints
- Hand/Palm geometry
- Handwriting
- Face Recognition
- Dental biometrics
- Retinal
- Vein
- Voice
- Pattern (walking/typing rhythms)

This often requires a special hardware for measuring the characteristics. Though a person is unique, there are similarities (e.g. between father and son, twins, brothers). In fact, a person also changes in time. Thus, it is advisable that a single biometric might not be sufficient for authentication.

2.2.4 Trust

Another form of authentication is trust. Assuming that a person who has access to a room is a trustworthy person, an authentication at the console might be irreverent. However, the idea of trust also includes location, trusted third party and time.

2.2.4.1 Third party authentication

In several cases, authenticating an unknown person can be difficult. An intuitive solution is to ask a third party agent to authenticate and endorse this person. For example, a company A may not have any information of users in company B. Sharing information between the companies should require authentication from each company. To be specific, users from company B should be able to log on to a system in company A with the same password for logging on to a system in his/her company. (Given that you trust the system of company B to authenticate the right person.)

The scenario is similar to the security at the entrance of a building. At the gate, visitors are required to show their ID and sign their name. In return, the visitors will get a badge While the badge is used for identification, it also shows that a particular person has been authenticated at the gate. In the building, everybody trusts the badge as an identity of a user. In this example, we trust the security system at the entrance to authenticate a person.

This form of authentication is often used in distributed systems and applications. Another type is the third party certificate when a key is generated and transferred by a trusted party.

2.2.4.2 Proximity/Trusted Zone

Imagine that a machine is able to identify a user sitting at the console and logout when the user is away from the console; authentication method is automatically enforced here. This is useful when physical access is necessary. One good application is medical record. In medical industry, a doctor always has to walk all over the place. Once the doctor log in for accessing the medical record, the system should be able to prevent another user from accessing while the doctor is not in the office.

Detecting proximity of a person can be done in several ways. In several cases, a special token is used as a tracking device. A simple one is RFID. An RFID on the watch or key card allows a system to detect the proximity of a person. In some countries (e.g. Japan), a cell phone is used for proximity detection and authentication. We often see this kind of authentication at the entrance of a secure office where a card is used to open the door.

(see http://www.ensuretech.com/products/technology/technology.html for more details)

2.2.5 Interactive Protocol

2.2.5.1 Password + Token

We know that human is incapable of remembering a long and Secure Bit password. Thus the ideal of a password generator token is introduced. With a complex password generator, a user will use the simple password for accessing the generator. The password generator then generates a one-time password for logging on to the machine. This way, it is theoretically impossible to capture a password for replay attack. This password generator can be a piece of software installed on a user machine, or a dedicated embedded hardware token (e.g. crypto card).

The critical concept of this one-time password scheme is the password generator and the algorithm. Both the server and the generator share the same algorithm for generating the same sequence of password. If the seed and the algorithm can be identified, it is possible to duplicate the generator.

Figure 3 Crypto card (http://www.cryptocard.co.za/images/tokens.jpg)

2.2.5.2 Zero Knowledge Password Proof

While traditional password protected system would require a subject to unveil the password to the system, there exists an idea of proofing the knowledge with you leaking it. This concept is known as "Zero Knowledge Proof" [CITE]. To ease explaining the idea, we will give a sample situation using pieces of paper (the idea is taken from [CITE]).

Suppose that Alice (A) want to proof that Bob (B) knows the same number (assuming that the number is between 1 and 10). Let assume that Alice picks 3 and Bob picks 7. Given same ten pieces of paper (card) putting in a deck, here are the steps for proofing.

- Alice marks card number 3 on one end and passes the hold deck to Bob.
- Bob marks card number 7 on the other end (without seeing what Alice has marked on the other end).
- We may shuffle the deck and lay them all on the table.
- If there exits a card with marks on both end, we know that Alice and Bob choose the same number.
- In case there are two marks on different cards, we know that they do not pick the same number.

We may apply the same concept for proofing the password. Here is the possible scenario.

