## Authentication

* + Authenticate is the process of verifying the identity of a user or system.
  + Goal of authentication: bind identity to card/token/password/key
  + By using a secret key, the AS can verify the identity of AliCE
  + Certificate: is a token containing: identity of principla, public key, signature , timestamp
  + Hash is signed by atrust authority using her private key call signature

### II.Public-Key Infrastructure (PKI): bind identity to public key

* + Crucial as people will use key to communicate with principal whose identity is bound to key, Erroneous binding means no secrecy between principals, Assume principal identified by an acceptable name - called Common Name.
* A PKI consists of: Cetificates, Certificates Authority (CA), a resposity for retrieving certificates, A method of evaluating a chain of certificates from known public keys to the tartget name, amethod of revoking certificates
* **PKI Trust Models:**
  + Hierarchical CAs with cross-certification: Multiple root CAs that are cross-certified
  + Oligarchy model (commonly used in browsers):Browsers or Operating Systems come pre-configured with multiple trust anchor certificates, New certificates can be added( be careful), Bad certificate can be revoked
  + Distributed model: No root CA; instead, users certify each other to build a “web of trust”
* **PKI Security**
  + What happen if root authority is compromised? The certificate chain rooted from this CA is corrupted
  + PKI faces many challenges: Hash collisions: Obsolete hash algorithms, Weak security at CAs: attackes can issue rogue certificates, Users not aware of attacks happening

### III.Certificate

* Certificate Authority (CA): CA is a trusted third party who issues certificates
* PKI Hierarchy: Root CA -> Intermediate C -> End Entity C
* Certificate Verification: Decrypt the encrypted hash using CA’s public key, Re-compute hash from certificate end, compare hash values to check validity, check if the principal is Bob.
* Certificate Expiration: Certificate holds an expiration date and time, Certificate may need to be **revoked** before expiration, Revocation is **very important** to PKI.
* **Revocation:** request revocation, verify the request, publish the revocation, update the software
  + Certificate revocation list (CRL): A list of revoked certificates, Issued by CA, Signed by CA, Distributed to clients, Clients check CRL before using certificate
  + Online Certificate Status Protocol (OCSP): A protocol for checking the status of a certificate, Issued by CA, Signed by CA, Distributed to clients, Clients check OCSP before using certificate.

### IV.Password authentication

* Authentication is the process of verifying the identity of a user or system.
* How do you prove to someone that you are who u claim to be? Show **credential**
* **Credential can be:** Something you know (password, certificate,..)**,** Something you have (token, IP address, hardware/moblie device,..)**,** Something you are (biometric)
* How to steal or exploit passwords? After a sucessful intrusion: **Steal** install sniffer or keylogger to steal passwords, **Exploit** fetch password files and run cracking tools
* Use of strong password, why? Because weak password caused 30% of ransomware infections, Stolen credentials led to nearly 50% of attacks
* How to **store** password in the system? In password files indexed by user ID, in plaintext, Encrypted, hashed.
  + Hashing, Salting, Encryption, Password managers

**Password Hashing.** is the process of converting a password into a unique string of characters that cannot be reversed.

* When user enters a password: System computer H(password) and compares with the entry in the password file, System does not store the actual password
* Password hash function: Onewayness: given H(password), it is hard to deduce password, slow to compute; restrict the speed of brute force attacks

**Brute force attack**: after attacker gets ur password file, he tries to hash all possible values and compare the results witht e entries in the password file.

* + There are 94 candidate characters, 8 characters long password, 94^8 = 6.5\*10^14 possible passwords
  + But since password are not truly randomm. **Dictionary attack** is more effective

**Dictionary attack**: attacker uses a dictionary of common passwords to crack the password

* + Attacker pre-computes H(password) for every word in the dictionary; Pre-computing needs to be done only once and offine; One the password file is obtaioned, cracking is done immediately(search and compare); Password guessing tools also ultilize frequency of letters, password patterns, etc.

**Rainbow table attack**: attacker pre-computes H(password) for all possible passwords and stores the results in a table.

* + A space-time tradeoff, can purchase from the Internet.

