

CSE 409/509: Computer Systems Security

Lecture 15

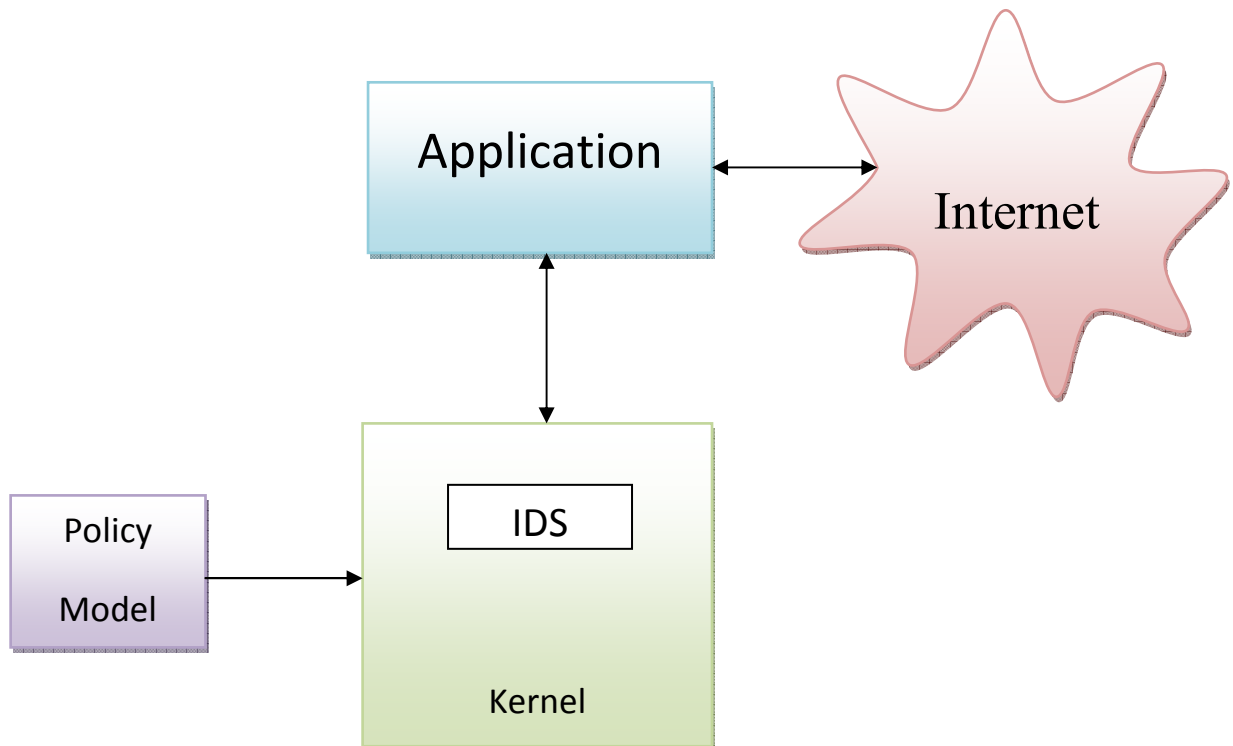
Intrusion Detection Systems/Sandboxing

/Host-based IDS/

It monitors system-calls to detect malicious behavior.

/Network IDS/

It analyzes packets and uses string-matching techniques to detect malicious behavior.



Intrusion Detection System (IDS)

A system that sets off an alarm to indicate that the application is being attacked.

Intrusion Prevention System (IPS)

A system that kills the application when its behavior deviates from specification.

Two matrices used by IDS

- False positives
- False negatives

The goal of the IDS is to try and reduce the number of FPs and FNs as far as possible.

Methods to generate models for IDS

1. Control Flow Graphs \Rightarrow NDFA (static analysis)
2. Log real traces, and use it to describe a model (dynamic analysis).

/Trace Based Intrusion Detection/

1. Levenshtein Distance
2. N-gram Model

Example:

Trace 1: open, read, write, close

Trace 2: read, read, close

Trace 3: open, read, exec

Let us now consider a **3-gram model**

Therefore the model would be:

| | |
|----------------------|---------------------|
| (open, read, write) | (open, read, read) |
| (read, write, close) | (read, read, close) |
| (open, read, exec) | |

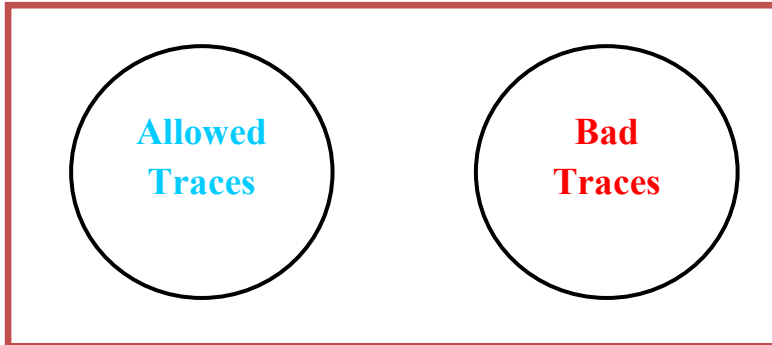
/Problem/

False Positives; May miss valid traces

/Advantage/

Can capture site-specific configurations (as you can block a particular code segment if required)

/Important Assumption in IDS/



A mimicry attack is possible when this assumption does not hold. In other words, if **Allowed Traces** \cap **Bad Traces** $\neq \Phi$, then a trace can be selected from the intersection of the two sets to carry out the mimicry attack.

