

# COMBINING STATIC ANALYSIS AND DYNAMIC LEARNING TO BUILD ACCURATE INTRUSION DETECTION MODELS

Zhen Liu, Susan M. Bridges, and Ray Vaughn  
{zliu, bridges, vaughn}@cse.msstate.edu

Supported by the National Science Foundation  
Cyber trust award No. SCI0430354-04090852

Department of Computer Science and Engineering  
Mississippi State University

# Outline

---

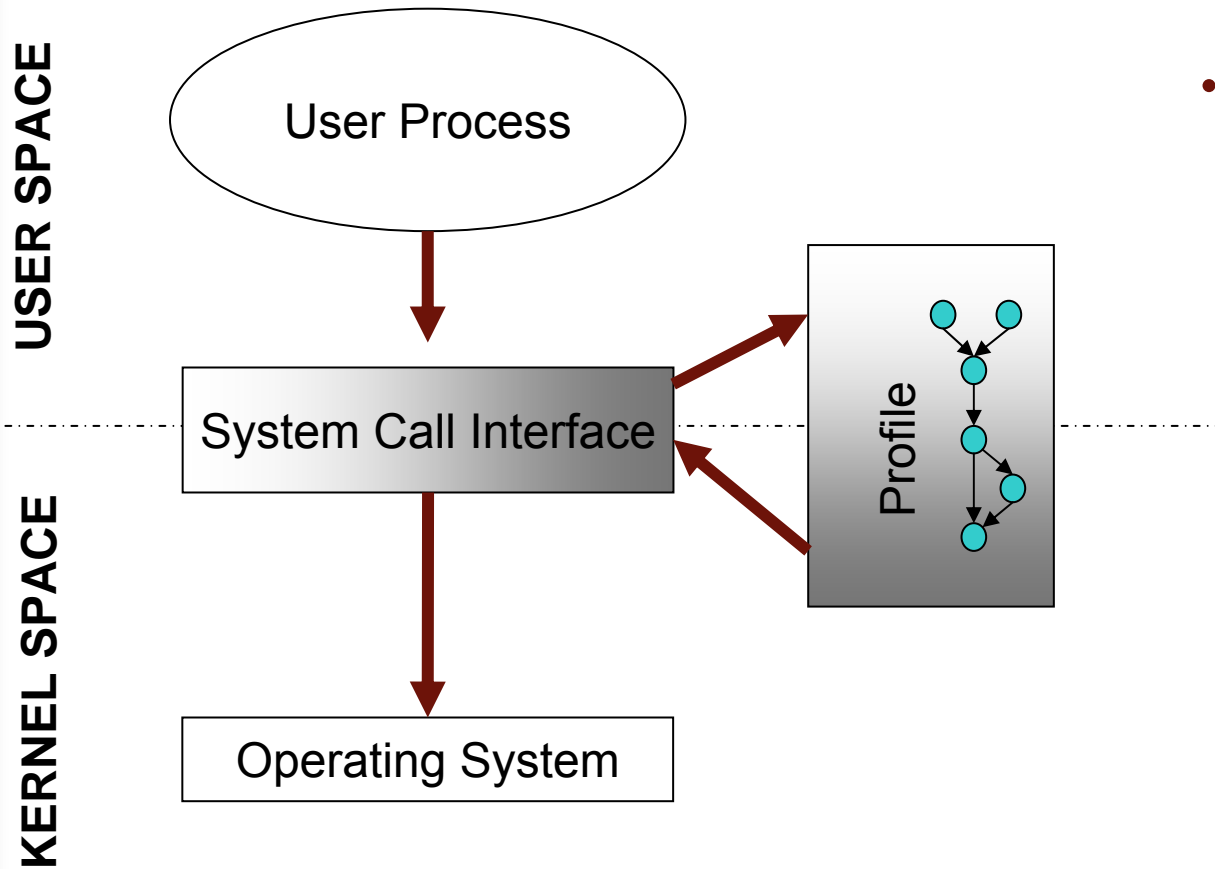
- Introduction
  - Scope
  - Objective
- Research Approach
  - Combination of static analysis and dynamic learning
  - HPDA (Hybrid Push Down Automata) model
  - Properties and advantages of HPDA model
- Experimental Results
- Conclusion and future work

# Host/program based intrusion detection and prevention

---

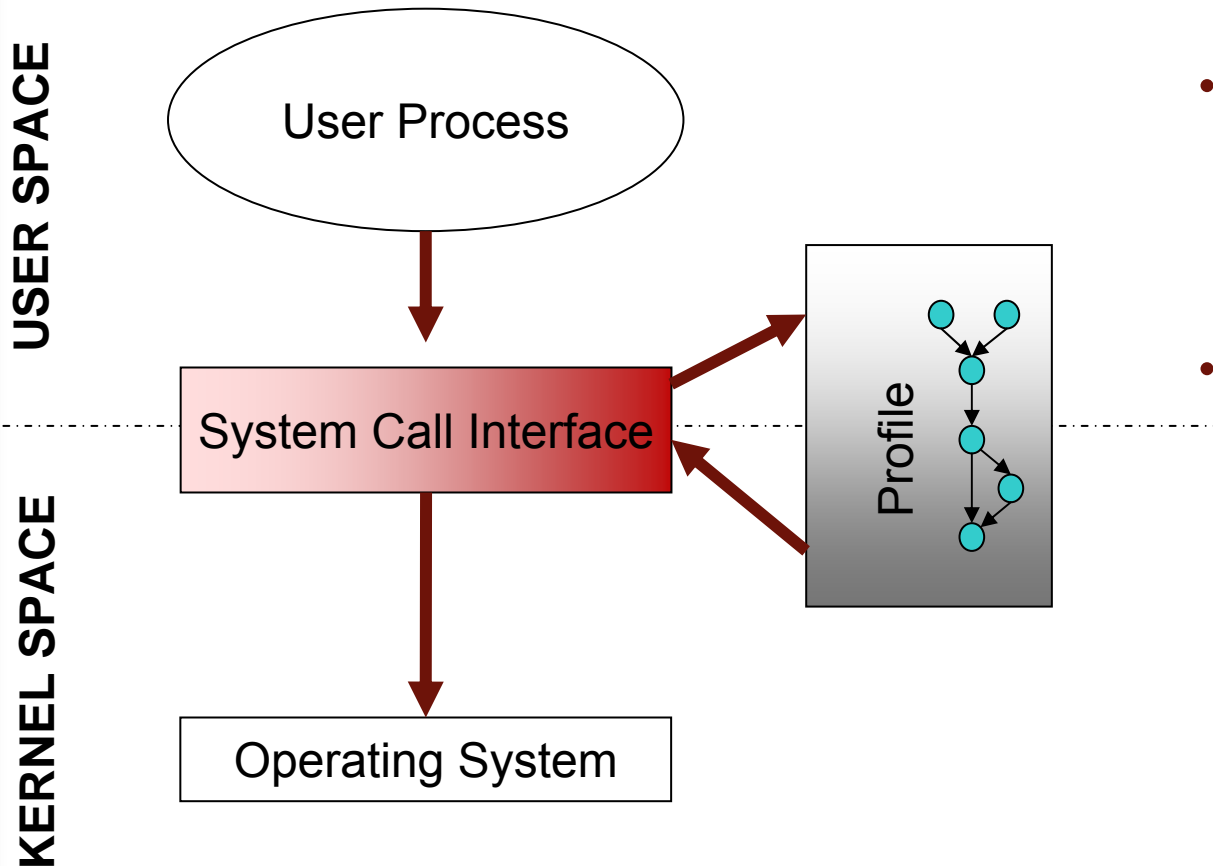
- Defense in depth
- Method used by attackers
  - Exploiting vulnerabilities in applications
  - Buffer overflow, format string, etc.
- Defense before the malicious operation take effect

