

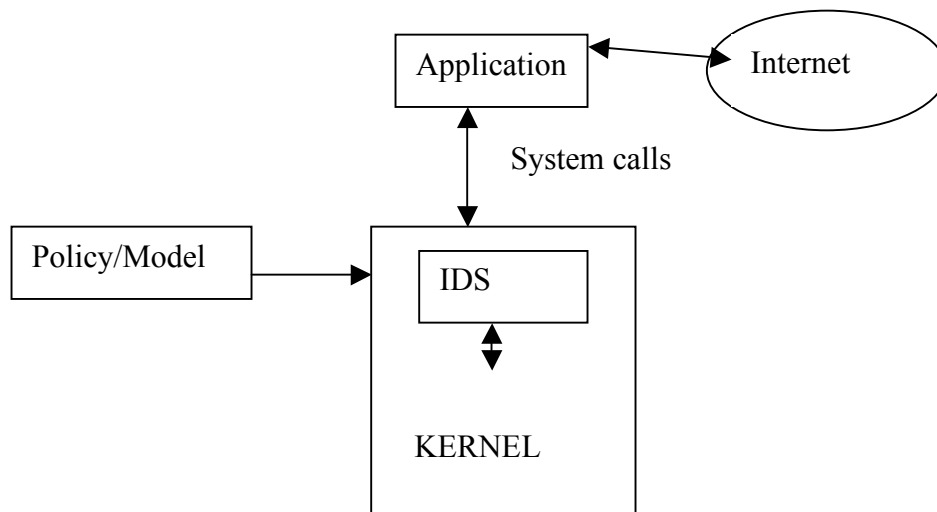
Intrusion Detection Systems/Sandboxing-Host-based IDS vs Network IDS

- Network IDS - Internet -> IDS -> Internal Network

- Packet analysis, string matching,

- Host-based IDS – monitor system call to detect weird behavior

- Only care about externally visible data from application, most if not all externally visible data is modified by system calls.



Policy – generic definition of allowed operation

- Applicable to untrusted code

Model – abstracted model of a specific apps behavior

- Applicable to trusted code

Two Metrics

- False Positives

- False Negatives

Models

- Generating Models

- Static Analysis-CFG (Control flow graph) => NFA (Non-deterministic finite automata)

- Dynamic Analysis- log real traces

Trace-based Intrusion Detection

-Levenstein distance?

-n-gram model

-problem: false positives

-Advantage: can capture site-specific configuration

T1: open, read, write, close

T2: open, read, read, close

T3: open, read, exec

3-gram model

(open, read, write)

(read, write, close)

(open, read, read)

(read, read, close)

(open, read, exec)

Take 2 most recent system calls, and current system call, and see if those two plus the current are in the model.

When using an n-gram model, there is a list of 1st system calls that are allowed for the program, then 2-grams, 3-grams, ...up to n-grams. When n is reached, the program just uses the n sized window on system calls.

2-gram vs 3-gram for previous example:

(open, read)

(read, write)

(write, close)

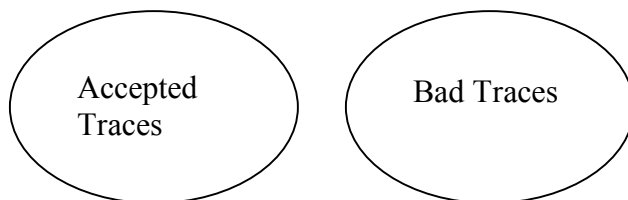
(read, read)

(read, close)

(read, exec)

(read, read) allows you to run (read,read) unlimited times, but the 3-gram model will not allow any such repetition so we gain more precision by using a higher n.