**Salting** is the process of adding a random string to the password before hashing.

Character**:** is a random value chosen for each user(It chosen randomly when password is first set and stored in the password file), password hash = H(salt + password), Users with the same password have different entries in the password file, Salting adds randomness to password hash, make offline dictionary attack harder

**Advantages** of Salting

* + **Without** salting, attacker can pre-compute hashes of all dictionary words once: for **ALL** password entries, for all hash algorithms (the same hash function is used on all UNIX machines), Hash of identical passwords have identical value (one table can be used for all password files)
  + **With** salting, attacker must pre-compute hashes of all dictionary words once: for **EACH** password entry (with 12-bit salt, same password can have 2^12 = 4096 different hashes), for all hash algorithms, attacker must try all dictionary words for each salt value in the password file

#### Other Password Security Risks

* Weak password, default password, keystroke loggers, broken implementations, social engineering,
* Password strength, Hard to remember passwords,
* Password management issues: Password reuse, Password sharing, Heavy reuse

#### Way to improve password security.

* Password managers are software programs that store and manage passwords
  + What happen when password manager is compromised?
    - Password manager is a single point of failure: Malware, social engineering, brute force attack, insider attack
* Graphical passwords easy to remember, no need to write down: Draw a picture, select a point, select a color, Side channel attack may reveal the password
* Add biometrics: unique, hard to fake, no need to remember: Fingerprint, retina, voice, face, etc.
  + Require special hardware, hard to revoke, false positive, can be stolen
* Multi-factor authentication: Levarage **more than one** authentication mechanism for authentication, Google: Password + SMS, FIDO: Password + hardware

### Distributed authentication

#### Basic concepts

* **Potential threats:** User impersonation: a malicious user with access to a workstation pretends to be another user using the same station; Network impersonation: a malicious user changes the network address of his workstation to impersonate another workstation; Eavesdropping, message modification, and replay attacks
* How to prove user’s identity when requesting services from machines on the network? Many to many authentication: m clients, n servers; Public-key based solution: need m+n public-private key pairs – PKI; Secret-key based solution: mxn secret keys shared between each(client,server)
* What can be expect with **Kerberos**? **Secure** against attacks by passive eavesdroppers and active malicious attackers, **Transparent** so that users do not notice authentication and users’ effort is minimal, **Scalable** to serve a number of users and servers

### Kerberos a computer network authentication protocol that provides secure authentication for client-server applications by using cryptography.

### Kerberos step

1. Send password to AS - insecure to send plaintext password: Convert “password” into client master key: Ka| Ka is shared with Key Distribution Center (KDC)
2. Issue ticket - ticket needs to be encrypted. Otherwise, it can be forged.
   * Client -> KDC: “I am Alice, I want to talk to Bob” : IDa, IDb, timestamp, lifetime, TGT
   * KDC -> Client: encrypted session key and ticket: Eka(Ka-b, IDb, Tb): - Ka-b: session key geneerateed by KDC for Alice and Bob; - Tb = EKb(Ka-b, IDa, IDb)

3. Send password each time to obtain different tickets- separate authentication for services: Two step authentication with KDC(key distribution center) and TGS(ticket granting server)

**Kerberos Messages**

* Message between Client and KDC:
* Client → KDC: I’m Alice; I request a TGT: IDA, timeA
* KDC → Client: here’re the encrypted session key and encrypted TGT ticket: 𝐄KA (KA−TGS, IDTGS, timeKDC, lifetime,TGT)| TGT = 𝐄KTGS (KA−TGS, IDA, IDTGS, timeKDC,lifetime)
* Messages between Client and TGS  
  – Client → TGS: I’m Alice; I want to talk to Bob; here’s my TGT ticket.: IDA, IDB, TGT, authenticator1 |
* - TGS → Client: here’re the encrypted session key and encrypted service ticket  
  ■ 𝐄KA−TGS (KA−B, ID𝐵,timeTGS, lifetime, TB) | TB = 𝐄KB (KA−B, IDA, IDB,timeTGS, lifetime)  
  ■ Messages between Client and Network Server  
  – Client → Server: I’m Alice; here’s my service ticket.  
  ■IDA, TB, authenticator2  
  – Server → Alice: Let’s talk.: EKA-B

#### Kerberos Discussion

* The first single sign-on system - sign-on once, access all resources
* Design goal: Transparent, Scalabe, Client authentication
* Remain issue: no server authentication, ticket hijacking and replay attack

#### Kerberos Term keys

* Long term key: use to derive short term key
* ShortTK: separate session key are used for each client-server pair, same pair session key is reused
* Ticket are short term credential, can be stored in memery or disk.
* It provides a centralized authentication service, it can support mutual authentication, Entirely based on symmetric cryptography
* Less keys to remember for clients: KDC maintains long-term secret keys for each client and server, but servers don’t; Client requests short-term session keys(ticket + session key)from KDC and manages them locally. Less communication overhead (client sends both ticket and authenticator toserver, so no need to wait), More scalable in a large distributed system.

