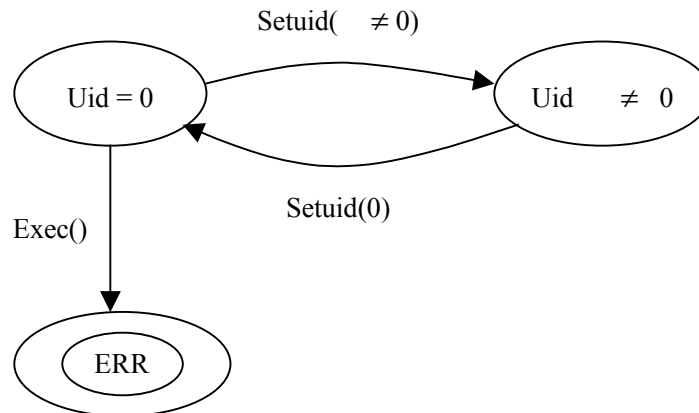


# CSE509 Spring 2006. System Security

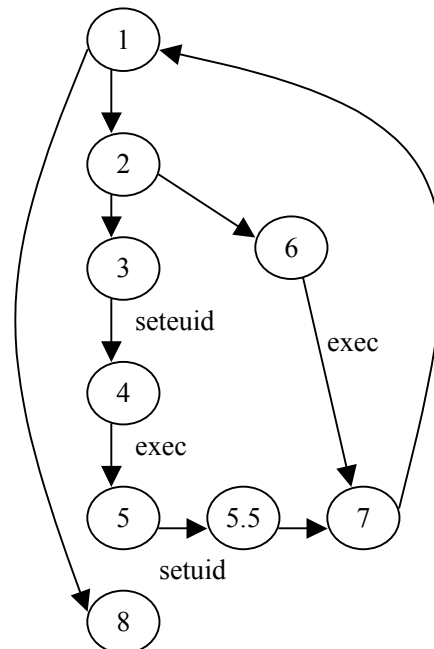
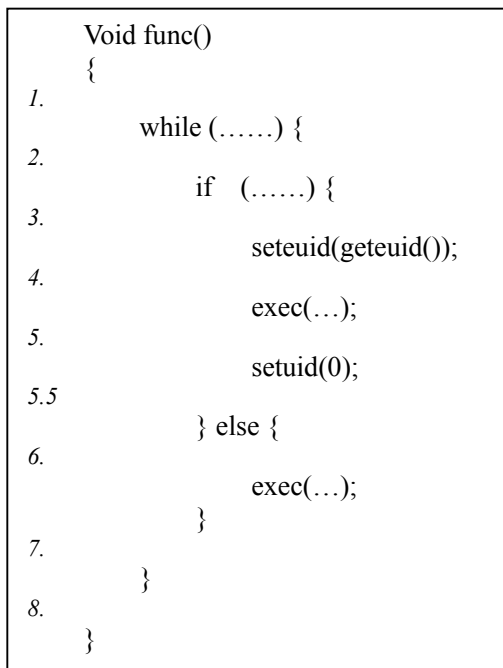
## Lecture Notes for 03/16/2006

### ● Model Checking

$L_p$ :



$L_{CFG}$ :



*Assume: Code starts with euid = 0.*

## CFG(Control Flow Graph)

CFG is a Finite State Machine.

Giving regular language  $L_{CFG}$ , security property is specified as a FSM, get  $L_P$ .

What we want is  $L_P \cap L_{CFG} = \phi$ .

$L_P \cap L_{CFG}$  is regular!

### Four steps to verify:

1. Specify security property as a FSA.
2. Convert input program into a FSA(PDA, PushDown Automata).
3. Compute product of machines from 1&2.
4. Check if product machine gives empty language.

### Tools:

MOPS  
SLAM  
BLAST  
MECA

### Choices in Model Checking design:

#### 1. Function Vs. whole program

MOPS, SLAM and BLAST are whole program checking. MECA is using function by function analysis.

#### 2. Soundness Vs. convenience

**Define:** A program analysis tool is *sound* if whenever it says a program doesn't have bugs, then the program doesn't have bugs

**Define:** A *false positive* is when an analysis tool claims that a program has a bug, but it doesn't

**Define:** Completeness – A *complete* analysis tool has no false positives.

**Theorem:** Choose 2 from *Soundness, Completeness and Termination*.

MOPS: sound, terminates

SLAM and BLAST: sound, complete

MECA: terminates, convenience

#### 3. Theorem Proving

MOPS has none, SLAM/BLAST have lots of, and MECA has a little.