Assume that password consists of four numbers; and each number is between 1 and 9. Both machine and user knows the password. The machine would generate 4 random strings of size 10 and pick the corresponding character of a digit of password from each string and remember it. Then these strings are passed to user for challenging. The user will also select the corresponding characters, shuffle it and send it back to the server. If any combination of the password is matches, the password is then authenticated.

Assume that password is {1, 2, 5, 7}.
Server creates four strings and sends them to user:

```
      1 2 3 4 5 6 7 8 9
S[1]: T H I S I S 1 S T
S[2]: R A N D O M 2 N D
S[3]: E X A M P L E 3 1
S[4]: 4 T H N U M B E R
Password: [BATP]
```

In this case, the password would be encoded as any combination of "TAPB" (e.g. "PATB" or "BATP"). This way, server can authenticate the password without directly sending the password.

This scheme requires shared information to be transmitted first. If attackers can see both shared strings and input, it is still statistically possible to discover the password. This idea is served as a fundamental for several encryption algorithms in that at least a shared key is sufficient for protecting the integrity of data.

2.2.6 Implementation Issues

Up to this point, we have learned several authentication methods. In this section, we will continue to focus on the implementation. There are several issues in creating an authentication system. Among the critical issues are:

- Management Cost
- Communication Channel
- Human Factor
- Accuracy
- Transferability

(Elaborate the idea)

In the modern day of distributed application and system, the idea of network authentication has emerged. The concept is to use a single identity for a group of system. We will focus on two related ideas: Centralize authentication and Single Sign-on.

2.2.6.1 Centralize vs. Distributed

There are two main ideas in this section. First is the authentication method for several machines (systems) that use a central account. For example, a user may log in to any workstation in the network using the same account and password. Second is the method for managing the distributed authentication. For example, authenticating users from different branches of an enterprise company may require an assist from branches' main server.

The two ideas are not mandatory and always mix together. In several cases, an authentication protocol might not be able to handle both cases efficiently. Later in this chapter, the case study will show that some protocols are designed for several machines, but require a central account agent. Similarly, some protocols distribute the authentication agent and have difficult managing the roaming account (e.g. proxy account).

2.2.6.2 Single Sign-on

With respect to the distributed applications (e.g. web application), some processes involve several agents. Rather than entering a password whenever a process is passing to an agent for the first time, it is preferable to users for using a single sign on for all agents. At least, the scheme provides user-friendly experience for the application.

2.2.7 Conclusion

Authentication is the first step of security. There exist several authentication methods. Each method has different criteria. Through the section, we have provided a guideline to a wide variety of authentication protocols. Readers are expected to understand the pros and cons of each method. It is worth to emphasize that there is such completely secure system. Every authentication method has a cost associated with it and can be circumvented. At minimal, human is one of the weakest links. Designing an authentication system should take these factors into account.

2.2.8 Case Study:

Move all cases to appendix.

Authentication methods and technologies have been evolved for several decades. To appreciate a wide variety, we will guide you to the contemporary authentication protocols using in the modern applications.

2.2.8.1 Unix, NIS (SUN YP)

2.2.8.2 Kerberos

Key management, Limitation

2.2.8.3 RADIUS

2.2.8.4 Microsoft PASSPORT

Centralize, trust

2.2.8.5 LDAP

2.2.9 Exercises

1. Is an 8-character UNIX password secure?
2. Find 2-3 tokens that can be used to identify your identify, and analyze the integrity of those tokens.
3. An automatic teller machine (ATM) uses ATM card and 4 digits number for authentication. Do you think this method provide sufficient protection? Please justify your answer.
4. An attacker is able to sends a malicious network packet to a database server behind the firewall by spoofing the source address of the packet. Do you agree (or disagree) that this is the failure in the authentication protocol. Please justify your argument.
5. Supposedly, you were a computer engineer in charge of designing a new mechanism for used as a key to a new model of vehicle. What kind of token or method will you choose? And Why?
6. You are asked to design an authentication system for an online stock trading web site. Please outline the protocol or method that you will use and analysis the potential threat.
7. In an attempt to avoid remembering several passwords, a web browser provide a mechanism for storing password of each web site and allow the user to secure it with a master password. Do you think this is a good idea?
8. Given that a password is not unique and can be found in a dictionary. Please evaluate the probability of successful attack (assuming both forward and backward spelling).
9. What is the market value of a botnet?