# Intrusion detection and prevention via program behavior monitoring



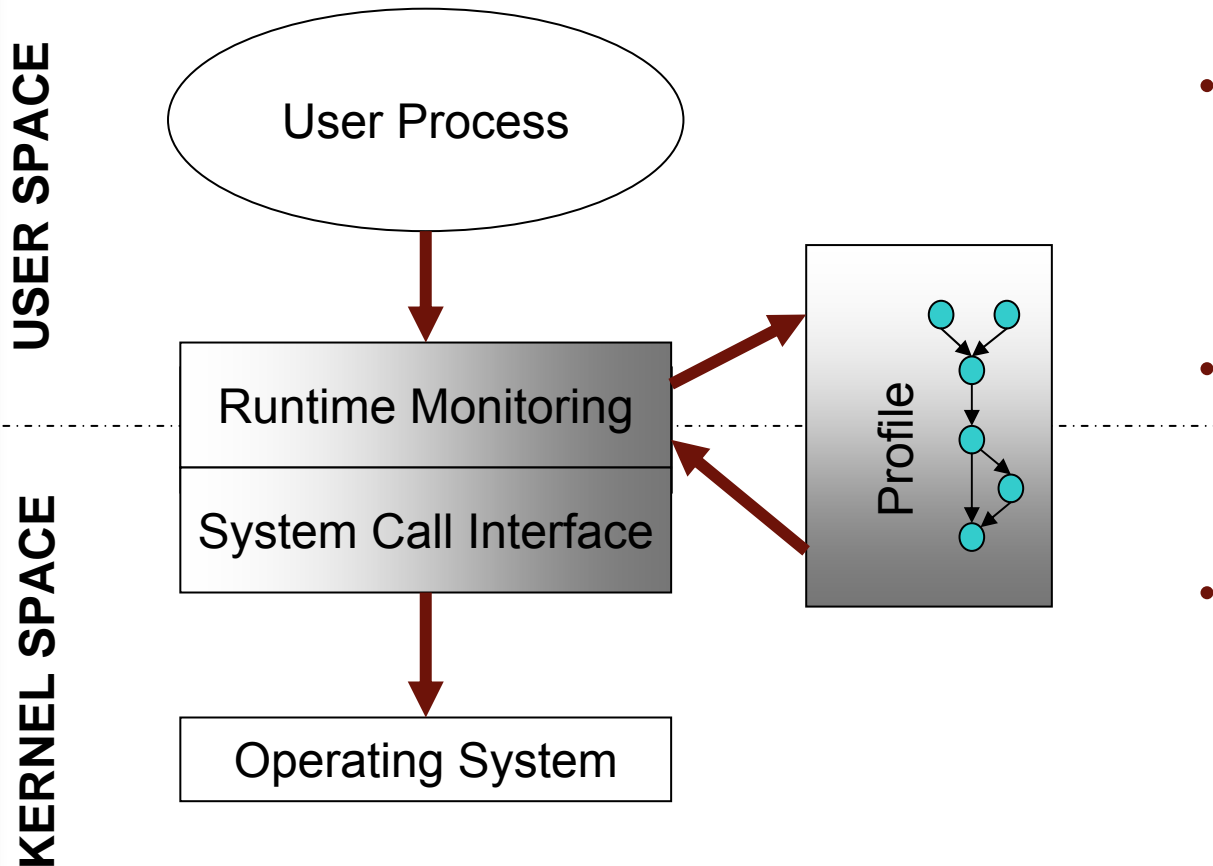
- User processes make operating system requests via system call interface

# Intrusion detection and prevention via program behavior monitoring



- User processes make operating system requests via system call interface
- Malicious operations are conducted via these requests

# Intrusion detection and prevention via program behavior monitoring



- User processes make operating system requests via system call interface
- Malicious operations are conducted via these requests
- Monitoring detect and prevent malicious requests before their execution

# Normal program behavior modeling based on system-call monitoring



Where A, B, C, etc. stand for system calls (*exit, open, read, .....*)

```
main()
{
  fd = open("foo", O_RDONLY);
  if(!fd) {
    write(1, "Fail to open file", 255, 1);
    exit(0);
  }
  read(fd, buf, 255,1);
  close(fd);
  exit(0);
}
```

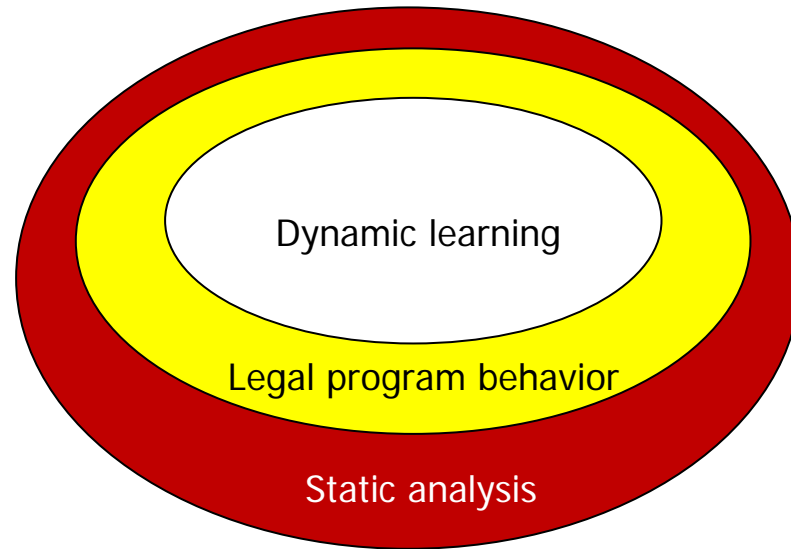
Possible sequences:  
open, write, exit  
open, read, close, exit

# Methods to acquire a model of normal program behavior

---

- Dynamic learning
  - Execute the application in normal condition
  - Collect audit data
  - Extract the model from audit data by training the learning algorithm
  - FSA and PDA from source code (Wagner et al. 01), Dyck from binary executable (Giffin et al. 02, 04), VtStatic from binary executable (Feng et al. 04)
- Static analysis
  - Extract the control flow from binary executable or source code
  - Use control flow to build the model
  - Stide (Forrest et al. 96), VtPath (Feng et al. 03), Neural networks, hidden Markov models, rule learning systems, etc.

# Modeling program behavior



■ False Positives      ■ False negatives

- Dynamic learning
  - Builds the normal model by observing the runtime behavior of the program. (Forrest et al. 96, Lee et al. 98, Ghosh et al. 99, Sekar et al. 01, Feng et al. 03)
  - Generates **false positives**
- Static analysis
  - Builds the normal model by analyzing the binary executable or source code of the program. (Wagner et al. 01, Giffin et al. 02, 04, Feng et al. 04)
  - Is susceptible to **false negatives**

# Objective

---

- Construct a model of program behavior that matches the behavior defined in its executable as close as possible
  - accepts all legal sequences of system calls that can be generated by a program
  - accepts no illegal sequences

Note: We assume that the design of a program is correct and program should run as its designer's intention

# Approach

---

- Combine static analysis and dynamic learning to build accurate intrusion detection models
- Hybrid Push Down Automata (HPDA) model
  - Facilitates the combination approach
  - Is a compact and general representation
  - Captures context of function calls
  - Is efficient to operate
    - No ambiguity
    - No need to maintain an extra stack

# HPDA construction

---

- Static analysis alone
- Dynamic learning alone
- Combination of static analysis and dynamic learning
  - Static analysis used to acquire base model
  - Dynamic learning used to supplement the base model with behavior defined at runtime

