Access Control

COMP 435
Fall 2017
Prof. Cynthia Sturton

Access Control: enacting a security policy
Which users can access which resources and with which rights

Who
Who

How
Which users can access which resources and with which rights

What

Subject

Right or Type

Object

Which users can access which resources and with which rights
Subjects

- Users
- Processes

Objects

- Users
- Processes
- Files
- Memory
- I/O devices

Access Type

- Read
- Write
- Execute
- Create
- Transfer
Best Practices for Access Control

**Best Practice**
- Universal application
- Least privilege
- Type checking

**Universal Application**
Every access by a subject to an object should be checked

**Non-Universal Application**
- Random checking
- Random auditing
- Selective checking
Least Privilege

Every subject should be granted the least amount of access necessary to do its job

Type Checking

Operations should be meaningful for the object accessed

Access Control Policies

- Discretionary Access Control (DAC)
- Mandatory Access Control (MAC)
- Role-based Access Control (RBAC)
- Attribute-based Access Control (ABAC)
Discretionary Access Control (DAC)

Access Control Matrix
+ Single listing of all objects
+ Eases revocation
+ No aliasing
- Sparse
- Inefficient

Access Control List
Access Control Directory

- Easy to implement
- Easy to understand
- Long lists
- Revoking access requires a search through every list
- Aliasing may cause ambiguous access rights

Permission vs. Authority

Type of actions or rights granted directly to a process for a given object
Authority

Type of actions or rights granted directly or indirectly to a process for a given object

Ambient Authority

All the extant permissions of the current execution context

Confused Deputy

A program running with multiple sets of permissions uses all permissions indiscriminately

SetUID

```
    Group   SetUID
    rwx     rwx     rwx   suid   guid
    Owner   Users
         myfile.exe
```
```c
int main(int argc, char *argv[]) {
    // compile code
    ... //write to log
    FILE *fp = fopen(argv[2], "w");
    //write to fp
    ...
    //write out statistics:
    fp = fopen("/etc/compiler_stats", "a");
    //write to fp
    ... }
```

```
$ gcc prog.c log.txt
$ gcc prog.c log.txt
$ gcc prog.c "/etc/passwd"
```

**Capabilities**

- Unforgeable token
- Possession of the token grants access rights
- Directly ties access right to object
- Think physical key

**Analogy:** Confused Valet
gcc prog.c log.txt

gcc prog.c "/etc/passwd"
> ERROR: no capability for passwd file!

```c
int main(int argc, char *argv[])
{
    // compile code
    ... 
    // write to log
    FILE *fp = fopen(argv[2], user_cap);
    // write to fp
    ...
    // write out statistics:
    fp = fopen("/etc/compiler_stats", system_cap);
    // write to fp
    ...
}
```

Mandatory Access Control

Discretionary Access Control

<table>
<thead>
<tr>
<th>Access List for secretFile.pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
</tr>
<tr>
<td>Bob</td>
</tr>
</tbody>
</table>

Discretionary Access Control

Alice: I have a file!
Bob: Can I see it?
Alice: Sure!
Security Levels

top secret > secret > confidential > restricted > unclassified

Bell-LaPadula Model

- Confidentiality
- No read up
  - Simple security property

Biba Integrity Model

- Integrity
- No write up
- No read down
- No write down
  - *-property
A Reference Monitor

- Complete mediation
- Tamperproof
- Verifiable