IS53012B/A Computer Security

Dr Ida Pu

Room 10, 29 St James
Goldsmiths, University of London

2019-20 (since 2007)

Part I

Authentication

Authentication

Assure that you are who you say you are, e.g.

1. password systems
2. fingerprint, or retina verification systems

Assure that a communication is authentic

1. Single message
2. Ongoing communication messages
3. X.800
    1. Peer entity authentication: provides confidence about the identity of a communication peer. It offers protection use at the establishment or during the data transfer phase.
    2. Data origin authentication: provides confidence about the source of a communication data unit. It does not offer protection against the duplication or modification of the data.

Authentication may be required without confidentiality

For example,

- The same message is broadcast to a number of destinations, e.g. users
    - inform “the college network is unavailable” in plaintext with an associated authentication message tag
    - alarm signal in a control centre
- The receipt is too busy for decryption and the authentication is carried out selectively on randomly chosen plaintext messages.
- Frequently used software for execution with an authentication tag

Message Authentication Code (MAC)

The technique uses a secrete key to generate a small block of data referred to as the MAC and append it to the message.

Suppose Alice and Bob shares a key k, and Alice wants to send a message m to Bob.

Alice uses a short message a to generate her MAC

1. $MAC_a = f(k, a)$
2. $c = append(MAC_a, m)$

Bob has a MAC_a, upon receipt of c

1. Computes $MAC_b = g(k, c)$
2. if $MAC_b == MAC_a$ then authentication test is passed.

Part II

Cryptographic hash functions

Review Hash functions

- A hash function takes data of arbitrary size and returns a value in a fixed range.
- If we compute the hash value of the same data at different times, we should get the same answer. If not then the data has been modified.

Review Hash functions

A hash function h acts on data x and returns a value $h(x)$. The hash function should have these 4 essential properties:

1. Given x it should be easy to compute $h(x)$.
2. The input x can be of arbitrary length.
3. Given a value y, it should be hard to find an x such that $h(x) = y$.
4. It is hard to find two different inputs x_1 and x_2 such that $h(x_1) = h(x_2)$.

Why are these important?

1. To make computations fast and efficient.
2. So that any message can be hashed.
3. To prevent a message from being replaced with another with the same hash value.
4. To prevent the sender claiming to have sent x_2 when in fact the message was x_1.

The Secure Hash Algorithm SHA-1

- Arbitrary length input is divided into blocks of 512 bits.
- The last block is padded to make 448 bits and the rest of the block is used to give the length of the input before padding.
- Each 512 bit block is split into 16 32-bit words denoted by $w(0), w(1), \cdots, w(15)$. These are expanded into 80 words $w(0), \cdots, w(79)$

The Secure Hash Algorithm SHA-1

- Initially 5 32-bit words $h_0, \cdots, h_4$ are given particular values.
- For each 512-bit block, SHA-1 operates on $w(0), \cdots, w(79)$ and $h_0, \cdots, h_4$ using shifts, bitstring operations (and, or) and modular arithmetic $\mod 232$ to produce new values of $h_0, \cdots, h_4$.
- These new value of $h_0, \cdots, h_4$ form the initial data for the next block which is expanded and operated on as above to produce the next values of $h_0, \cdots, h_4$.

This process of generating new values of $h_0, \cdots, h_4$ continues until all of the 512 bit blocks has been processed. The final set of values $h_0, \cdots, h_4$ are then concatenated to form a 160 bit value which is the output of the hash function.

SHA-1 and digital signatures

Bob
instead of sending and signing a message m,

1. hashes m to get $SHA(m)$ and then encrypt $SHA(m)$ using his private key to get a signature s.
2. Bob then sends the pair (m, s) to Alice.

Alice
on receiving the message,

1. decrypts s using Bob's public key hashes m to get $SHA(m)$
2. If these two values are the same then the message is authenticated.

Diffie-Hellman Key Exchange

- The Diffie-Hellman protocol allows 2 people to use random values and yet each generates the same symmetric key without transmitting the value of the key.
- The security of the protocol lies in the discrete log problem: given y, g and p find x such that $y = g^x \mod p$
- Alice and Bob need to agree on a key to use in a symmetric key cryptosystem. They choose a large prime number p and generator g.