# Finite state automata (FSA) of Wagner et al.

```
int h(int x)
```

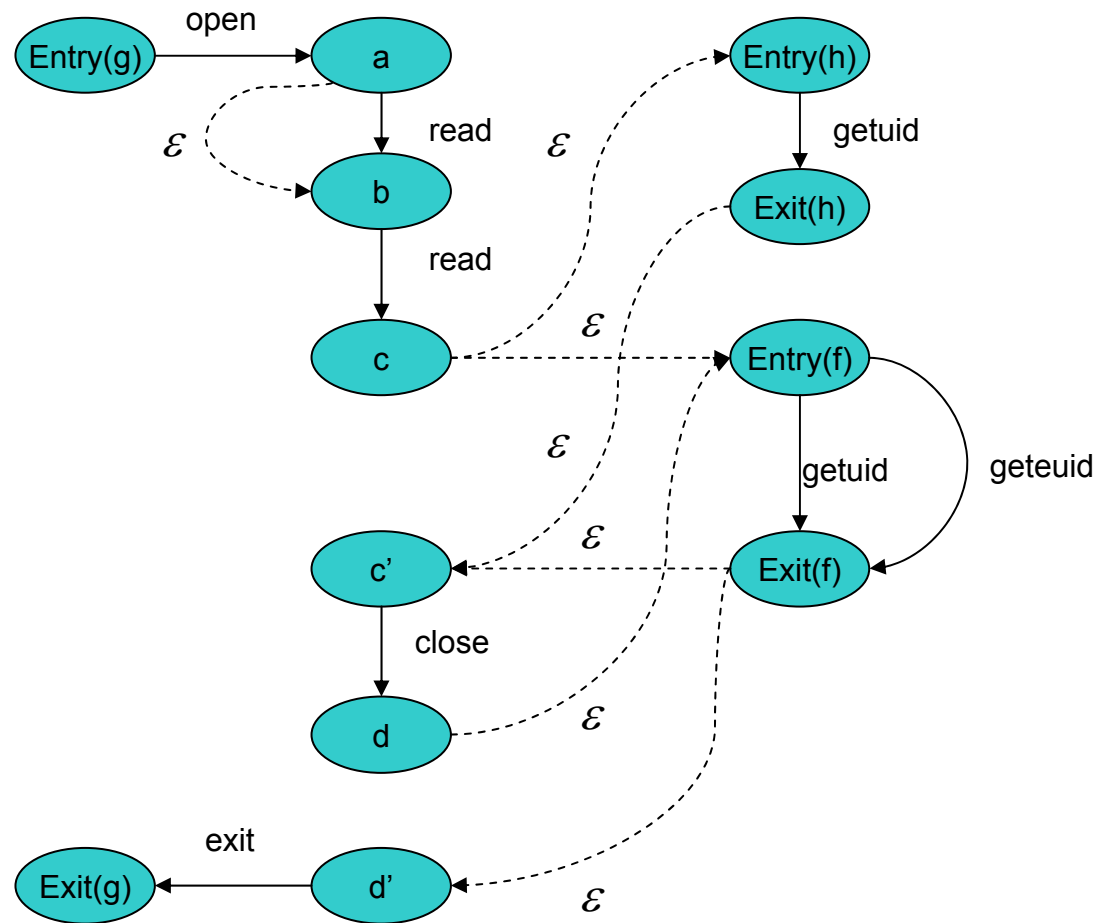
```
{
1  id = getuid();
2  id += x;
3  return id;
}
```

```
int f(int x)
```

```
{
4  if(x) id = getuid();
5  else getuid();
6  return id;
}
```

```
g()
```

```
{
7  fd = open("foo", O_RDONLY);
8  if(fd)
9      read(fd, buf, 255, 1);
10 read(fd, buf, 255,1);
11 if(fd) h(0);
12 else f(0);
13 close(fd);
14 f(1);
15 exit(0);
}
```



# Impossible path problem

```
int h(int x)
```

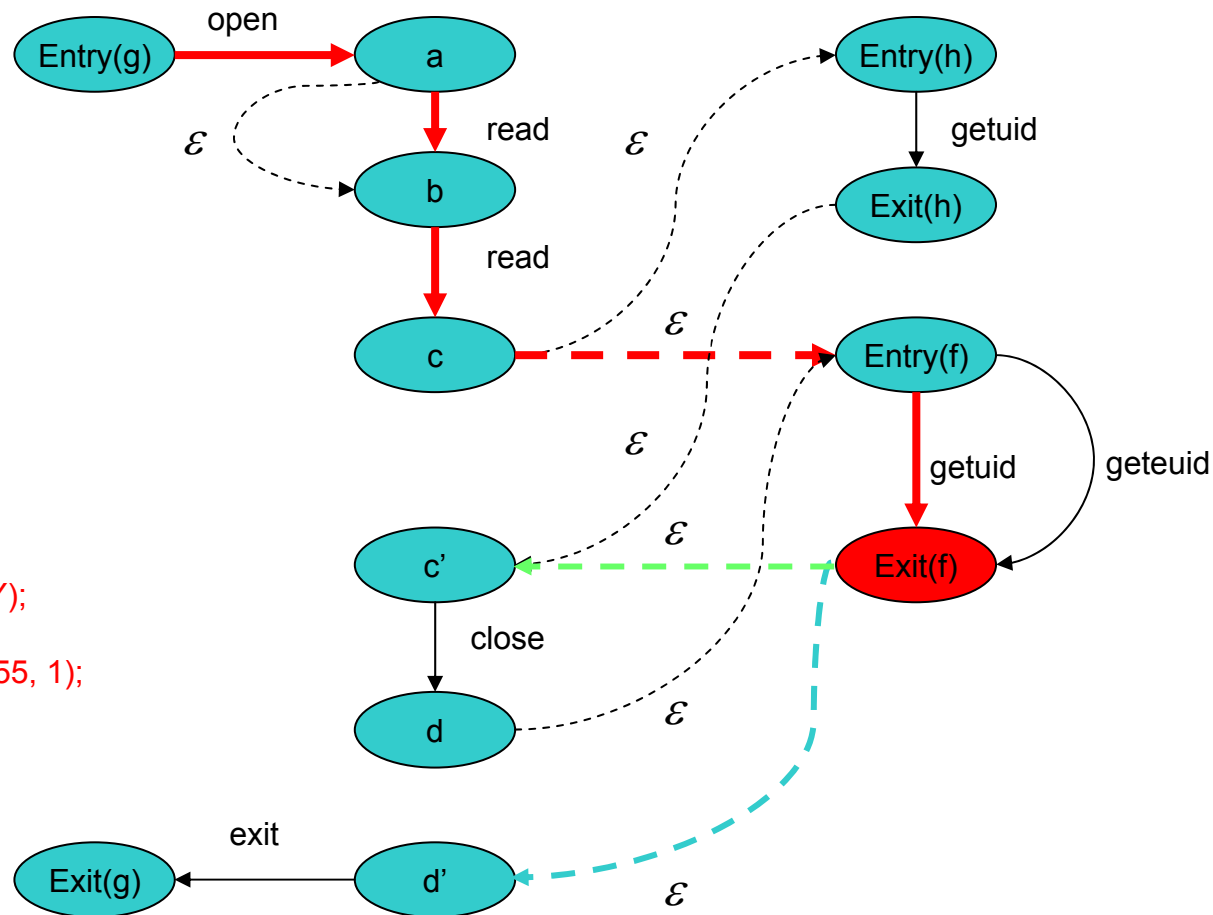
```
{
1   id = getuid();
2   id += x;
3   return id;
}
```

```
int f(int x)
```

```
{
4   if(x) id = getuid();
5   else getuid();
6   return id;
}
```

```
g()
```

```
{
7   fd = open("foo", O_RDONLY);
8   if(fd
9       read(fd, buf, 255, 1);
10  read(fd, buf, 255, 1);
11  if(fd) h(0);
12  else f(0);
13  close(fd);
14  f(1);
15  exit(0);
}
```



# Impossible path problem

```
int h(int x)
```

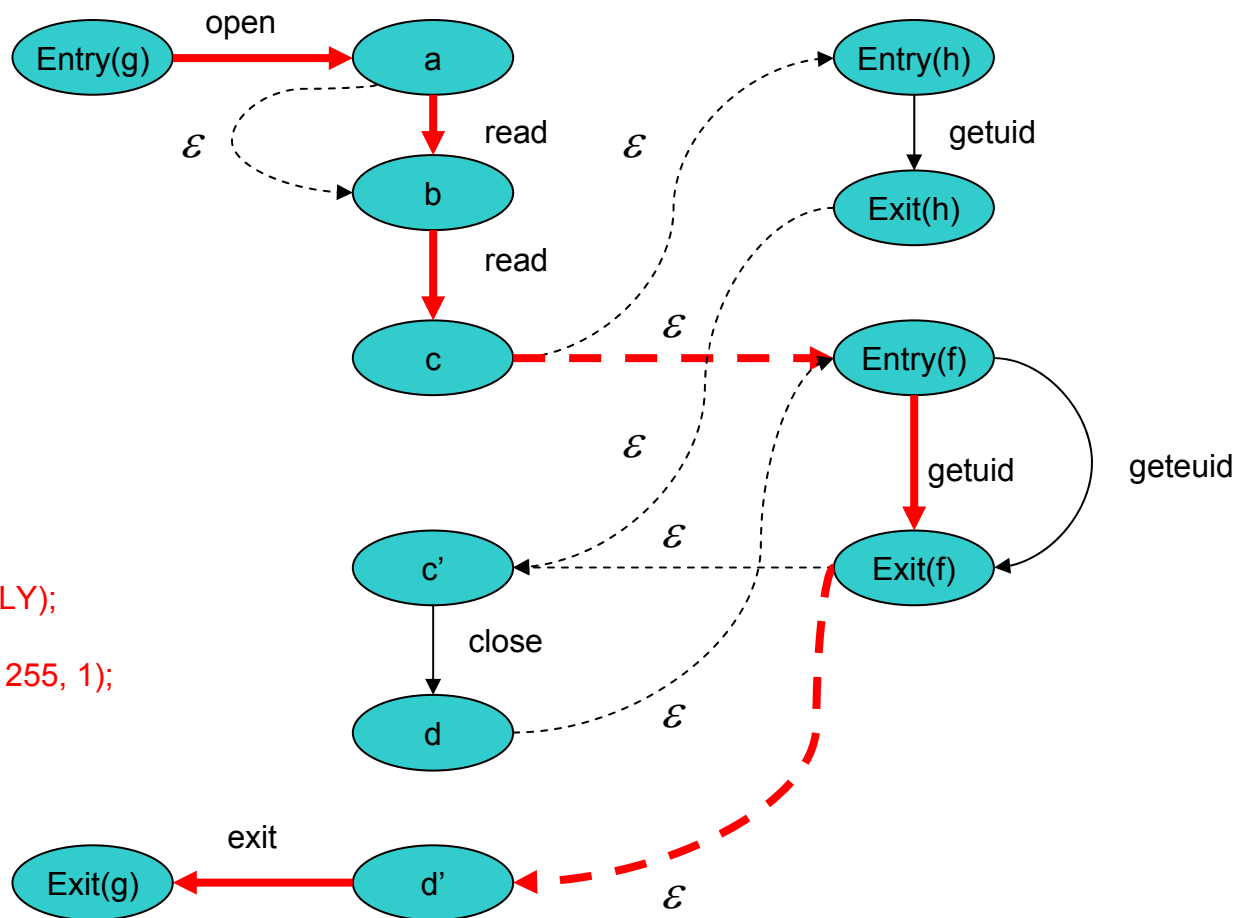
```
{
1   id = getuid();
2   id += x;
3   return id;
}
```

```
int f(int x)
```

