Intrusion Detection
- Host-based Intrusion-detection
  - System call monitoring HBIDS
- Goal: Prevent damage from attack.
- Key Idea
  - A program under attack behaves differently.
- Key Idea
  - Attacker must make system call to do real damage.

Two questions:
- What does model look like?
- How do we build it?

Idea 1.
Model = set of system call w/app source code.
- too coarse.
- alternative: run program & record syscalls it makes.

Static Models
- harder to build
+ no false positives

Dynamic Models
+ easy
- false positives
⇒ just issue warnings.
N-gram models

\[ \text{n-gram} = \text{sequence of n items} \]  
(Syscalls)

Ex) Suppose application performs

open, read, read, close, open, write, close

2-gram model = \{ (open, read), (read, read), (read, close), 
(close, open), (open, write), (write, close) \} 

Usually: 6-gram.

At runtime

Monitors stores last n-1 syscalls, (S_1...S_{n-1})
when app makes syscall S_n, check whether  
(S_1, S_2..., S_n) \in M, store (S_2, S_3..., S_n)

FGSA model

- typical built statically
- just like model checking

\[ F \left( \left( fd = \text{open}(...) \land < 0 \right) \right) \]

exct();

while (read(...)  
gettimeofday();

close(fd);

open

read

read

gettimeofday

read

gettimeofday

close
Convert P into Mp
Mp accepts \( L(M_p) \)
Fact: all legitimate executions of P should produce a sequence S of system calls s.t. \( S \subseteq S_E(L(M_p)) \)

```c
void foo(void)
{
    read(...);
}

void main(...)
{
    open(...);
    foo();
    get_setuid(-1);
    foo();
    exec(...);
}
```

0x1)

```c
void foo(void)
{
    if (...) foo();
    read(...);
}
```

0x2) Same `main(...)` above

```c
void foo(void)
{
    ...
    foo();
    read(...);
}
```
Efficient Context Sensitive Intrusion Detection
- Modify app to inform monitor of function calls.

```c
void foo(void)
{
    notify_call(foo);
    if (...) foo();
    read(...);
    notify_return(foo);
}
```

Diagram:
```
E/fooz
\|--
   \|--
      \|-- foo
         \|--
             \|-- foo2
                 \|--
                      \|-- foo3
                           \|-- foo4
```

```
read
```

(continued)