ECS655U-ECS775P: Security Engineering

Week 9: Access Control, Memory Protection

Dr Arman Khouzani, Dr Na Yao

March 09, 2018

EECS, QMUL

Learning Outcomes

- Systems/OS security:
  - Introduction to Access Control
  - Introduction to Memory-Based Vulnerabilities

Introduction to Access Control

Access Control

Access Control is a broad term that describe the administrative, physical, and technical controls that regulate the interaction between *subjects* and *objects*.

- a *subject* is any active entity that requests access to a resource (an object), e.g. users, processes, etc.
- an *object* is a resource, a passive entity that is or contains the information that is needed by a subject, e.g., files, I/O, database entries, etc.
- an *access controls* is used to perform granting, preventing, or revoking access to an object.

Access Control

Table 1: Phases of Access Control Process

Phase	Purpose	Example
Identification	Who are you?	USER ID, IP Address
Authentication	Prove who you claim to be.	Password, badge, fingerprint
Authorization	Which resources can you access? What are you allowed to do with those resources?	User A accesses Resource B in read and write mode
Accounting	What have you done?	User A has modified Resource B on `Date:Time`

Access Control

- **Identification** should be *unique*, and *non-descriptive*, and have a *secure issuance*.

Table 2: Authentication Methods

Authentication Method	Description	Examples
by Knowledge	Something only the user knows.	Password, PIN
by Ownership	Something only the user has.	Smart Card, badge, token
by Characteristic	Something only the user is/does	Fingerprint, hand geometry, voice, keystroke dynamic, iris

Access Control

Authentication Methods (usually Authentication by Characteristic, e.g. through biometrics) may not be completely accurate and may be susceptible to errors:

- Type-I error (false rejection): when a known legitimate authorised user is rejected as unknown/unauthorised user.
- Type-II error (false acceptance): when an unknown/unauthorised user is authenticated as a known/authorised user.

Access Control

- Multi-factor Authentication (examples: Bank Card+PIN) – but be careful not to mix identification with authentication.

- **Authorization**: should conform with the following:
  - *Implicit Deny*: if no rule is specified for the transaction of the subject/object, the authorization policy should deny the transaction (conforming with the more general “default-safe” principle).
  - *Need to know*: a subject should be granted access to an object only if the access is needed to carry out the job of the subject (conforming with the more general “least-privilege” principle)

Access Control

- *Separation of Duties*: a single individual should not perform all the critical- or privileged-level duties. Important duties must be separated/divided among several individuals.

- **Accounting**: auditing and monitoring, through creation of “log” files that are specially protected (necessary for detection/investigation of cybersecurity breaches).

Access Control Models

- Mandatory Access Control: MAC
- Discretionary Access Control: DAC
- Role-Based Access Control: RBAC
- Attribute-Based Access Control: ABAC

Mandatory Access Control (MAC)

Mandatory Access Control: (aka rule-based access control) describes the situations where subjects cannot alter their access to objects and rather the access is enforced (by setting rules) through a system mechanism at the discretion of a centralised system administrator.

Discretionary Access Control (DAC)

Discretionary Access Control: (aka identity-based access control) describes the situations where subjects can set an access control mechanism to allow or deny access to an object at their own discretion.

- e.g. the Unix OS allows users to determine the read/write/execute access rights for their own files.

Role-Based Access Control: RBAC

Role-Based Access Control:

- For various tasks/job functions, “roles” are created, such that certain operations can be performed by specific roles.
- users are assigned particular roles. Users acquire the permissions to perform particular computer-system functions, not directly, but through their assigned roles.
- Note that a user can have more one role.
- Simpler to manage user rights because it becomes simply a matter of assigning appropriate roles to their accounts; this simplifies common operations, such as adding a user, or changing a user’s department.

Attribute-Based Access Control: ABAC

Attribute-Based Access Control:

- access rights are granted to users through the use of policies which combine “attributes” (properties) of the resource (object), the subject (user/process) and the environment.
  - example attributes of an objector: “the subject that created/owns it”
  - example attributes of a subject: “age: older than 12”.
  - example attributes of environment: “time: between 9:10–9:55 AM”
- The strength of ABAC is its flexibility and expressive power, but it is computationally heavy to implement/maintain for large systems.

Access Control Policy

An access control **policy** is a specification for an access decision function.