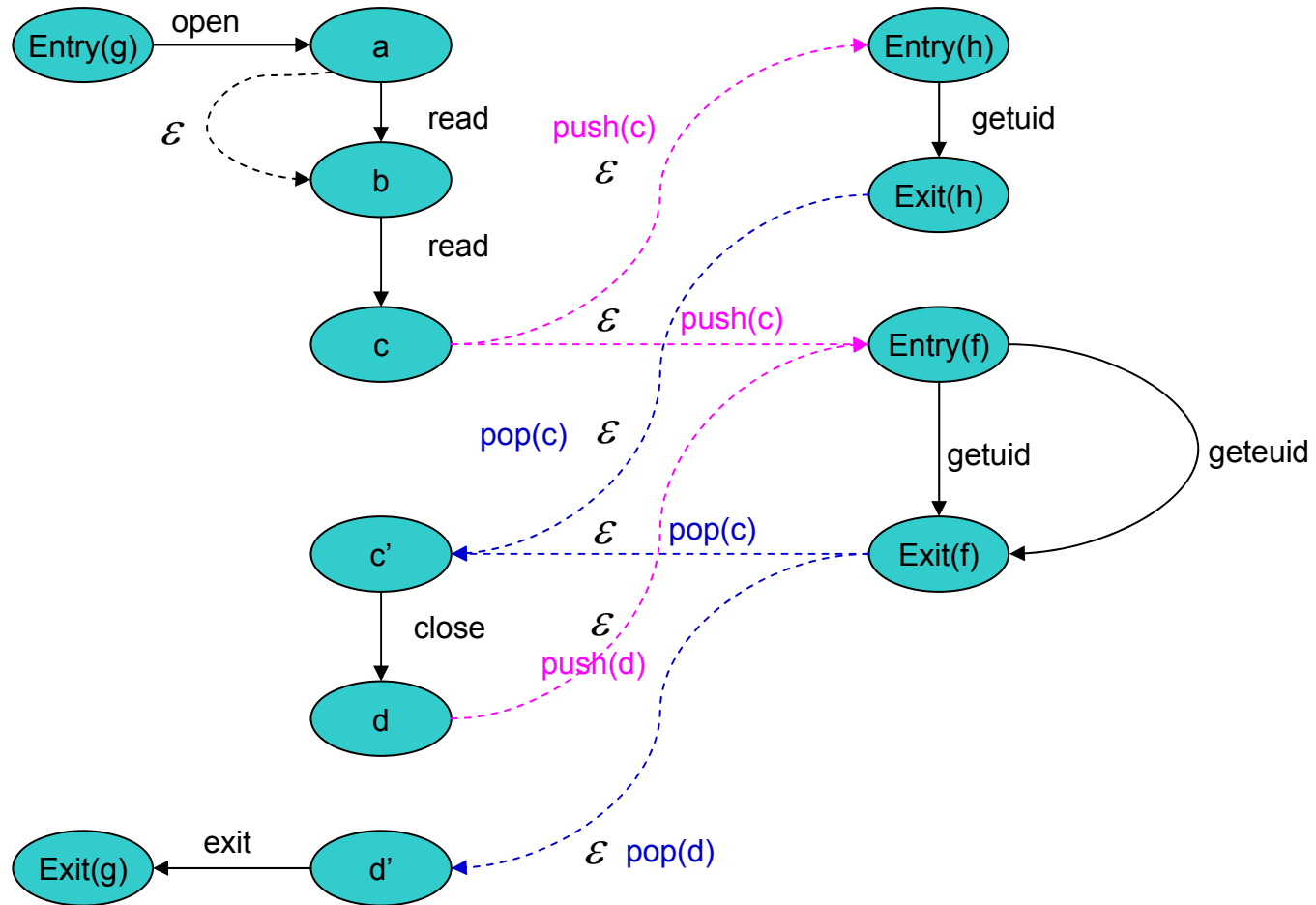
```
{
4   if(x) id = getuid();
5   else getuid();
6   return id;
}
```

```
g()
```

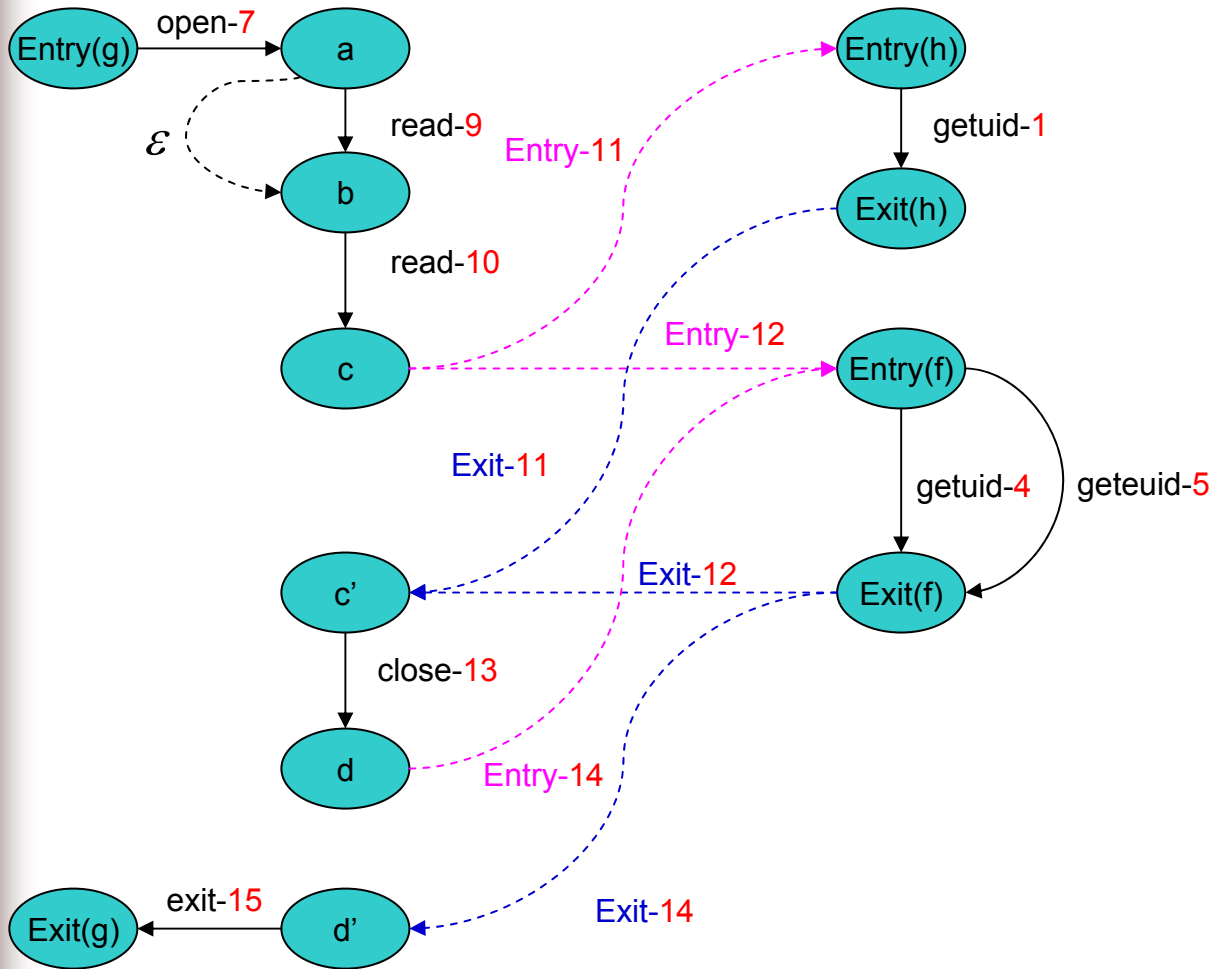
```
{
7   fd = open("foo", O_RDONLY);
8   if(fd)
9       read(fd, buf, 255, 1);
10  read(fd, buf, 255, 1);
11  if(fd) h(0);
12  else f(0);
13  close(fd);
14  f(1);
15  exit(0);
}
```