**Kerberos Security**

* - How to support reuse of TGT while defending replay attack? We add timestamps to message. The ticket can be valid for a limited time.
* - Client authenticates to KGS and network servers with an authenticator

**DATABASE SECURITY**

**Database System:** a collection of data and set of rules that organize data according to the relationship among them.

* + **Security:** enforce S at all DS levels. DATA: need to enforce data integrity and necessary privileges | DATA FILES: protect by permission and encrypt || DBMS: logical structure and software || OS: gateway to the data, authentication to DB system || Network: S access point to DB || Application || People
  + S Control: S policy; integrity protection; fault tolerance and recovery; Auditing and intrusion detection; access control models; privacy problem and control.
  + **S Objectives**: protect sensitive data from unauthorized disclosure, modification, denial of service attack

**Inference Problem:** to infer or deduce sensitive data from non-sensitive data.

* + Direct Attack: generate a query that return the value of sensitive fields of a few records -> use statistical DB/queries, this allows statistical use without revealing individual entries.
  + Indirect attack: use set of queries to infer sensitive info by exploit correlation or pattern in data-> data masking, query restriction.
  + **Tracker Attack**: use rules of logic and algebra to rewrite query. Used by web to track user across internet without their knowledge or consent. -> use extension that block tracking scripts,

**Controls for Inference Attacks:** no perfect solution

* **Three paths to follow**: Suppress obviously sensitive information (easy to implement, but it hurts database usability)|Track what the user knows (costly to implement): Used to limit queries accepted and data provided| Disguise the data using random perturbation, rounding, swapping (cause new problem with precision): Applied only to the released data, “Differential Privacy”

**-----🡪Possible controls**

* Query controls – Limit overlap between new and previous queries.
* Item controls – Suppression: query is rejected without sensitive data provided. ■ Limited response, combined results, random sample – Concealing: the answer is close to but not exactly the actual answer.
* Partitioning – Cluster records into exclusive groups and only allow queries on entire groups.

**SQL**(Structured Query Language): is A programming language primarily used for managing and manipulating data in relational database management system. Can be use for: data retrieval, data manipulation, data def, data control, data aggregation.

* Common Command: SELECT: to retrieve information from a database| INSERT: insert a new record into a table| UPDATE: modify an exiting record| DELETE: delete file|UNION: combine result of multiple queries into a single result.

**Implement AC in DB**: SQL implement AC with database privileges.

* RIGHT of Owner: SELECT: right to query the relation(one attribute)| INSERT:right to insert tuples| DELETE: right to delete |UPDATE: right to update(may apply to only one attribute).
* Privilege delegation: when owner grant privileges to other users. WHAT CAN BE GRANT? User can do an action to the entire table, inly SELECT on certain columns, can grant privileges with GRANT OPTION.
* Revoke Statement: we can revoke the grant privileges with: REVOKE<list of privileges> ON<relation or other object> FROM<list of authorization ID’s> ||Need **Append** the Rstatement w/ CASCADEoRESTRICT

**Access control** is the process of restricting access to objects in a system.

* Access control policy specifies the authorized accesses of a system: Managed by the database administrator (DBA)
* Access control mechanism implements and enforces the policy: Implemented in AC models, enforced by DBMS.

**Access control models:** Subject the active entity that requests access to an object**|** Object the passive entity accessed by a subject**|** Access Operation how a subject is allowed to access an object

**-** Similar access control for OS: Mandatory access control (MAC), Discretionary access control (DAC), Role-based access control (RBAC)

**Discretionary access control (DAC)** is a form of access control in which access rights are assigned to objects based on the identity of the subject requesting access.Widely used in multi-user systems

* What does **discretionary** mean? Access to data objects(files, directories, etc..) is **permitted based on the identity of the user**; Users can be given the ability of **passing on their privileges** to other users; **granting** and **revoking** privileges is regulated by an administrative policy.
* **Subjects:** A user is referred to by authorization ID(Typically, the login name.); There is an authorization ID: “PUBLIC” (Granting a privilege to PUBLIC makes it available to any authorization ID.)
* **Objects (on which privileges exist):** In database systems, the objects include stored tables and views; Other privileges are the right to create objects of a type, e.g., triggers.
* **Privileges:** A file system identifies certain privileges on the objects (i.e., files) that it manages, typically, read, write, execute.