2.3 Authorization

Honesty is the best policy
Italian Proverb

While Authentication is critical for identifying a person, Authorization is critical for controlling people and resources in a system based on their identity. Since authorization is tightly related to the system, it is strictly depending on the application. This section provides a framework for implementing the authorization process of the system.

2.3.1 Introduction

A secure system should also protect resources by only allowing those resources to be used by those with grant authority. Restricting access based on consumers and resources is generally referred to as *Access Control*.

Enforcing access control is theoretically straightforward (e.g. a few lines of code are sufficient). Once a user is authenticated, a mechanism is needed to control the resources and functions that this user can access. Since systems vary in resources and users, a general approach for authorization is necessary. To generalize the idea, we divide the process into two parts: What to control (Policy) and How to control (Type Enforcement).

Policy is a given plan of action to guide and determine decisions. It a generic document that outlines rules for accessing information. In another word, policy is how human specify the authorization. It is highly derived from modeling a threat to the system (Threat Model). The eventual translation is the type enforcement.

Type Enforcement is a tool for enforcing the policy. It is the low-level interface to the system. A system would request for permission from the type enforcement for managing users and resources.

Figure 4 Policy and Type Enforcement

While policy is specifically tighten to an application, Type Enforcement is more general in a system. Hence, one critical task of authorization is the translation from policy to type enforcement. Currently, there is no explicit method for such transition. Though such translation is naïve, several cases show the loss of information.

2.3.2 Policy

Policy is a human language for stating the access control matrix. “It is a definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions” – [Merriam-Webster online Dictionary]. Though

policy can be stated in various methods, the main idea is to partition the states of the system into authorized and unauthorized. To generalize the concept, we define the security policy.

Definition:
A security policy is a statement that partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized or insecure, states. – [BISHOP].

Intuitively from the definition, we can say that "the art of securing a system is the process of preventing a system from entering an unauthorized state" – hence, the definition of secure system.

Definition:
A secure system is a system that starts in an authorized state and cannot enter an unauthorized state. – [BISHOP]

State Transition diagram goes here.

Authorized states and unauthorized states vary. However, they are commonly partitioned using two properties of data[2]: *confidentiality* and *integrity*. A state that can provide confidentiality and integrity of data is considered to be an authorized state (secure). If one or both property cannot be preserved, such state is unauthorized state (insecure). Before getting further, it is worth clarifying confidentiality and integrity.

Confidentiality is the obligation to confine or protect data from being access by unauthorized person. In another word, only the right person can access the data. (Who can read the data?)

Integrity is the condition of being unimpaired. In this context, it simple means that data is not being altered by unauthorized user. (Who can alter the data?)

Based on these definitions, there are models proposed for preserving confidentiality and integrity. We will describe these models as access control models. The concept of access control models is to define rules for user accesses to resources—hence a guideline for type enforcement. Keep in mind that each model is proposed to handle different security issue. Thus, a particular model may focus on a particular property (e.g. toward integrity). It is also worth clarifying that confidentiality and Integrity are sometime viewed as dual concepts in that confidentiality views security as who can read while integrity views as who can write.

2.3.2.1 Access Control Models

There exist at least three access control models. We will state the underlying principle and briefly describe each model.

Mandatory Access Control (MAC)
Principle: Users are untrustworthy and must be controlled.

The most important feature is that the user cannot fully control the access to resources that they create. The system security policy (as set by the administrator) entirely determines the

[2] Researchers sometime use different terminology to refer to data that a system wants to protect in different contexts. These terms include (but not limited to): sensitive information, secrecy, and privacy.

access that is to be granted and a user is not permitted to grant less restrictive access to their resources than the administrator specifies.

Discretionary Access Control (DAC)
Principle: Users are responsible for their own data.