Diffie-Hellman Key Exchange

Alice

1. Generates random number a,
2. Computes $x = g^a \mod p$
3. Sends x to Bob

Upon receipt of y from Bob

1. Computes $k = y^a \mod p$

Bob

1. Generates random number b,
2. Computes $y = g^b \mod p$
3. Sends y to Alice

Upon receipt of x from Alice

1. Computes $k = x^b \mod p$

Why Diffie-Hellman works

Alice has computed

$$\begin{aligned} k &= y^a \mod p \\ &= (gb)^a \mod p \\ &= g^{ba} \mod p \\ &= g^{ab} \mod p \end{aligned}$$

Bob has computed

$$\begin{aligned} k &= x^b \mod p \\ &= (ga)^b \mod p \\ &= g^{ab} \mod p \end{aligned}$$

So Alice and Bob both have the same value of k

How secure is it?

- We assume that cryptanalyst Charles knows the values of p and g and that he eavesdrops on the exchange between Alice and Bob
- so that he also knows x and y.
- However, unless Charles can solve the Discrete Logarithm Problem (DLP), he is unable to find a or b.
- It is believed that it is just as hard to find k from x and y without finding a or b.

The Needham-Schroeder Protocol

- This is another protocol for exchanging keys between Alice and Bob.
- This time they use only symmetric key cryptography
- But
  They need a trusted third party (TTP) or Server (S).

The Needham-Schroeder Protocol

- Alice and the server have a key K_{AS}
- Bob and the server have a key K_{BS}
- Alice and Bob want to establish a shared key K_{AB} so that Alice can send Bob a message.

Outlines

They communicate with each other and the server as follows:

1. Alice sends the server S the names of Alice and Bob to request a session key to be generated.
2. The server sends to Alice:
   i. The name of Bob
   ii. A session key for Alice and Bob to share
   iii. The name of Alice and the session key are both encrypted using K_{BS}.

   All 3 items above are encrypted using key K_{AS}
3. Alice uses key K_{AS} to decrypt the items sent to her in step 2. Alice now knows the session key with Bob K_{AB}.
4. Alice sends Bob the value of 2iii which is the name of Alice and the session key K_{AB} encrypted with K_{BS}
5. Bob decrypts the name of Alice and the session key K_{AB} using his key K_{BS}. Now Bob knows the session key K_{AB} for his communicate with Alice.

Demonstration

Let A be ID Alice and B ID Bob, $x \Longrightarrow y$ be that x is sent to y, K_{xy} be that the key from x to y, $eK(x)$ be that x is encrypted using the key K

Alice	Server	Bob
1. $A \Longrightarrow S$: A, B		
2.	$A \Longleftarrow S$: $eK_{AS}(B, K_{AB}, eK_{BS}(A, K_{AB}))$	
3. Alice decrypts to get $B, K_{AB}, eK_{BS}(A, K_{AB})$		
4. $A \Longrightarrow B$: $eK_{BS}(A, K_{AB})$		
5.		Bob decrypts to get A, K_{AB}

Analysis

1. Good even if Charlie eavesdrops on the communications because of the encryption.
2. Charlie cannot masquerade as the Server (to Alice) because Charlie does not know K_{AS}
3. Charlie can cut off the message from Alice to Bob because Bob would know nothing about the message from Alice until he receives the message. (fix: to let Bob use K_{AB}) to send Alice a message signifying his receipt of K_{AB})
4. Charlie can fake the reply from the Server in step 2 by sending Alice a replay from a previous run of the protocol. (fix: to use *nonce*. A nonce is a number used once and includes some extra information, e.g. time-stamp and a unique identifier, e.g. a random number)
5. Alice needs to store only the K_{AB}.

Full Needham-Schroeder Protocol

Let N_x be a *nonce* used by x. A nonce is a number used once, and includes some extra information, e.g. a time-stamp and a unique ID (random number).

Alice	Server	Bob
1. $A \Longrightarrow S$: A, B, N_A		
2.	$A \Longleftarrow S$: $eK_{AS}(B, N_A, K_{AB}, eK_{BS}(A, K_{AB}))$	
3. Alice decrypts to get $B, N_A, K_{AB}, eK_{BS}(A, K_{AB})$		
4. $A \Longrightarrow B$: $eK_{BS}(A, K_{AB})$		
5.		Bob decrypts to get A, K_{AB}
6.		$A \Longleftarrow B$: $eK_{AB}(N_B)$
7. $A \Longrightarrow B$: $eK_{AB}(N_B - 1)$		