**Role-based access control (RBAC)** is a form of access control in which access rights are assigned to users based on their roles within an enterprise.The AC is centered around the concept of a role, is a semantic construct.It provides good flexibility and granularity.

* + NIST’s RBAC architecture: U(users), R(roles), P(Permission), S(session).
  + UserRole(U): set of roles that U is authorized to occupy, role less than object/ RolePrivs( R): set of priv granted by a role, roles less than users
  + Privi Inheritance: roles can inherit access right from other roles, use role hierarchy to the relationship.
  + Constraint: can def relationship among roles or condition related to roles. WAY: Multually exclusive roles, Cardinality, Prerequisite roles.
  + Limitation: RBAC cannot enforce the way the principles are applied. System admin configure to violate.

## OS SECURITY

* **OS:** a software program that manages computer hardware and software resources and provides common services for computer programs. Is an interface between users and computer hardware. The fundamental tradeoff of OS security is the tradeoff between sharing and protection.
* Component: kernel, file system, device driver and user interface,…
* Manages Resource: disk drives, CPU, memory, I/O devices, network interfaces.
* OS Security Feeature:enforced sharing, inter process communication and synchronization, protection of critical OS data, user authentication, memory protection, file and I/O device AC, interface hardware
* **Separation**: is dividing the OS’s resources and function into distinct components or layers. Help protect user from each other.
  + What do we need to protect? General control and access to resources: shared I/O devices, serially reusable I/O devices, network resources|| Memory protection: Sharable programs, protecting OS kernel, process isolation. || File protection: Sharable data.
* Way to Implementation: physical S, temporal S, Logical S, Cryptographic S, Combination of two or more.
* **Level of Protection:** not P, isolation, share all or share nothing, share but limited access, limited use of an object. This is to support necessary sharing, but still protect sensitive info.
* Kernelized Design: Kernel mode: unrestricted access to all the resources | User mode: no direct access to hardware|| Switching from User mode to Kernel mode: voluntary(use system calls) or involuntary.
* **Memory Protection:** most fundamental protection offered by OS
* Hardware mechanisms MP: FENCE: set boundary around a specific area of memory| Base/Bounds registers || TAG ARCHITECTURE: associates a tag with each memory address, indicate when it can be read, written or executed|| SEGMENTATION: a technique that divides memory into segment, with its access right and permission || PAGING: divides memory into fixed-sized and maps them to physical loca

**Access control in OS:** need to ensure security, privacy, compliance, resource management

* Policy object involves 3goals: Check every access, enforce least priv, verify acceptable usage
* How to determine the AC policy and implementation? Limited. Priv: balance between need access and the risk of inappropriate access| Granularity: specificity of AC | Access log: history of control access.
* **AC implement OR PROTECTION**
  + Reference Monitor: a mediator between the subject(user or process) requesting access. SHOULD BE: unpassable, assuredly correct, tamper resistant.
  + Protect object with AC matrix, AC directory, AC directory, AC list
  + UNIX File AC: <setfacl> assign a list of users and groups to a file| <mask> field to denote the maximum permissions for any name user/group.
* **Virtualization:** a method in which OS kernel is shared by multiple virtual environments. USE: allow for efficient use of hardware resource, enables rapid deployment and scaling of application
  + Security: SANDBOX is a protected environment prevent program from endangering the rest of the system. || HONEYPOT is a faux environment to lure the attacker.
* **Access Control Problems:** 
  + 1. DAC has no control over the flow of the info -> **Trojan horses** will try to bypass OS’s AC -> one defense is to use the trust OS (implement a mandatory AC policy in TCB, use reference monitor to check
  + 2. Don’t support fine grained AC, implement it is over complicated-> implement RBAC, ACLs instead
  + **Rootkit:** attack package that attain the root privilege, used to bypass or manipulated AC mechanism in OS. Hide malware on ur device.-> Use anti malware tools, secure boot, limit user privileges
  + **Memory Attack**: aim to exploit vulnerabilities in a system memory management mechanism, allow attacker to unauthorized access to sensitive data or carry out malicious activity on that system.
  + Hardware-based memory A: extrack data directly from the system M by physically accessing the memory chips (use tool) || Software-based memory A: exploit vulnerabilities in software to gain unauthorized access to sensitive info or execute malicious code in the system memory.
* **OS Hardening:** remove unnecessary, service, application, and protocol | configure user, group, authentication | configure resource control | add extra security control | test security on the basic OS.