Discretionary access control systems permit users to entirely determine the access granted to their resources, which means that they can through accident or malice give access to unauthorized users. Thus, DAC is frequently referred as a model toward integrity. Generally, these are done at the *discretion* of the object owner -- file/directory permissions and user/group ownership.

Role-Based Access Control (RBAC)
Principle: least privilege.

Role-Based Access Control (RBAC) is another approach to restricting system access to authorized users. It is a newer and alternative approach to Mandatory Access Control (MAC) and Discretionary Access Control (DAC). RBAC uses the security principle of *least privilege.* The idea is to grant only resources and information that are immediately necessary to performing a particular role. The eventual result is the enhancement of protection against malicious behaviors. This model works well for corporations with a large turnover of personnel.

2.3.2.2 Environment & Policy

Another emerging concept is that policy should be able to adapted to environment. A good example is that an employee should be able to access a customer's account only from his desk during office hour. (Maybe Dr. Enbody can provide more description here.)

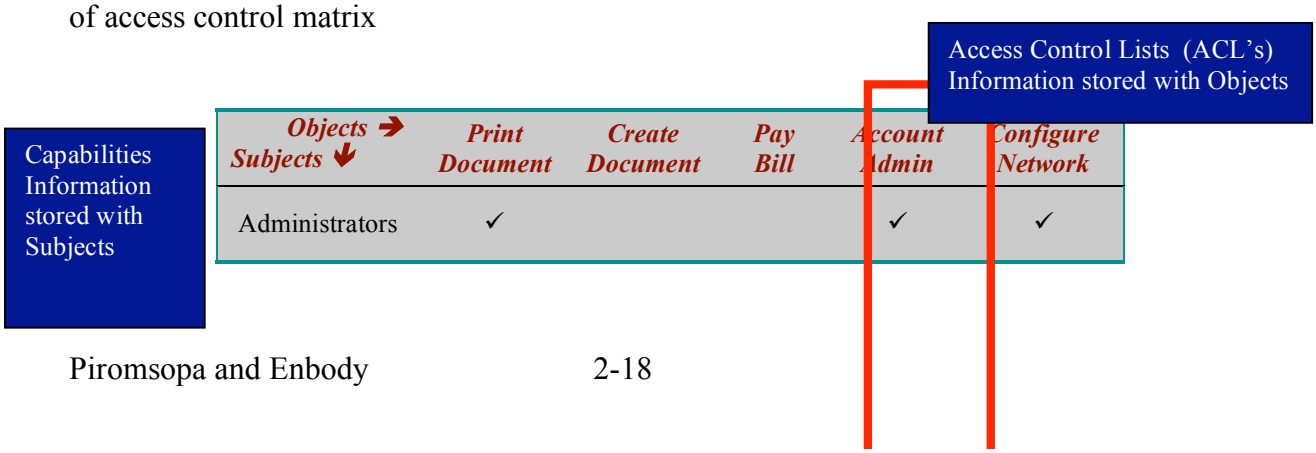
2.3.3 Type Enforcement

Type enforcement is a table-oriented access control mechanism well suited for confining applications and restricting information flows. It is an interface to users and applications. Although it is both flexible and strong, type enforcement alone imposes significant administrative costs and has not been widely adopted early.

There exist three well-known methods: Access Control Matrix, Access Control List, and Capabilities. Each method shares similarity in associating subjects, operations, and objects. However, they differ in management scheme and provide different granularity.

2.3.3.1 Access Control Matrix

Access control matrix is a list of the users, groups and roles down the left-hand side, and all the resources and functions across the top. The matrix represents rules. Figure 5is an example of access control matrix



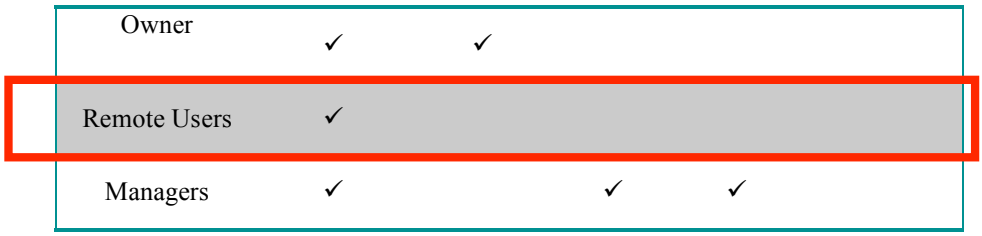


Figure 5 Access control matrix

Designing an access control can be complicated. However, if the rules can be expressed in this matrix, it is unlikely for the implementation to be correct. To appreciate the concept, we give a guideline for designing the matrix.