**Unsound:** may not be security after using, but still helpful.

- Lasebeam Vs. shotgun approach
- Asymmetry of attacking Vs. defending

- Defender must close all vulnerabilities
- Attacker only need find one weakness
- Combined methods:
  - Surely to be a bug
  - Likely to be a bug
  - How about the percentage of possibility to be a bug

**MECA Results:**

- 1000's bugs
- Low false positive rate: 10%~100%
- Commercial coverage
- Misses real bugs

**Open:**

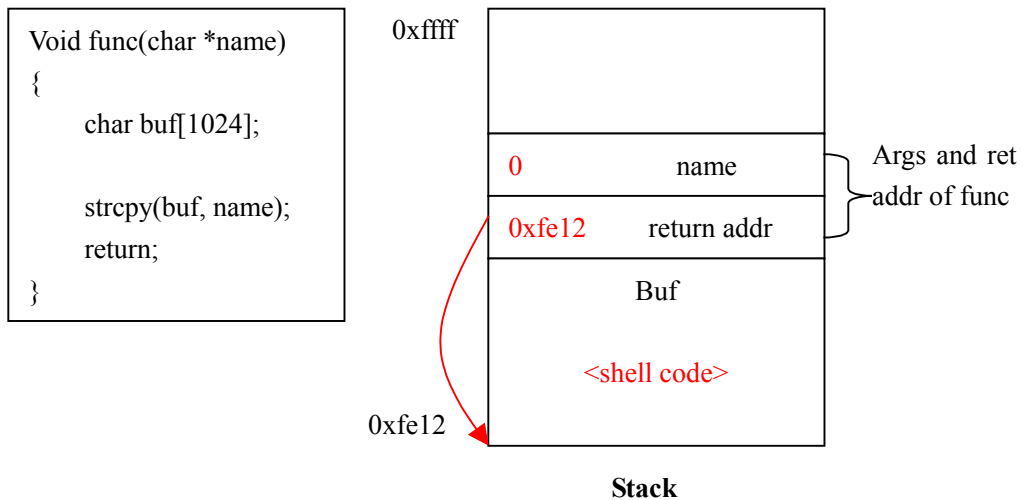
Design-level bugs.

**Summary for the four analysis tools:**

	Soundness	Completeness	Termination	Analysis approach	Theorem Proving
MOPS	Y	N	Y	Whole program	N
SLAM	Y	Y	N	Whole program	Many
BLAST	Y	Y	N	Whole program	Many
MECA	N	N	Y	Function by function	A little

● **Buffer Overflow**

**Example 1:**



The attacker makes

Name = "<shell code>\xfe12\0"

Name = "NOP NOP NOP <shell code>\xfe12\0"

**Q1:** How the attacker can know the return address need to be filled?

**A:** For example, the attacker find the victim machine is using Linux. Then the attacker need install exactly the same version of Linux on his local machine, download the same open source code. Run it, the attacker will be able to find the return address.

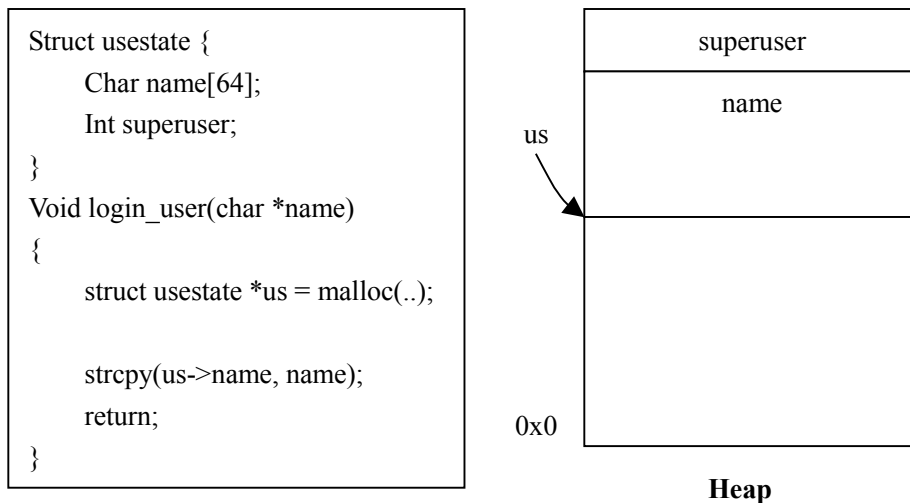
**Q2:** How to execute the shell code inside the buffer?

**A:** The attacker can use a decoder to eliminate forbidden bytes.

**How to fix?**

Use NX bit, forbid execution on stack.

**Example 2:**



Note: the 'superuser' field may be overwritten if the argument 'name' is long enough.

**Example 3:** return to libc attack.

```
Void func(char *name)
{
    char buf[1024];

    strcpy(buf, name);
    return;
}
```