## SOFTWARE SECURITY is to engineer software so that it continue to function correctly under an attack

* Way to find Software S: Code review( manual code inspection), Static Analysis( automate reasoning of the code with static program analysis), Dynamic Testing(run the software with various input and check for anomaly
* **Buffer Overflow:** went program attempt to store more data in a buffer than it can hold, very common, still a major concern. Normal condition: can cause system to fall || Adversarial condition: can allow attacker to violate the S of system( gain escalate control, access private data,..)
* What is BUFFER? A memory space in which data/ code can be hold; has finite capacity, often predefined size.Where is BUFFER? In data section of a process, in the heap, stack. Way to make attak? Attacker can insert specially crafted code to overflow and cost the system to crash or take control of the system.
* Cost of BO? wrong result, illegal exception, crash, unexpected transfer of control, privilage escalation,
* **BO REQUIREMENT**: attacker need to guess the location of return address in the stack (in theory we need to guess 2^32 memory space for a 32 bit, in real world it smaller space), to improve the success use NOP sledding.
* **BO Defenses**: Compile-time defense: harden the program to resist attack in new program (compiler flags, input validation, memory protection) || Run-time defense: detect and abort attack in exiting program( runtime monitoring, stack canaries, data execution prevention,
* **Stack Overflow** when info is written into a variable on a stack but it size exceed what was allocated.
  + Stack: a region of memory that is used to store local variable and function call frames. When SO happen? Went the amount of memory allocated to the stack is exceeded.
  + Attacker can exploit a SO vulnerability by crafting input that exceed the amount of memory allocated for the stack. COMMON Attack use: attacker use “buffer overflow attack” to attack a SO vulnerability. TYPICAL ATTACK METHOD: overwriting the return address of the current function call.
  + What attacker can do when gain access? Change the flow of program to run any code.
* **SOLUTION:**
* **Address Randomization:** don’t need to start the stack in fixed location. (by randomization the location of key data structure, location of the heap,…)
* **Use GNU Compiler Collection:** help return address to a safe region of memory

**Incomplete Mediation:** a failure of not checking and controlling access completely and consistently.

* **Injection attack:**when a program reuses other existing program and the system utilities to develop application on a system. Common type of attack: command injection, code injection, SQL injection
* **Defenses**: block assignment of form field value to global variables, only use constan value in command such as include/require.

**Complete Mediation:** validate all input(restric choice to validate one) , protect input sources, test all assumption of type and size of value before use

**1. User Alice wants to access the** File Server (FS) in a private network through Kerberos. So, she first  
logs in to the Authentication Server (AS) and authenticates herself using her password. During the  
login session, Alice and AS exchange the following messages:  
Alice → AS: message1= ID\_TGS||timestamp\_Alice  
Alice ⇓ AS: message2 = ENC( < K\_1||Ticket\_TGS||ID\_TGS||timestamp\_AS>, K\_Alice), where  
Ticket\_TGS = ENC( <K\_1||ID\_Alice||ID\_TGS||timestamp\_AS||lifetime>, K\_TGS).  
Notes:  
(1) K\_Alice denotes the secret key of Alice and K\_TGS denotes the secret key of the Ticket-  
Granting Server (TGS).  
(2) ENC(m, k) denotes using a symmetric key encryption algorithm to encrypt the message m  
with the key k and “||” denotes concatenation.  
a. Please explain how the AS could verify the secret key K\_Alice.  
Answer: AS is the authentication server. When Alice first logs in to the AS, she sends her  
credential to the AS. From her password, AS deduces the secret key K\_Alice. This key is shared  
between Alice and AS.  
b. What is K\_1? Who generate it?  
Answer:  
K\_1 is sent to Alice by the AS in message 2. So, it is generated by the AS.  
This message is encrypted using the encryption algorithm ENC() and the secret key K\_Alice. So,  
K\_1 is securely sent to Alice.  
K\_1 will be used for Alice to authenticate to the TGS server. So, it is a secret session key, shared  
between Alice and the TGS.  
c. Can Alice read the content of Ticket\_TGS? Can Alice forge the ticket? Can Alice reuse the  
ticket in another time?  
1) No, Alice cannot read the ticket. Ticket\_TGS is encrypted by the key denoted as K\_TGS. This  
is the secret key of the TGS server, which is shared only with the authentication server AS.  
So, Alice does not know the key and cannot read the content of the ticket.  
2) No, Alice cannot forge the ticket. For the same reason that she does not hold the secret key.  
3) The ticket has a value called timestamp and another value called lifetime. So, if the ticket is  
still valid within its lifetime, it can be reused by Alice. Otherwise, it cannot.