- Label subjects
- Label resources and functions
- Specify the rules

However, labeling is just an abstract concept. In real application, subjects may not be a person, but characteristics. For example, a label may be an active customer (accessing the system more than one times within a week). Depending on the nature of an application, one may want to label subjects based on age, gender, and country of origin or religious. This is applicable to web application where a certain group of customer should not be exposed to particular information (e.g. Male customer should not be exposed to feminine product.)

Similarly, labeling resources and functions require some metadata for binding. So, the system should incorporate this information in the design when creating resources and functions.

2.3.3.2 Access Control List

Access control list is a management scheme that stores the access control matrix a column at a time. It is widely used in an environment that a user is responsible for managing the security of her/her resources such as Unix (and NT). The obvious advantage is that access control list is simple to implement. Nonetheless, it is not efficient in checking the permission at runtime (e.g. list all files that are readable to a particular user).

Though it is data oriented, access control list is not suitable for a system with a large population of users and constantly changed. In several cases, users want to hand over their authority for running a program to another user (e.g. managing password file). This cannot be handled efficiently with Access Control List. In fact, it is tedious to revoke all the files to which a user has access. Similarly, it is time consuming to list files with world readable.

2.3.3.3 Capabilities

Another method for storing the access control matrix is to associate it to the subject. These are called the capabilities. The benefit and drawback of capabilities is merely the opposite of Access Control List. With capabilities, it is easy to list a capability of a user or even hand over an authority from one to another. For example, I have a full control of this file and I want to allow you to read it. To do so, I simply extract a part of my capability. However, assigning a group of users with a capability to read project's files would require a list of users and a list of files and result in a few numbers of capabilities.

To simplify this, it might be easier to implement capabilities as a token. The idea is that an object does not care who accesses it as long as a valid token is provided. This way giving a group of user a permission to access a list files can be done by simply handing them a copy of

valid token. However, an issue of the authentic of a key is raised (make a key difficult to forge). As a result, a capability token is often crypto.

2.3.4 Security Models

Given variations of access control models and type enforcement, adopting them for using in a system is, sometime, not obviously straightforward. In this section, we will present a high-level concept, security models, to ease understanding and specifying policy. A security model is a generic framework for dictating a policy system. As a result, it is arguably easier to layout a policy by first outlining a security model. Security model is sometime referred as threat model in that it also describes a potential problem from administrator's point of view.

There are two general framework of security: Multilevel security and Multilateral Security. While multilevel security is primarily used to handle security within an organization, multilateral Security is used to handle between organizations. Though the concepts are related, they are fundamentally different. In fact, we sometime have to due with both multilevel and multilateral at the same time (which makes models more complicated).

2.3.4.1 Multilevel Security

Multilevel Security is a model that dues with security within an organization. The fundamental idea is clearance and label. In particular, every subjects and objects have to be labeled. Intuitively, we can say that mandatory access control (MAC) is a requirement for multilevel security.

(Maybe picture of levels/models goes here)

There are at least two models proposed: Bell Lapadula and Biba.

Bell-LaPadula
Principle: The system may leak the information.

David Bell and Len LaPadula [CITE] proposed this model in 1973. It is used in the classification of military and intelligence data. The underlying concept is driven by the fear of attacks using malicious code. The basic properties of Bell-LaPadula model are:

- The simple property: no process may read data at a higher level. This is also known as *no read up* (NRU).
- The *-property: no process may write data to a lower level. This is also known as *no write down* (NWD).