# Push down automata (PDA) of Wagner et al.



# Hybrid Push Down Automata (HPDA)



- As powerful as PDA
- More efficient to operate than PDA
- Extended input alphabet sets  $\Sigma$
- Types of transitions
  - Entry
  - Exit
  - System call
  - $\epsilon$

# Advantages of HPDA model

---

- Impossible path problem is not encountered
- Mimicry attacks (Wagner et al. 02) are much more difficult to implement
- Non-determinism is reduced
- Simulation of the model is efficient
- Construction by a combination approach is allowed

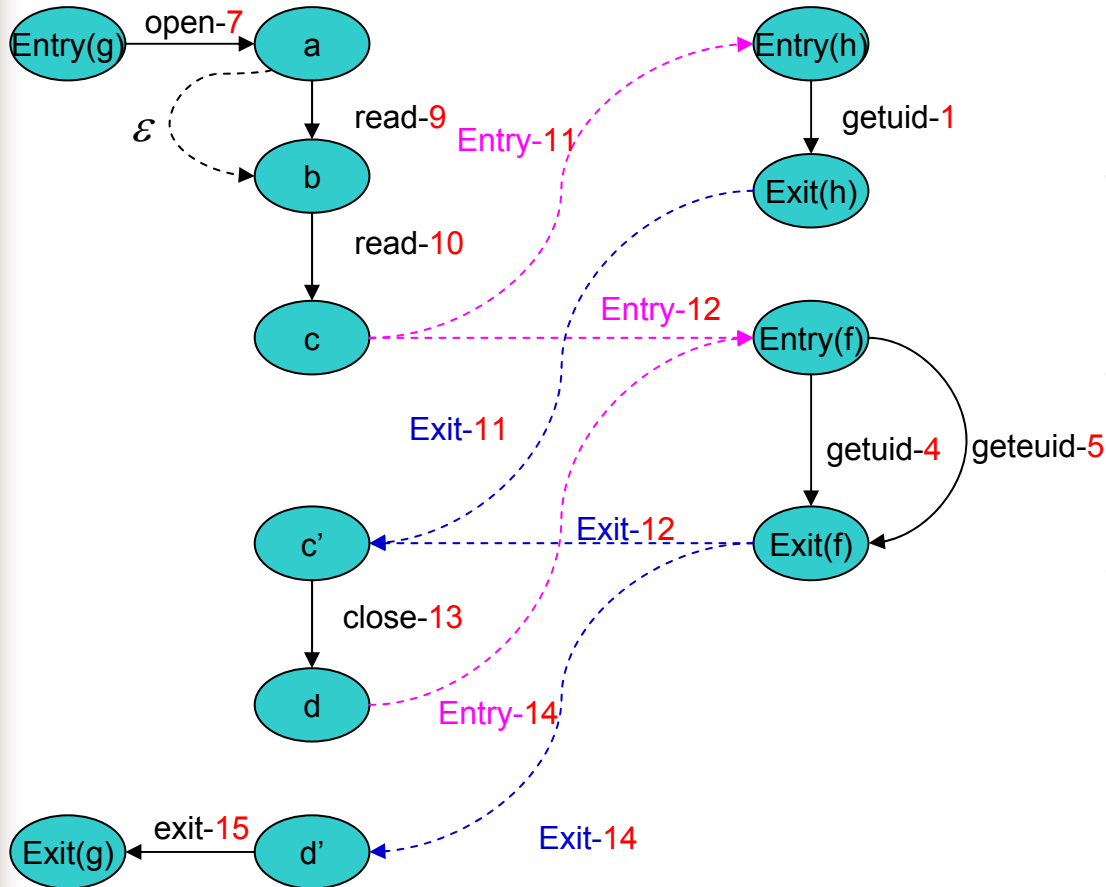


# Programming environments assumed by HPDA

---

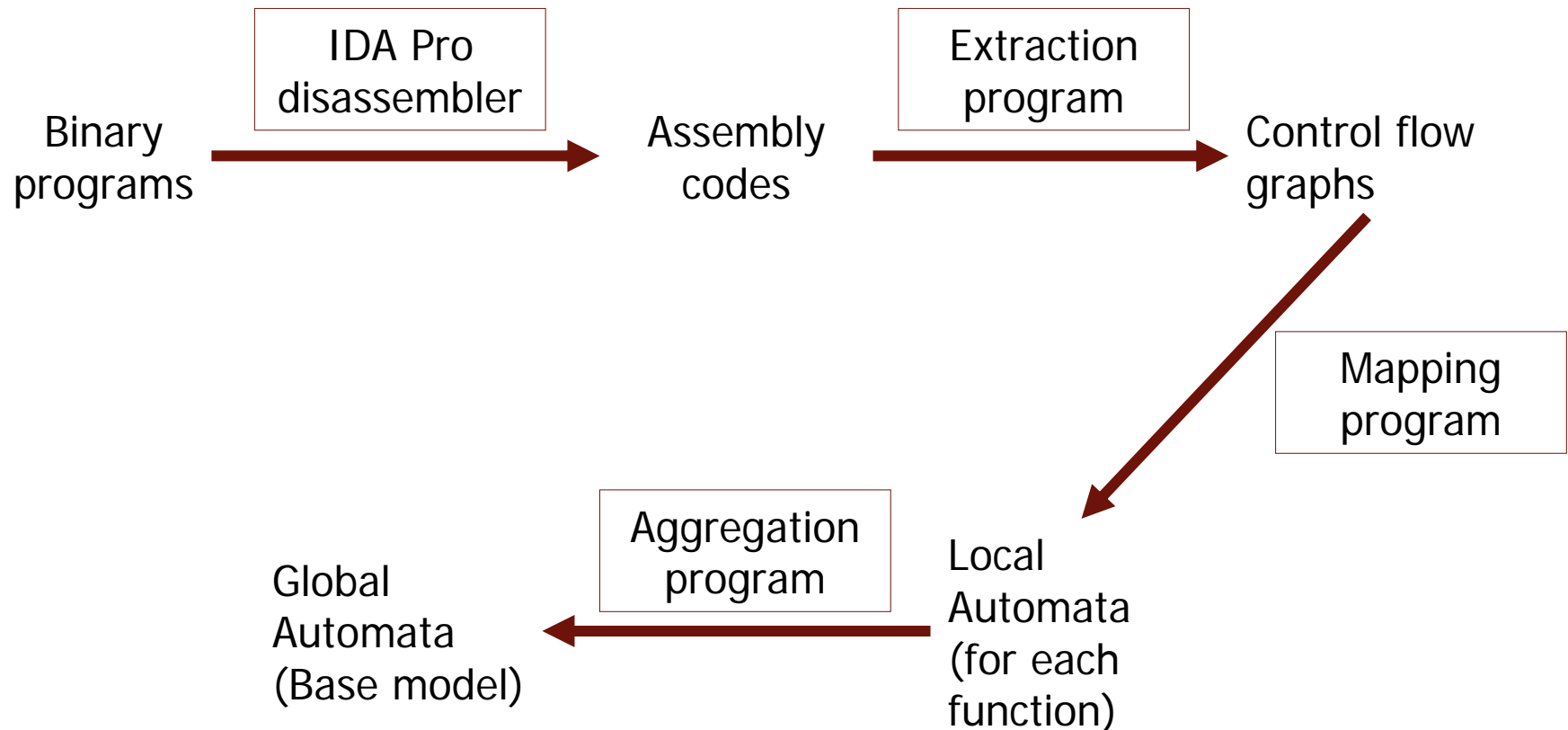
- Addresses uniquely identify instructions
- No overlay or self-modifying code
- Each address is a pair
  - relative address (in either the program executable or a dynamic library executable)
  - the identity of the executable