- The policy aims to achieve:
  - Permit the subject's intended function (availability)
  - Ensure security properties are met (integrity, confidentiality: also known as "constraints")
  - Enable administration of a changeable system (simplicity)

Bell-LaPadula (BLP) model

The two main rules of the “Bell-LaPadula” model:

- ▷ the **simple security property:** A process running at a security level can read only objects at its level or lower (no read up).
- ▷ the *** property:** A process running at a security level can write only objects at its level or higher (no write down).

- ▷ Hence, “Bell LaPadula” is to guarantee “Confidentiality” (why?)

Biba Model

The two main rules of the "Biba" model:

- the **simple integrity property:** A process running at a security level can write only objects at its level or lower (no write up).
- the **integrity * property:** A process running at security level can read only objects at its level or higher (no read down).

- Hence, "Biba" model is to guarantee "Integrity" (why?)

Access Control

Often provided using an Access Matrix:

- ▷ lists subjects in one dimension (rows)
- ▷ lists objects in the other dimension (columns)
- ▷ each entry specifies access rights of the specified subject to that object

- ▷ Access matrix is often sparse (huge memory for representation)

Access Control Mechanisms

But how do we implement an Access Control Matrix?

- **Access Control Lists:** An Access Control List of an object shows all the subjects who should have access to that object and what their access is. It is like a column of the access control matrix, so there is an access control list per each object.
  - Example: ACL(file 1) = [(process 1, $\{$read, write, own$\}$), (process 2,$\{$append$\}$)].
- **Capability Lists:** It is like a row of the access control matrix, so there is a capability list per each subject.
  - Example: CAP(process 1) = [(file 1, $\{$read, write, own$\}$), (file 2, $\{$read$\}$), (process 1, $\{$read, write, execute, own$\}$), (process 2, $\{$write$\}$)$\}$.

Access Control Mechanisms: Trade-offs

Advantage of ACL over C-List:

- ACLs are easier for human interpretation (e.g. by the administrator) to quickly identify who has access to a given resource, and are a more natural way of thinking about access control;
- It is also easy to remove rights on a particular resource in ACL (we only need to modify one list)
- ACL is particularly suitable when new resources may be added/removed but the users are pretty stable.
- It also scale up well and work in distributed settings.

Access Control Mechanisms: Trade-offs

Disadvantage of ACL compared with C-List:

- In ACL, it is difficult to grant/remove rights to a user on all resources.
- ACL is not suitable for situations where the users may change a lot, because each time a user changes, the entire C-Lists should be updated. In contrast, in C-List, if a user leaves an organisation, only the C-List of that user is removed, and if a new user joins, only a single C-List is created.

Introduction to Memory-Based Vulnerabilities

Memory-Based Vulnerabilities: Buffer Overflow

Consider the following C-code:

```c
#include <stdio.h>
#include <string.h>
int main(void){
  char s[15];
  int success = 0;
  printf(”Enter password : \n”);
  gets(s);
  if(strcmp(s, ”ECS655U775P”)){
    printf (”Incorrect password! \n”);}
  else{ success = 1;}
  if(success){
    /* Grant root privilege to the user*/
    printf (”Root privilege granted! \n”);}
  return 0;
}
```

Memory-Based Vulnerabilities: Buffer Overflow

this is a straightforward case of **buffer overflow**:

- ▷ 15 characters are reserved for the variable `s` , but the C function `gets(s)` reads the input characters and writes them inside `s` without checking the length.
- ▷ This means that if the input has more characters than 15, then the boundary of the buffer reserved for `s`, and overwrites the adjacent memory location
- ▷ The adjacent memory location holds the value of other variable defined after `s`, which in this example is the `success` variable.
- ▷ So if a long string is passed at the prompt, the password checking is completely bypassed: a user is authenticated without knowing the password!

Memory-Based Vulnerabilities: Buffer Overflow

- **Q:** How should the code have been written?
- **A:** Usage of functions that do not check the boundary of the buffer of variables before assignment must be avoided. Specifically, **Never** use **`gets(s)`**. Instead, use functions that provide the same functionality, but do check the length. For instance, we should have used **`fgets(s,15)`** instead.

Memory-Based Vulnerabilities: Buffer Overflow

Even when the programmers make mistakes, the compiler and OS can provide some protection mechanisms:

- ▷ for instance, using *canary values*, which demarcate the boundary of a buffer and are checked to see if their value has changed, indicating a buffer overrun. Upon such an event, the execution of the affected program can be terminated, preventing it from misbehaving or from allowing an attacker to take control over it.
- ▷ *Bound checking* is another technique, which checks accesses to each allocated block of memory so they cannot go beyond the actually allocated space, etc.

Questions?