Part III

Identification

Security challenges

All software systems, especially those with general purposes software such as operation systems, face two security challenges:

- shared access needs to be controlled
- user interface to allow the access

Two security issues regarding authentication:

- Who you are
- Verify you are whom you say you are

Two stages

1. username for user identification
2. password for authentication (verification of identification)

Threats to password systems

steal password access to a list of usernames and corresponding passwords
guess password exhaustive or intelligent search
password spoofing pretend to be the system and lure the user to hand over the username and password without knowing it
bad user interface flawed interface design

Example

User interface can release system information and help exhaustive attacks.

```
welcome to xxxx system
enter user name:  adams
invalid user name
enter user name:
```

Example

```
welcome to xxxx system          welcome to xxxx system
enter user name:  adams         enter user name:  adams
enter password:  *******        enter password:  *******
invalid access                  invalid access
enter user name:                enter user name:
                                enter password:
```

Defence solutions

protect password files cryptographic protection
password salting avoid username collision
challenge-response systems strengthen password systems by adding intelligent elements or control
biometrics making use of individual features of human body
access control monitor and regulate the access to different part of a computer system

Protect passwords

A password file can be protected using a one-way function f:

1. The system receives a username and password x from the user
2. It uses the one-way function on the password to transform it into a set of characters $y = f(x)$
3. The system does not store the password but instead stores y indexed by the username
4. To verify a user, the system asks for the username and password x' and computes $y' = f(x')$
5. If the value of y indexed by the username is the same as y' then the user is authenticated.

Password salting

- This process overcomes certain problems associated with a large user base where it is possible that two users may have the same password
- Before the password is (encrypted and) stored, the system adds some 'salt' such as appending the username. Now all passwords should be unique.

Example

`ma601mb, ma602mb`

Something you have

- Some for low risk situation and others for top security environment
- Based on personal information, something only you are likely to know, e.g. your mother's maiden name, date of birth or home postcode, something only you have, such as a credit card, standard order or direct debit in an bank account
- Fingerprints, retina patterns, palm prints
- Based on where you are - access may only be available in a secured area
- Asking to enter an answer to a specific question, e.g. favourite places, first-time teacher
- Informing the owner about via mobile phones, etc.

One-time passwords

- Also called a 'challenge-response system'.
- A password that changes every time being used.
- The computer system assigns a static mathematical function, instead of static phrase to a user.

One-time passwords

Example

$f(x) = x + 1$	The system prompts a value x and expects the user to input $x + 1$ as the password, or $f(x) = 3x^2 - 9x + 2$ or
$f(x) = p_x$	user inputs the xth prime number
$f(x) = d \times h$	user inputs the product of the date and hour
$f(x) = r(x)$	user is prompted with a seed x of a random generator available for both the system and user; and inputs the first random number generated (or xth)
$f(a_1a_2a_3a_4a_5a_6) = a_3a_1a_1a_4$	user is prompted with a string, and inputs a transform predefined
$f(E(x)) = E(D(x) + 1)$	user is prompted with an encrypted value $E(x)$, and first decrypts the value and transforms it by adding 1 (or in other ways), and then inputs the encrypted.

Authentication failure

A password system may

- accept an unauthorised user
- reject an authorised user

Facts

- People like to choose an easily memorable password
- A spelling checker Carries usually online dictionaries of the most common English words, as many as 80,000 words
- It only takes 80 seconds to try all of these words as passwords
  How long does it take to check a combination of two words? .. of three words? .. of four words?
- Of 3,289 passwords in a research in 1979, 89% passwords could be uncovered in about one week's worth of 24-hour-a-day testing, or less than a one-millisecond-per-password check

Facts

- Of 15,000 passwords in a repeating experiment in 1990,
  - 2.7% of the passwords were guessed in only 15 minutes of machine time
  - 21% were guessed within a week
  - 28.9% consisted of only lower-case alphabetic characters
  - The average length of the passwords is 6.8 characters
- On-line bank Egg found in 2002 that
  - full 50% of their on-line banking customers' passwords were family members' names
  - 23% were children's names
  - 19% used spouses' or partners' name
  - 9% own names
  - 8% pet names
  - 9% celebrity names
  - 9% football starts' names
  - 35% are deduced from syllables and initials of own names.