# Properties of HPDA



- Paired *Entry-Addr* and *Exit-Addr*
- Uniqueness of *Addr* in  $\Sigma$
- Uniqueness of symbols in model

# HPDA construction using static analysis of the executable



# Difficulties in static analysis

---

- Indirect calls (function pointer and virtual functions in C++)
- The *setjmp()* primitive and exception handlers
- Signals
- Dynamic libraries

# Dynamic learning of HPDA

---

- Adds transitions that cannot be learned in static analysis
- Adds transitions based on properties of HPDA
- Requires one pass through the training data

# Dynamic learning algorithm

---

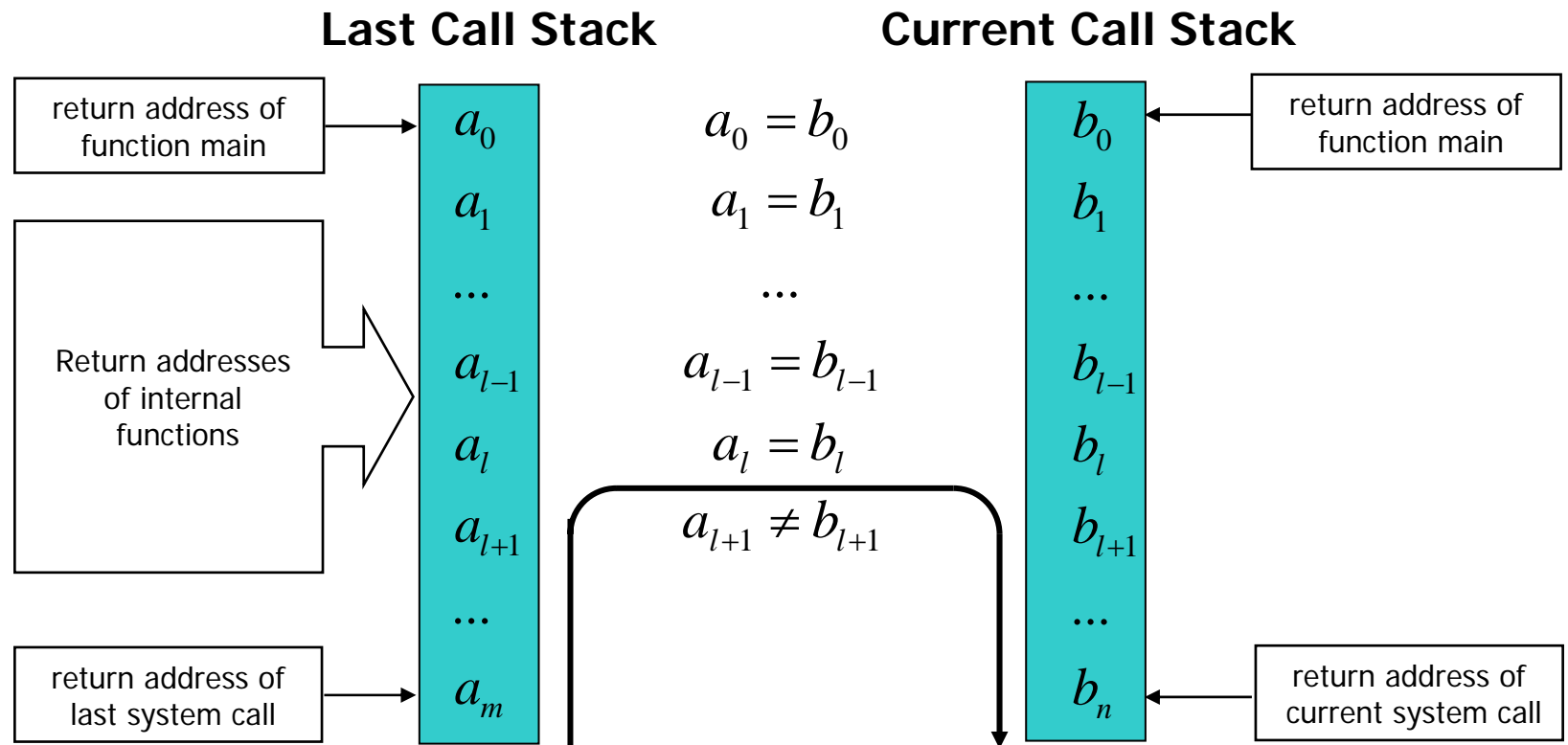
- Input sequence extraction
- Learning on the symbol sequence
  - No model update and move current state
  - Model update
    - Transitions addition
      - $\epsilon$  transition
      - normal transition
    - States combination

# Online detection using HPDA

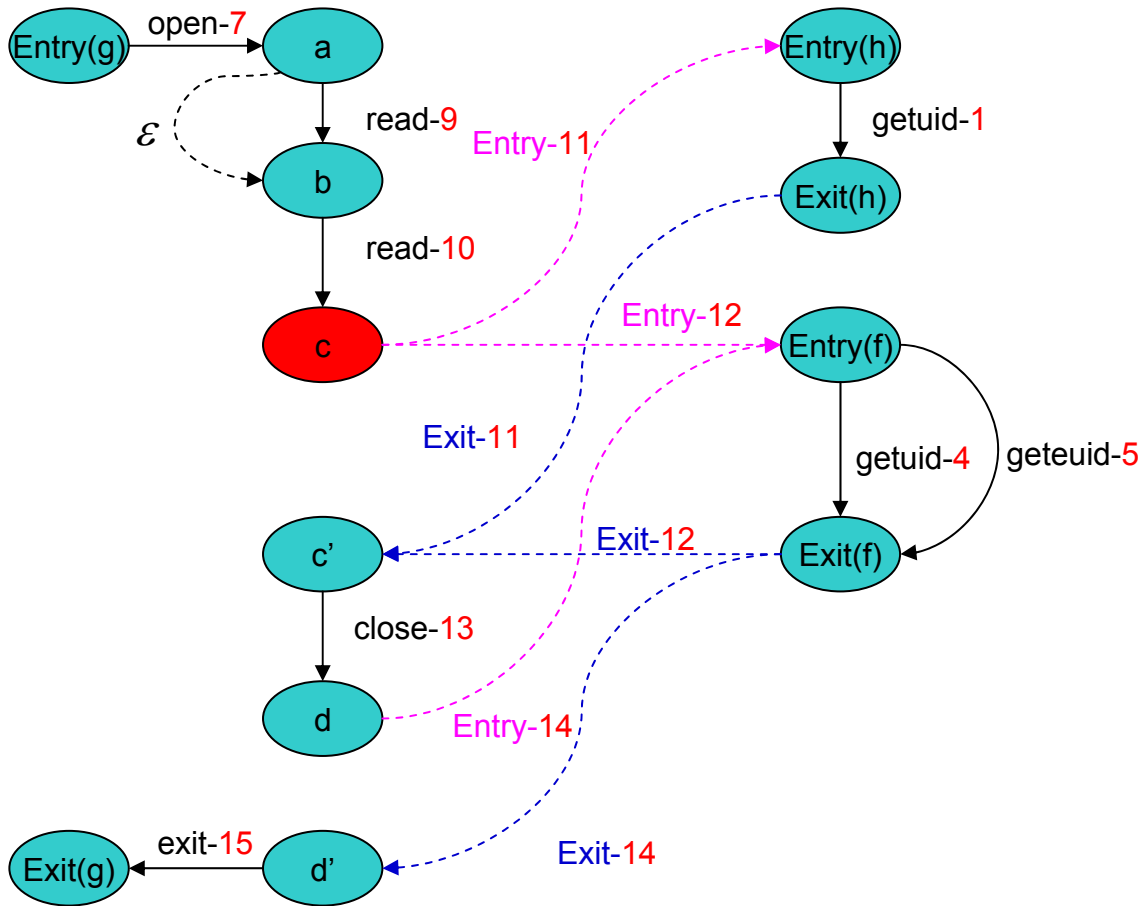
---

- Detection system is activated when the system call is invoked
- Input sequence since last system call is extracted (Feng et al. 2003)
- The input sequence is processed by the model
- Detection requires:
  - HPDA model
  - Current state
  - Last call stack