Example of Control Flow:

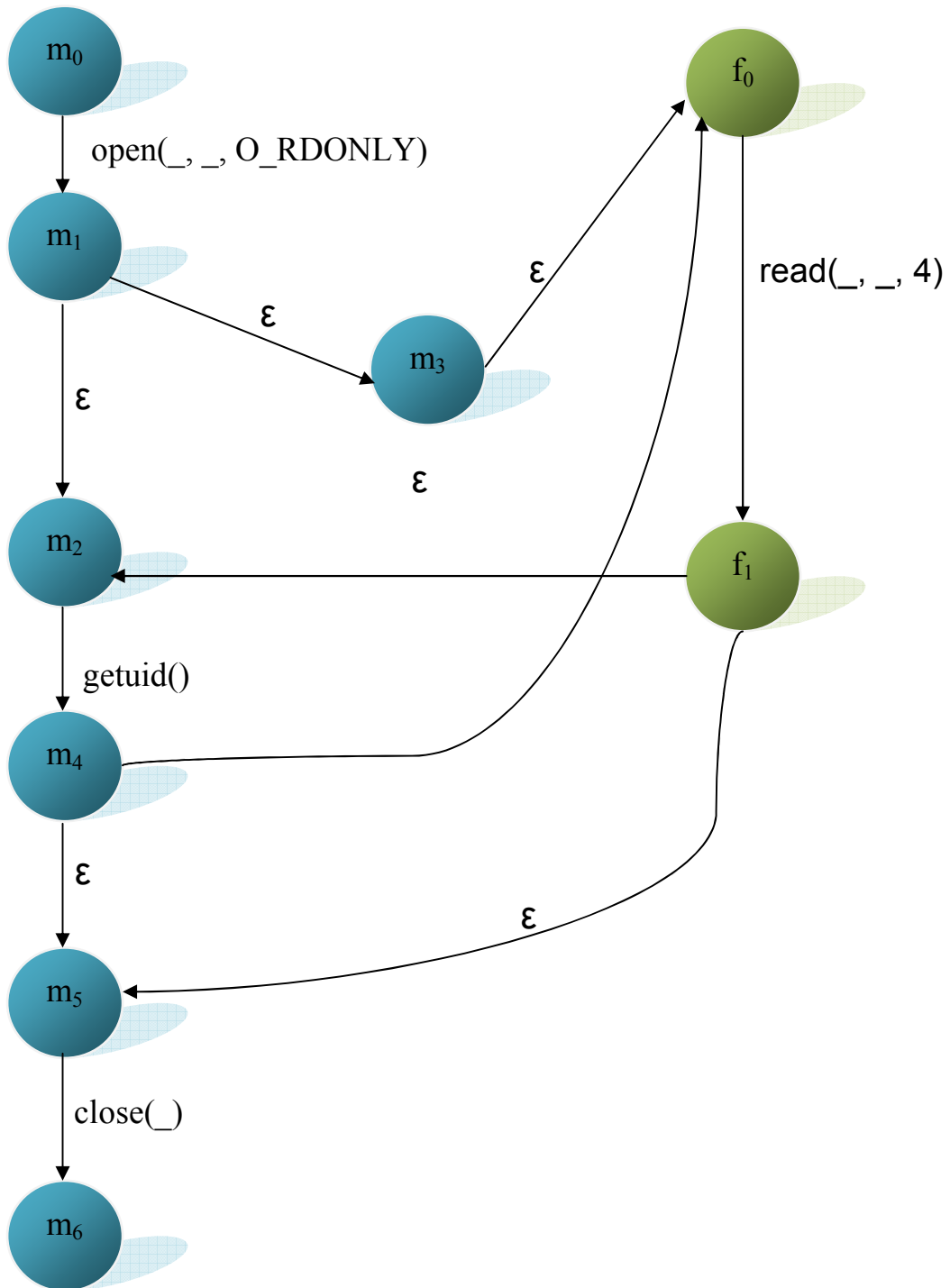
```
void foo(int fd, char *buf)
{
    read(fd, buf, 4);
}

int main(int argc, char **argv)
{
    int x;

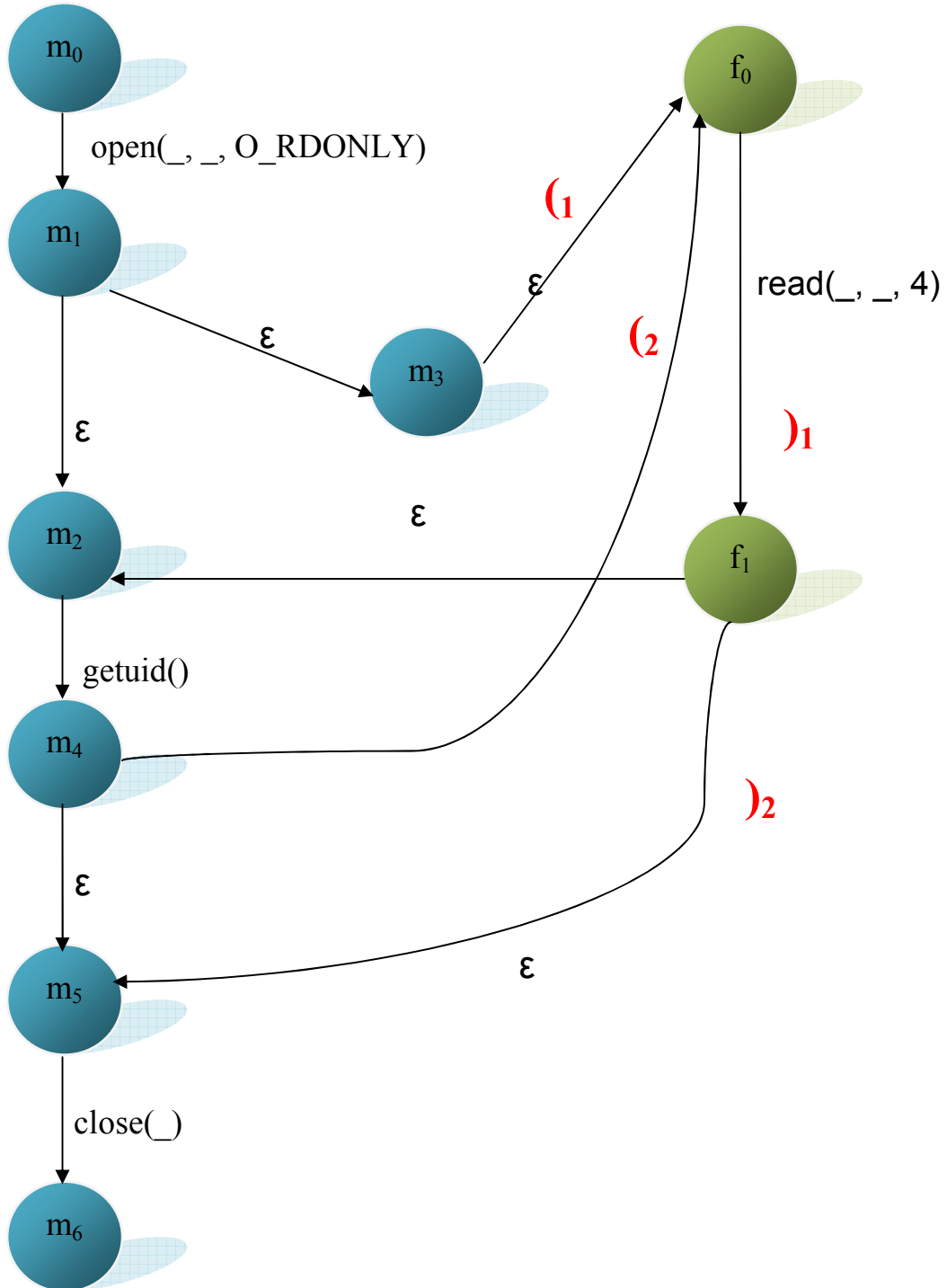
    fd = open(argv[1], O_RDONLY);
    if ( ... )
        foo(fd, &x);

    getuid();
    foo(fd, &x);
    close(fd);
}
```

The corresponding control flow graph is given below:



As the above model is Context-Insensitive, we can go directly from f_1 to m_5 , without executing `getuid()`. Another problem of this model is that it suffers from loops ($m_4 \rightarrow f_0 \rightarrow f_1 \rightarrow m_2 \rightarrow m_4$).



The next most powerful tool after **Finite State Automata** is the **Push Down Automata**.

Look at the red parenthesis in the above diagram, The basic rule says that the value on top of the stack should match with the operator currently being viewed, i.e. all the parenthesis should match.

Checking the arguments helps in reducing the overhead since it removes the non-determinism. (The branching factor is reduced).

Methods to generate models for IDS (cont.)

3. Efficient Context-sensitive Intrusion Detection

- ✿ Tracks values flowing through programs
- ✿ Reviews influence of system calls' return values on subsequent execution
- ✿ To improve performance, application is rewritten to record its path and report it to IDS
- ✿ It works on binaries

What could you accomplish as an attacker if the IDS does not read the content of I/O?