We can summarize Bell-LaPadula model as "No reads up. No writes down." The model generally assumed that bad software (e.g. buggy or vulnerable program) tends to find a way into the system. Thus, allowing software to pass data to or signal a lower level software may leak information. This property preserves confidentiality. Nonetheless, the model does not directly describe the creation (or destruction) of subjects and objects, which are the most difficult issue.

Another variation of Bell-LaPadula model introduces *tranquility property*. A strong *tranquility property* states that the security level of an object cannot be changed while it is being processed by a computer system. With the strong version, users may ask administrator

to temporarily declassify a file from high to low, and eventually read up with out breaking the model. Alternatively, the weak version allows security labels (level) to be changed in a way that does not violate a defined security policy. The idea is to start a process with least privileges and upgrade whenever it accessing higher file. This principle is sometime referred as high water mark. Given the *-property, a program that suddenly upgraded to higher level then cannot access its file that is created early when it is in a lower level—hence the complexity of applications.

Biba
Principle: Information flowing from lower level may be malicious to the system.

Ken Biba [CITE] proposed this model in 1975. Its main focus is integrity and sometime referred as Bell-LaPadula upside down. Hence we can easily summarize the model as "Never read down or write up." The idea is that a low-level process must not alter a high-level data. For example, a process may access the kernel data, but may not modify it. On the other hand, kernel can freely modify users data. This principle is referred as low water mark.

This model matches with several applications in real life. We often design a system which users can view data but can only modify it using a trusted interface. An outstanding characteristic of this model is that it can indirectly prevent buffer-overflow attacks. A good implication of this model is Secure Bit [CITE]. The idea is that other process has low integrity and hence data coming from other processes should not be used as control (high integrity) data. Similar concept also applies to LOMAC [CITE], which assumes high integrity for system and low integrity for network. Once a program read data from network, it is automatically downgraded to low integrity and cannot access system file. This prevents a successful forking root shell from modifying a password file. However, it does not stop buffer-overflow attacks in general.

2.3.4.2 Multilateral Security

While multilevel security dues with information flowing between levels, multilateral security dues with information flowing between departments. In real life, controlling information flowing between organizations is critical. A good example is outsourcing. For example, a law firm may consult several companies with similar business. In such cases, there may be conflict between the customers.

To ease understanding, we will describe *China Walls* model as an example of multilateral security. Brewer and Nash [CITE] proposed this model by using the idea from internal rules in a financial firm. The idea is that if a person is recently working for a company in certain sector, this person then cannot work for another company in this sector for a certain period of time.

This concept also applies to reverse engineering by dividing workers into two groups. The first group studies a system and writes specifications. The second group creates a new system from the specifications. This way, the second group did not directly see the original system.

Another issue that has to be handled in multilateral security is privacy. Passing data, especially medical data, from one department to another should not violate privacy of a person. However, in certain cases (e.g. emergency, life treating), the necessary information must also be accessible.

In big picture, multilateral security is more complicated than multilevel security. The current models are partially the solutions.

2.3.5 Case Studies

Move all cases to Appendix.

2.3.5.1 ACLs, e.g. SunOS,

2.3.5.2 Windows NT

2.3.5.3 SELinux

SELinux [CITE] is an example of capability-based system. It is an implementation of FLASK architecture, a flexible and fine-grained mandatory access control, on top of Linux. Its security server contains a combination of Type Enforcement (TE) and Role-Based Access Control (RBAC). We will briefly describe the architecture and a sample configuration, and indirectly point out the benefit.

The architecture required that every subject (process) and object (file, socket, etc) be first labeled a security context, collection of security attributes. Each security context is then mapped to security identifier (SID), a unique integer, for being referred in Type Enforcement. It is worth qualifying that SELinux uses Type Enforcement to directly associate process to object and uses RBAC as an abstraction layer for authorizing users to a certain roles, In additions, SELinux also uses a user identity attribute that is orthogonal to legacy Linux userid .