# Input sequence extraction



- Input symbol sequence:  
(Exit- $a_{m-1}$ , ..., Exit- $a_{l+1}$ , Entry- $b_{l+1}$ , ..., Entry- $b_{n-1}$ , SyscallID- $b_n$ )



- Current state: c

Last Call Stack

10
----

- Detection system is activated when system call *getuid* is invoked

Current Call Stack

12
4

- Input sequence

*Entry-12, getuid-4*

### A normal system call invocation

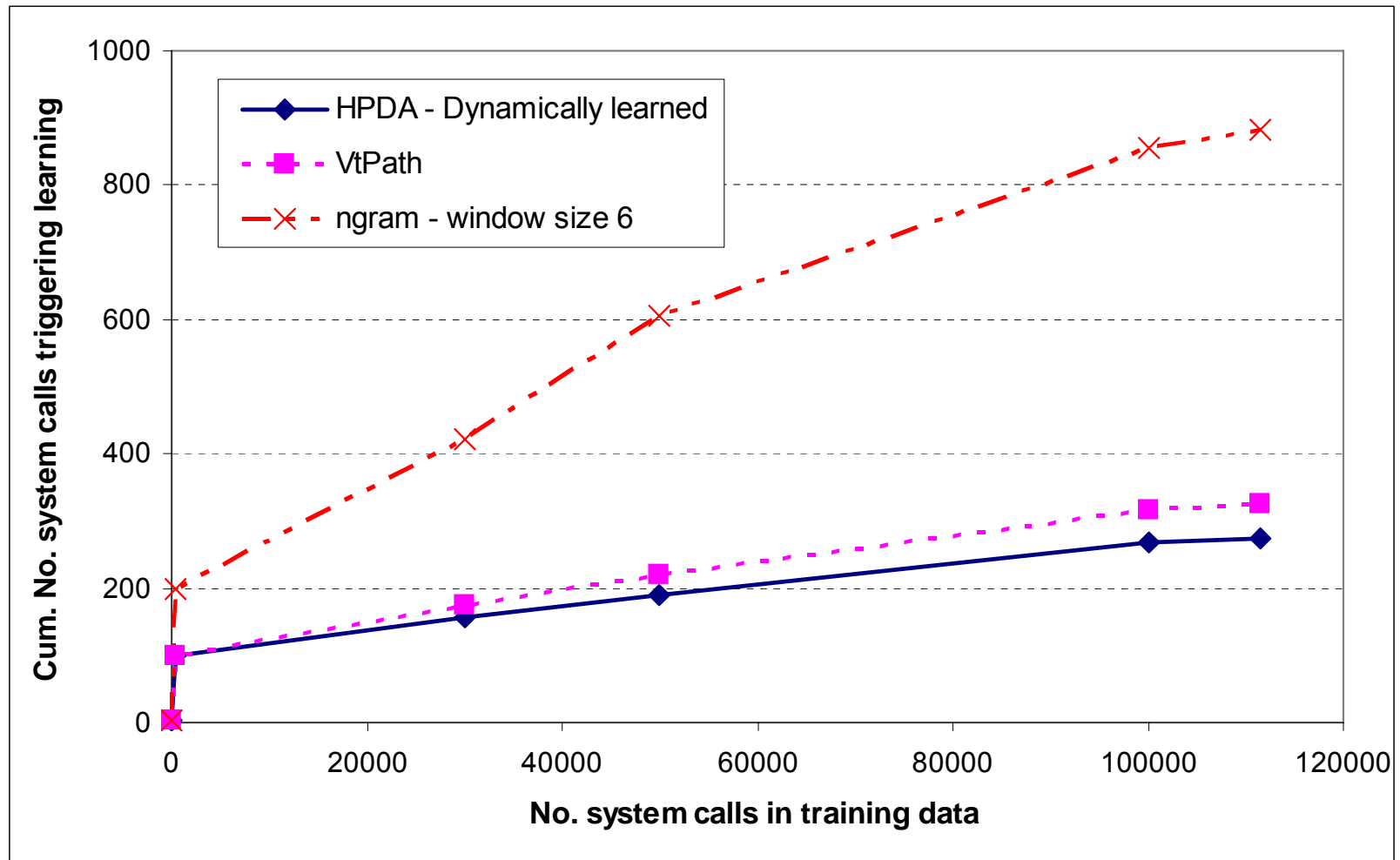
# Comparisons in experiments

---

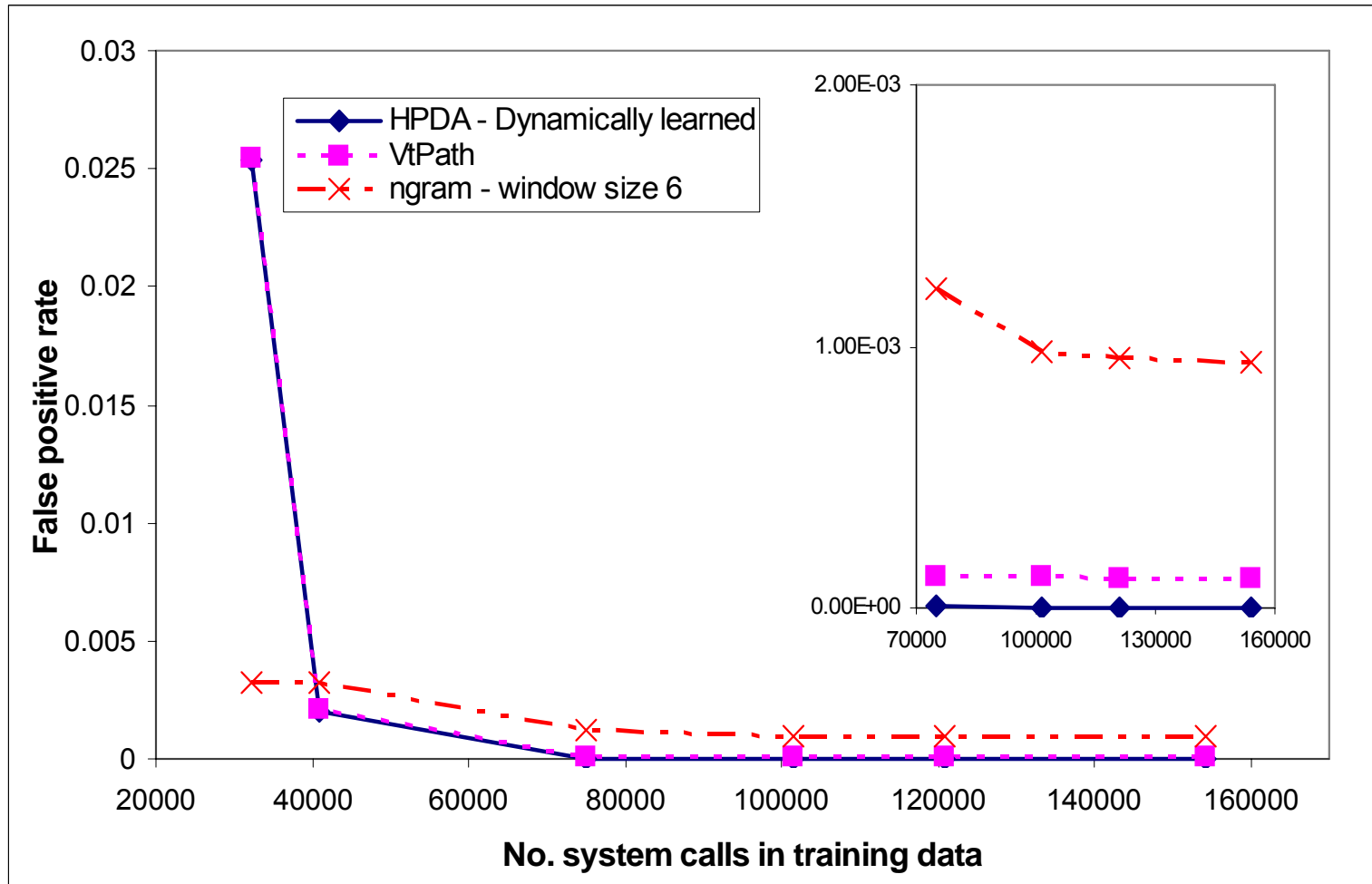
- Convergence speed - A measure of how fast a model can be learned as a function of the size of training data
- False positive rate

$$\text{False Positive Rate} = \frac{\text{Num of system calls where an alarm is raised}}{\text{Total num of system calls}}$$