Password guessing

The attackers or cryptanalysts often take the following steps to guess a password:

1. no password
2. the user ID
3. similar to user's name
4. common words list (e.g. god, sex, love, money; password, secret, private), or common patterns (e.g. asdfg, aaaaaa)
5. short college dictionary
6. complete English word list
7. common non-English language dictionaries
8. short college dictionary with capitalisations (GoldSmiths), or substitutions (g0ldsm1ths)
9. complete English with capitalisations and substitutions
10. brute force, lowercase alphabetic characters
11. brute force, full character set.

Summary

We have discussed about the

- process of identification and authentication
- threats such as password guessing and password spoofing and ways the user and the system can protect themselves against these threats
- protection of the password file using a one-way function.

Part IV

Access control

Objects and subjects

- A multi-user distributed computer system offers access to objects such as resources (memory, printers), data (files) and applications.
- The system offers this access to subjects such as users, processes and other applications

Subjects and objects represent respectively the active and passive parties in a request for access.

In defining access controls, you can specify

- what a subject is allowed to do, or
- what may be done with an object

Operations and modes

operations the system may give subjects' permissions to
- read
- write (including read)
- append
- execute an object
- modify contents or status (group, permissions)

modes control subjects' activities on an object
- observe: read only
- alter: read and write

	read	write	append	execute
observe	√	√		
alter		√	√	

Design tools

- Access control tables
- Access control lists
- Protect rings
- Graphs
- Lattice

Access control tables

- A matrix (table) whose rows are indexed by subject and columns are indexed by objects
- Every entry in the matrix is indexed by a subject and an object
- The entry of the table is the set of access rights for that subject over that object

Example

Subject=(Jones,Smith)
Object=(timetable.doc, install.exe,format.com)
Access-operations=(read, write, execute)=(r, w, e) for short

	timetable.doc	install.exe	format.com
Jones	r	w	e,r
Smiths	r,w	-	e

Access control lists

A security system is more likely to list the access rights according to the subjects or the objects:

Jones	timetable.doc:read;
	install,exe:execute;
	format.com:execute,read;
Smith	timetable.doc:read,write;
	format.com:execute;

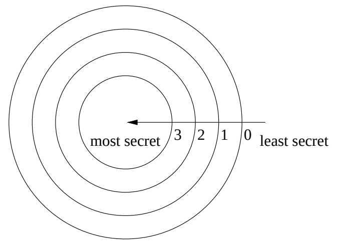
Groups:

- Subjects with same access rights can be organised in one group
- A subject may belong to more than one group
- The system may not be hierarchical
- Group membership and access rights may be expressed using graphs

Protect rings

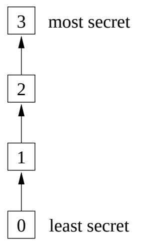
A hierarchical control structure:

- A subject is given a security level and can access all objects at that level or any level below
- A security level may involve operations too, e.g. read and write permission may be at a higher level than read only



Graphs

- directed graphs can be used to represent security levels and their structure
- each vertex represents a security level
- an edge from one security level to another shows that the second is at a higher level than the first

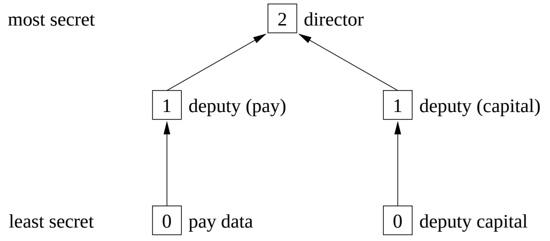


Example

Consider a security system involving three directors in a company:

- There is a director of finance and two deputy directors, one responsible for pay and the other for capital
- Each of the deputies has access to data that the other does not
- The director has access to both sets of data

The information can be represented by a direct graph:



Domination

If there is a direct path from vertex v_1 to v_2, then v_1 has the access rights to v_2, and vertex v_1 is said to *dominate* v_2.

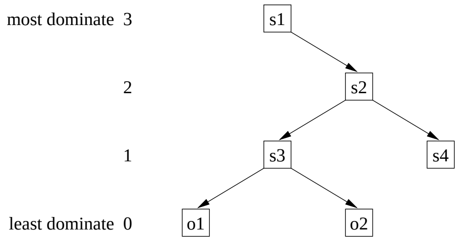
- If there is a path from a subject s to an object o then the subject has access rights to that object
- The path from s to o does not have to be a direct path. Subject s is said to dominate object o

Domination is anti-symmetric and transitive

- If a dominates b and b dominates a, then $a = b$, i.e. two vertices cannot dominate each other
- If a dominates b and b dominates c, then a dominates c.