The configuration files are divided into three groups: security_classes, initial_sids, and access vectors. The details are somewhat lengthy and require a lot of attribute declarations. The general idea is to specify security class, initial secure context (SID) and access controls.

2.3.5.4 Policies (HIPAA)

2.3.6 Exercises

1. Do you think that legacy UNIX file's permission scheme is sufficient for general purpose applications? Please justify your argument.
2. Windows 2000 has added an extension to Access Control List system (e.g. run "cacls [filename]" from the command prompt), which allows administrator to specify both allow list and deny list. Do you think this is a better access control system?
3. What is the FLASK architecture? What is the different between its Type Enforcement Model and a traditional Type Enforcement mentioned in this section?
4. LOMAC [http://opensource.sparta.com/lomac/] is an implementation of Low-Watermark Mandatory Access Control that can provide integrity of data. Please justify the security model of it.
5. In your personal computer system, give an example of cases when your system will be in insecure stage, draw a vector in which will put your system into the particular state, and suggest a security policy to eliminate such transition.

6. Given that a classroom management system consists of several functions as follow:
    - View grade
    - Edit grade
    - Add Student
    - Add Class
    - Give Assignment

    Assuming that users are students and instructors, please state policies related to this grading systems. From this policy, classify a role and draw an access control matrix

2.4 Auditing

To sway an audience, you must watch them as you speak.
C. Kent Wright

2.4.1 Introduction

Among the three components, Auditing is perhaps the easiest component to implement and is the fundamental of secure system. There are two concepts for auditing: (1) audit trails and (2) quality improvement. If a system works perfectly, there is no need for auditing. However, there is no such system. Having audit trails allows administrator to validate the system. In addition, once a problem occurs, it can easily be traced to the source of the problem and hence be fixed.

This process is somewhat similar to "Continuous Quality Improvement" or "CQI". The idea is evaluation of structure, process, and outcome, which focus on improvement efforts to identify root causes of problems, intervene to reduce or eliminate these causes, and take steps to correct the process. A good example is a firewall system. Firewall system allows administrator to create rules for filtering data. However, deriving such rules to prevent attacks requires continuous monitoring and analyzing of data.

2.4.2 Audible Systems

This is one quality that is lacking in most system. To create an application that can be audible, an ability of log every activity is required. The most common form is a log file. Nonetheless, not every system provides a mechanism for such function. For example, accounting an access to a file is not possible on every system. Even worse, some systems save only time of creation of a file and ignore the latest modification date. Such operation may be possible on a modern operating system (e.g. Windows NT), but it is not trivial on all (e.g. UNIX).

Push it another step, we sometime see a poor quality log file, which does not provide enough information to say, “who do what at when”. A good log file should give sufficient details to identify subjects, actions, locations, and instance. A related issue is the integrity of the audible trails. The system must ensure that a user cannot edit any trails record occurring from his/her actions to the system.

2.4.3 Analysis

Given a log file, the important step toward a more secure system is log analysis. Due the the large amount of data, most of the time, data from a log files is too much to be useful and is ignored by administrator. However, a simple association analysis or abnormally detection can be helpful in that it provides sufficient information for finding a problem. A good example is an intrusion detection system, MIDS, which can identify SLAMMER SQL worm by simply applying data mining to a log of network packets.

2.4.4 Case Studies

Move to appendix.

2.4.4.1 Intrusion Detection

2.4.4.1.1 Profiling

2.4.4.1.2 Signatures

2.4.4.1.3 NIDES, EMERALD

2.4.4.2 Intrusion Prevention

2.4.4.2.1 Firewall

2.4.4.2.1.1 Hardware (IP, ports, etc.)

2.4.4.2.1.2 Software (WinXP)

2.4.4.2.2 Intrusion Prevention Hardware (Juniper.com)

2.4.4.3 Virus Checking signatures

2.4.4.4 Spyware Detector signatures

2.4.4.5 Forensics

2.4.4.5.1 Disk

2.4.4.5.2 Network

2.4.5 Exercises

1. Most people argue that a secure system exists without accountability (e.g. a log file) ? Do you agree or disagree? Justify your argument.
2.