d. How could the ticket-granting server authenticate Alice?  
Answer: Alice will send the TGS ticket received from the AS in message 2 to the TGS server. Alice  
also encrypts some information about ID\_Alice, timestamp, and lifetime, etc. using the secret  
key K\_1. This is called an “authenticator”.  
The TGS could decrypt the ticket to get information about this authentication request: the user  
identified as ID\_Alice is requesting service at ID\_TGS (itself). Besides, it obtains the secret key  
K\_1 from the ticket. Meanwhile, the TGS could decrypt the authenticator to get information  
about Alice: ID\_Alice. With both information, the TGS authenticates Alice.  
**3. To prevent the inference** attacks, the database systems implement multiple controls. Please list  
these controls discussed in the lecture and pick one to explain which type of inferences it prevents.  
Answer: Inference attacks are confidentiality attacks against database systems (and their access  
control). You can pick any of the below to discuss.

1) Statistical queries and statistical databases: to prevent leaking sensitive data from direct  
inference, a database could disallow querying individual entries related to sensitive data,  
instead allow only statistical queries or store data in special databases that support statistical  
use only.  
2) Item control: database can (1) suppress the results if query is on certain sensitive data, such as  
not returning certain data; (2) conceal the results, such as modifying the data to be returned.  
Item control can be used to prevent indirect inferences, for example, suppress or conceal the  
results, if only a small number of records consist of a large portion of the data in the result. The  
goal is to make inference harder.  
3) Query control: keep a history of queries and limit the overlap between new and old queries. It  
could be used to control tracker attacks, which ask multiple queries and learn from the results  
based on their linear relations.  
4) Database partitioning: partition a database into exclusive groups and allow one user to query  
each group to prevent all types of inference attacks.  
4. What is the tracker attack? Can we use access control to prevent this attack? If so, briefly explain  
how this access control could be implemented?  
Answer: Tracker attack is a type of inference attacks in statistical databases. It uses a tracker query  
to bypass database’s count-based access control (i.e., if the number of results is less than a preset  
small number or larger than a preset large number, DBMS will refuse to return the results).  
Query control can be used to detect tracker attacks. Therefore, if we implement an access control  
based on query set overlap, we could prevent the attack. This access control could be implemented  
using any access control models, such as ACM, ACL, etc., by adding a rule to describe the  
intersection of query results that need to be controlled.  
**5. What is protection domain?**  
Answer: In operating systems, protection domain is a collection of objects that a process has access  
to. Each domain defines (1) a set of objects; and (2) the actions that can be invoked on each object.

**7. Set user (setuid) and** set group (setgid) programs are powerful mechanisms provided by the Unix  
system to manage access to sensitive resources through control invocation. However, because of  
this, this mechanism has potential security risk. Bugs in such programs have led to many  
compromises.  
Please briefly explain why this mechanism is needed in Unix and which security problems a bugged  
setuid program may cause.  
Answer:  
1) Why do we need to support setuid: Unix differentiates privileges for accessing system resources  
between normal users and superusers. However, for some tasks, we need to grant more  
privileges to a process of normal users, such as allowing them to change their own passwords.  
Besides, the privilege should be given only to this task, whose behavior is defined and restricted.  
To meet the need, Unix uses a special marking, which is a single bit called setuid bit, to tell the  
operating system using the change the effective user ID of the program to the one of the  
program owner.  
2) Which secure problems caused: since the setuid program will use the program owner’s  
privilege, but not the privilege of the user who run the program, it will cause privilege escalation  
if the program owner has more privileges than the user, for example, if the program is owned by  
root. If a setuid program owned by root has bugs, this program and its bug could be exploited by  
the attack with the root privileges.  
Software Security  
**8. Explain how to use “NOP sledding**” to assist buffer overflow attacks.  
Answer: A “NOP sled” is a run of NOP (no operation, do nothing) instructions, which are included  
before the desired shellcode to help overcome the lack of knowledge by the attacker of its precise  
location.  
In a buffer overflow attack, the attacker arranges for the transfer of control (via overwritten return  
address) to occur somewhere in the NOP Sled (guessing around the middle of the most likely  
location). Then, when control transfers, no matter where in this run it occurs, the CPU executes  
NOPs until it reaches the actual desired shellcode.