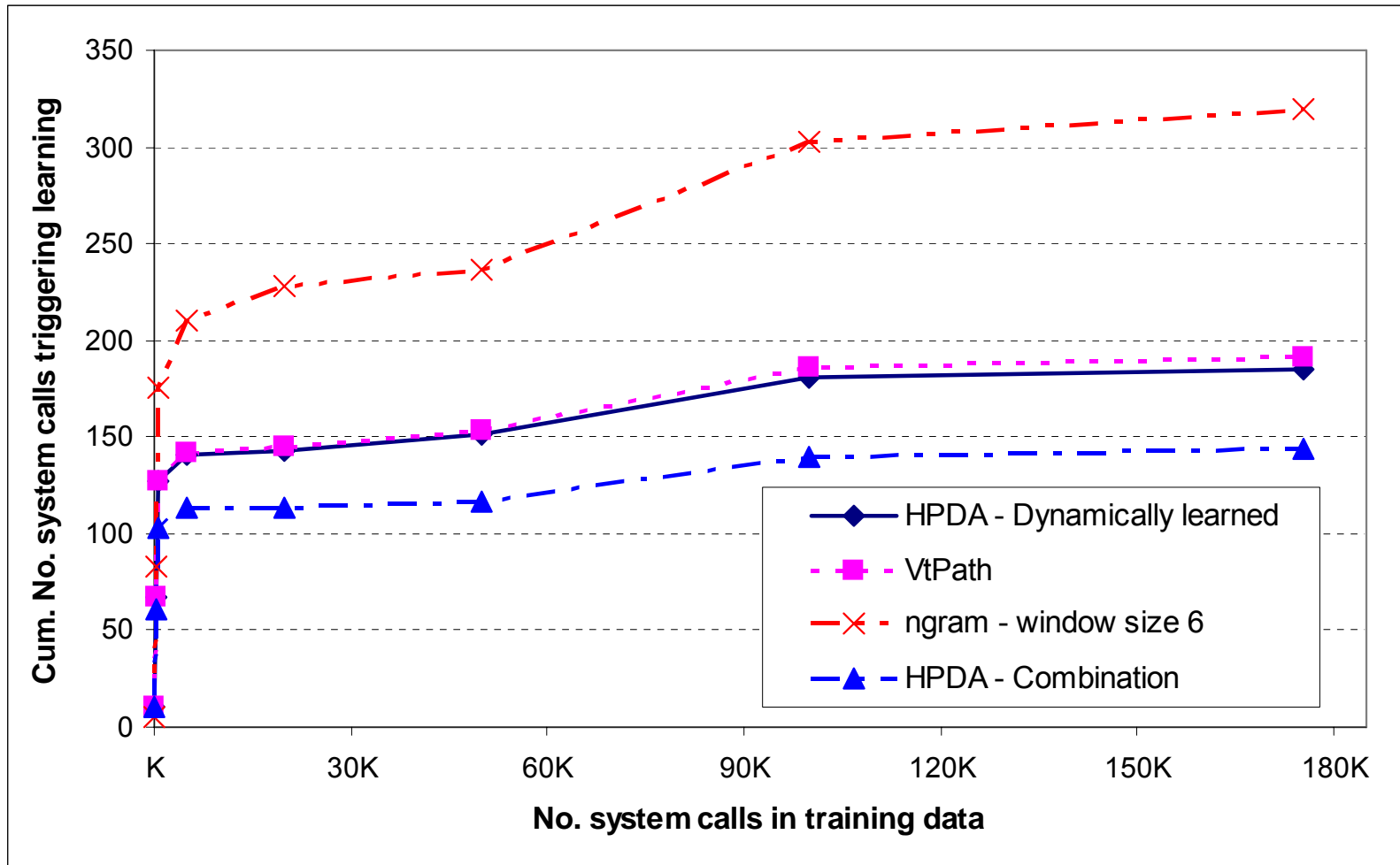
# Convergence speed for *apache*



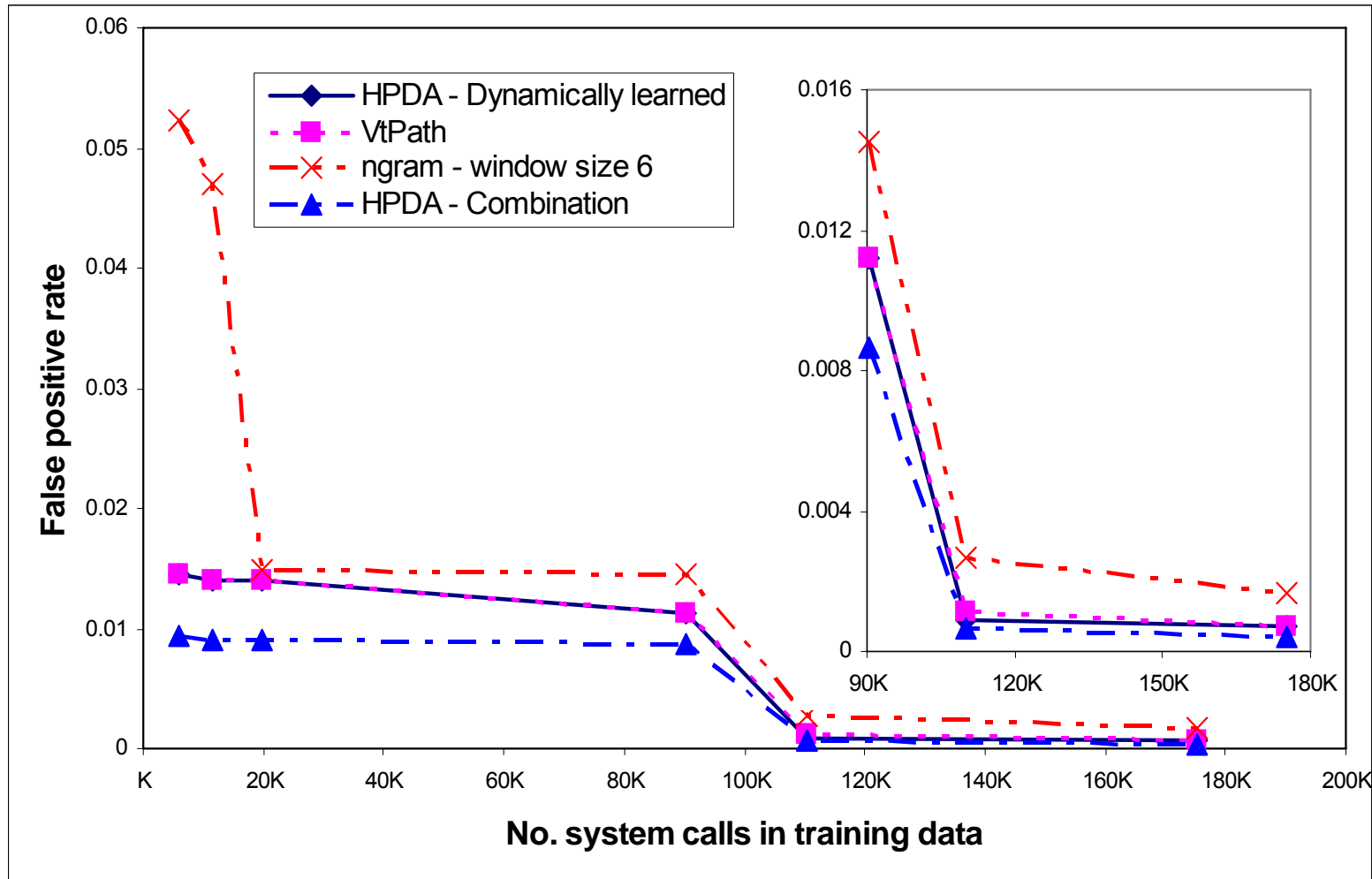
# False positive rates for *apache*



# Convergence speed for *vsftpd*



# False positive rates for *vsftpd*



# Conclusions

---

- The HPDA model is the first model that can be acquired by a combination of static analysis and dynamic learning
- Dynamic learning algorithm requires less training data than other dynamic learning approaches such as *VtPath* and *ngram*
- Combination approach requires less training data than dynamic learning alone
- Combination approach is less susceptible to false negatives than static analysis alone

# Future work

---

- Handle the dynamic libraries and combine the models for each execution units at runtime
- Implement detection system in kernel space
- Test our method with parallel programs in a cluster environment

Zhen liu, Susan M. Bridges, and Ray Vaughn  
{zliu, bridges, vaughn}@cse.msstate.edu

CCSR (<http://www.cse.msstate.edu/~security>)

# Questions?