Assumption:

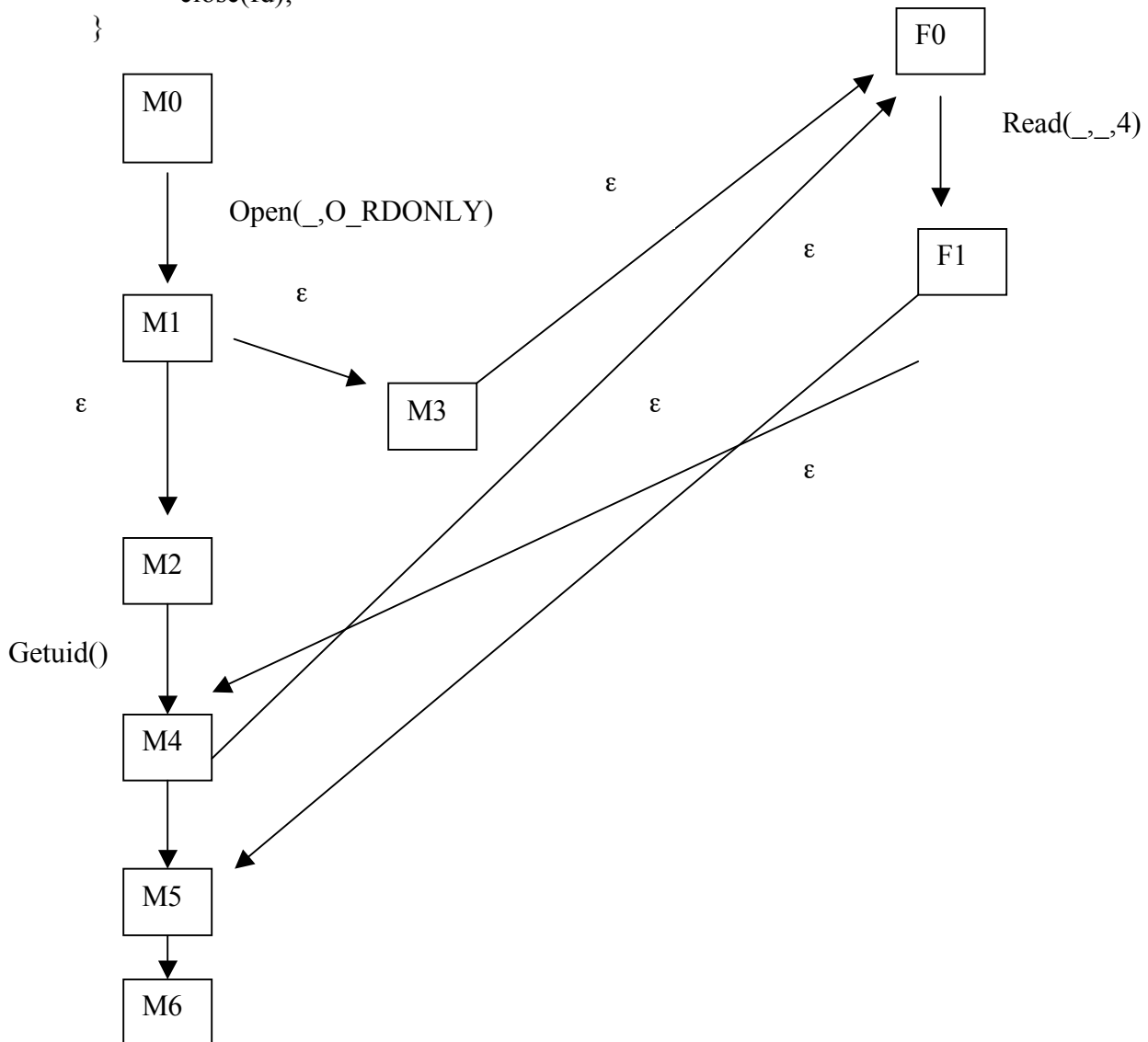


Mimicry attack:

If allowed INTERSECT bad != {} then pick a trace in intersection.

Model Construction

```
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(fd>0)
    {
        foo(fd,&x);
    }
    getuid();
    foo(fd,&x);
    close(fd);
}
```



Efficient Context-Sensitive Intrusion Detection(paper)

- Tracks values flowing through program
- Tracked influence of syscall return values on subsequent execution
- To improve performance, rewrote code to remove non-determinism. App recorded its path and reported it to IDS.
- worked on binaries