**9. Standard C library functions such as gets**(), strcpy(), sprint() are unsafe. Take gets() and fgets() as an  
example, explain why fgets() is more secure.  
Answer: Both functions allow reading a line from the standard input stream and store the input into  
a string. gets(char \*str) stores the input in a string pointed by “str”, while fgets(char \*str, int size,  
FILE \*stream) defines an integer “size” to specify the maximum number of characters to be copied  
to str.  
Because str is pointer, we need to allocate a buffer in the memory for it. gets() does not have any  
bound check on the spaced used by this buffer. It is vulnerable to buffer overflows. In contrast,  
fgets() defines the size of the buffer, so it is considered a safer function compared to gets().  
[Optional: However, fgets() only reduces the buffer overflow risk, but not eliminate it, since it does  
not do the checking itself and the size used may still exceed the buffer. The developers need to  
enforce the checking by themselves.]  
**10. The below program has a buffer** overflow vulnerability. Please identify the unsafe function used in  
the program and explain why.  
Answer: In this example, gets() is used. It defines an array of 16 bytes but does not check if the input  
is less than this value. Since the program expects a user input which could be large, if it’s longer than 16 bytes, the malicious content would overflow to other memory space.

**Injection Attacks:** Common in scripting languages, When a program reuses other existing programs and the system utilities to develop applications on a system, ...  
■ Common types of attacks: Command Injection, Code Injection, SQL Injection (to be discussed in web security)

**Complete Mediation:**■Validate all input: Restrict the choices to validate ones  
■ Protect input sources: Critical data should not be exposed to untrusted user.  
■ Complete mediation: Test all assumptions of type and size of values before use  
**Race conditions** occur when two concurrent threads of execution access a shared  
resource in a way that unintentionally produces different results depending on the  
sequence or timing of two threads.

**Time of Check to Time of Use(**TOCTTOU) is a race condition, exploiting the delay between two times.  
■ Take away:  
– Attacker can change the “state” between two events, to bypass checking.  
– Access control should be checked universally.

How likely/easily is a TOCTTOU attack?  
– Timing would have to be perfect (the window between two time exists in ms).  
– Attacker still has a good chance, because:  
■ Attacker can force the victim program to perform an expensive I/O operation (to extend the time  
window).  
■ He only has to get it right once.  
■ He can run the attack program over and over.  
■ He can run many other programs to lengthen the time between check and open.  
– So, it is still challenging to avoid or eliminate this vulnerability.

**Race Condition Countermeasure**  
■ Eliminate the window: Atomic operations, Ensure no interruption during validation  
■ Restrict attacker’s action inside the window: Repeating check and using multiple times to make racing more difficult for attacker  
■ Enable sticky symlink protection: Restricting program from following symbolic links inside a sticky directory (e.g., UNIX)  
■ Prevent over-privilege: Program may have more privilege than. So, if possible, reduce its privilege before checking/using.

**General Principles**Check inputs, Check return values, Least privilege, Securely clear memory (keys, passwords, ...), Failsafe defaults, Defense in depth, Simplicity, modularity, Reduce size of TCB, Minimize attack surface, Use vetted components, Security by design, Open design

**Defensive Programming**  
1. Assumes nothing, checks all potential errors.  
– Consider all aspects of program execution, environment, and type of data it processes.  
– Assumptions need to be validated by the program.  
– All potential failures handled gracefully and safely.  
2. Be aware of the consequences of a failure and the techniques used by attacker.  
– Understand how failures can occur and the steps needed to reduce the chance of them  
occurring in their programs.

**-**------  
■ Writing Safe Program Code: race condition  
■ Handling Program Input: buffer overflow attacks, injection attacks  
■ Handling Program Output: injection attacks