Example

Consider subjects = $(s1, s2, s3, s4)$ and objects = $(o1, o2)$:



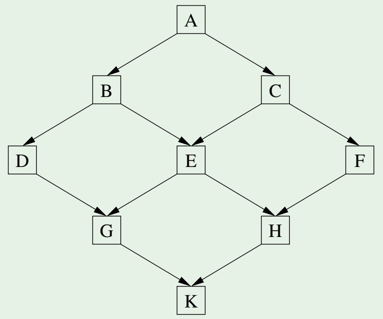
- o1 and o2 dominate no one
- s3 dominates both o1 and o2, i.e. has the access to both o1 and o2
- s2 dominates s3, o1 and o2, i.e. s2 has the access to s3, o1 and o2
- s1 has the most privileged access right
- Note the arrows represent the control direction which is opposite to the security level increase.

Lattice

Lattices are graphs with 2 particular hierarchical properties:

Example

A lattice with 5 security levels, 9 accesses.



Given 2 vertices x and y,

1. there is a unique vertex z which dominates both x and y. If vertex u also dominates x and y then u also dominates z
2. there is a unique vertex v which is dominated by both x and y and if u is also dominated by x and y then u is dominated by v

e.g.1. z=E, x=G, y=H, u=B (or C or A) 2. v=E, x=B, y=C, v=H (or G)

Definition

A lattice $(L, \leq)$ consists of a set L and a partial ordering $\leq$, so that for every two elements $a, b \in L$ there exists a *least upper bound* $u \in L$ and a *greatest lower bound* $l \in L$, i.e.

$$a \leq u, b \leq u, \text{ and for all } v \in L : (a \leq v \wedge b \leq v) \Longrightarrow (u \leq v)$$

$$l \leq a, l \leq b, \text{ and for all } k \in L : (k \leq a \wedge k \leq b) \Longrightarrow (k \leq l)$$

Two hierarchical properties

Given 2 vertices x and y,

1. there is a unique vertex z which dominates both x and y. If vertex u also dominates x and y then u also dominates z
2. there is a unique vertex v which is dominated by both x and y and if u is also dominated by x and y then u is dominated by v

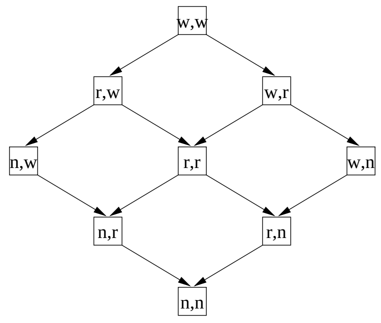
- These properties identifies the "least upper bound" and "greatest lower bound", and ensure that no portion of the graph looks like a 'butterfly'
- Advantage: for any two security levels, there is a unique minimum security level above both
- Such a system avoids conflicts
- Security systems based on lattices are very common.

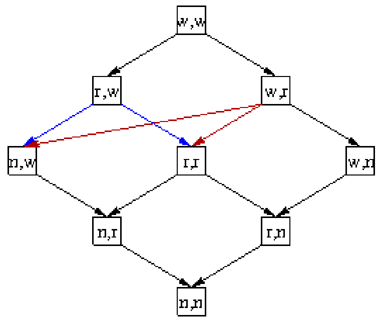
Example

In a file system there are two sets of files (classes I and II) and three levels of security for each of n, r, w (nothing, read, write). There could be a total of 9 different security access levels, each represented by a pair (x, y), where x represents the access to class-I files and y to class-II files. x or y can be one of n, r, w and shows the security level for each set of files. A subject with a security level of (r, n) would be allowed to read class-I files but neither read nor write with class-II files. We can use a lattice graph to represent this system.

Lattice and 'butterfly'

1. $n < r < w$ (denotes w dominates r and r dominates n)
2. All pairs: nr, nw, rw, rn, wn, wr, nn, rr, ww
3. $nn < nr = rn < rr, nw = wn < rw = wr < ww$





Summary

- An access control system offers subjects rights to perform operations on objects
- The operations available will depend on the security model and the application
- Access control tables represent access control for every subject
- Object lists may be used instead of matrices
- Groups can be used if different subjects have same access rights
- Directed graphs can be used to represent security levels and show domination
- Lattices are used by many security models to represent sophisticated design of access control.