# Backtracking Intrusions

Sam King
Peter Chen
CoVirt Project, University of Michigan

# Motivation

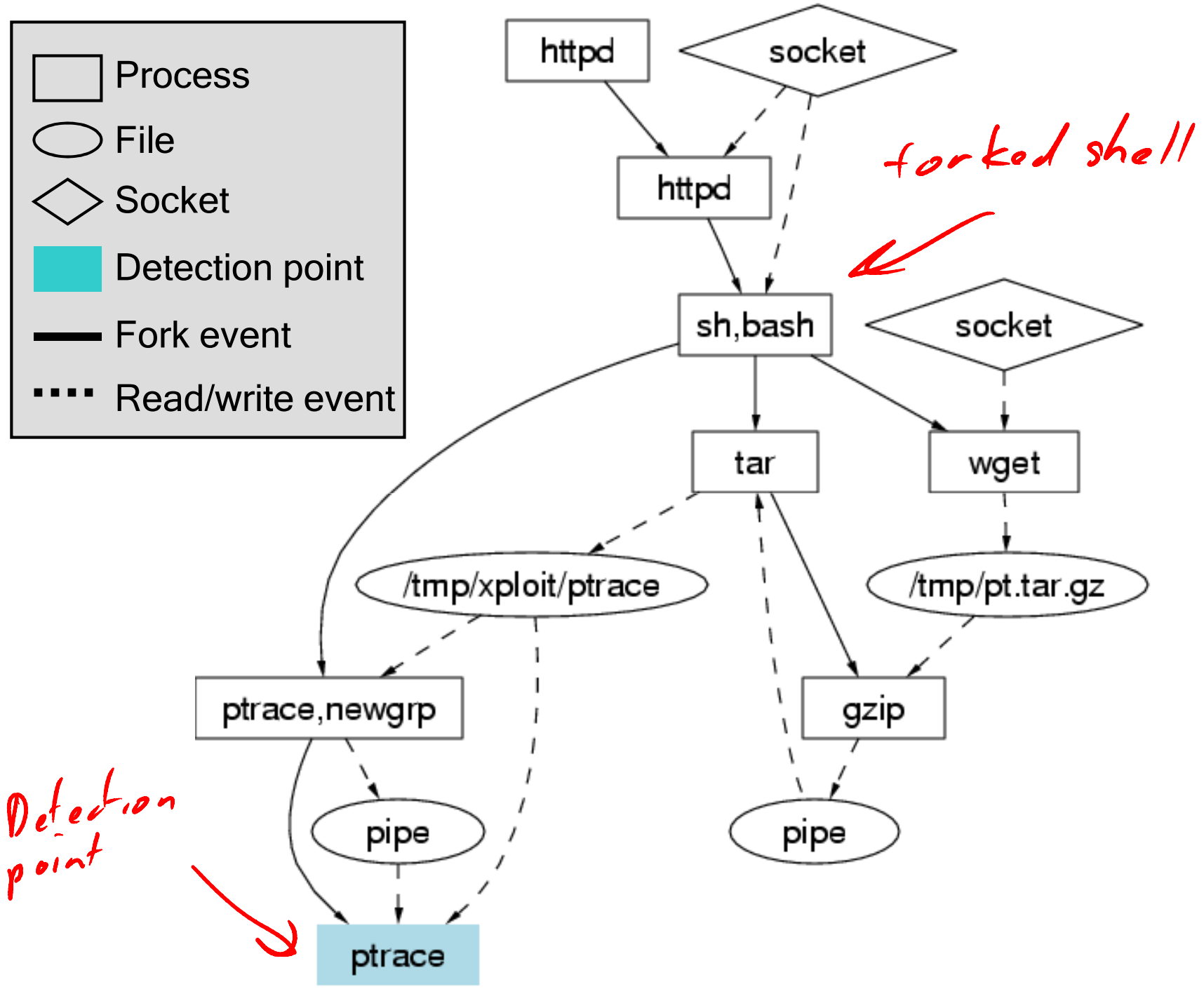
- Computer break-ins increasing
- Computer forensics is important
  - How did they get in

# Current Forensic Methods

- Manual inspection of existing logs
- System, application logs
  - Not enough information
- Network log
  - May be encrypted
- Disk image
  - Only shows final state
- Machine level logs
  - No semantic information
- No way to separate out legitimate actions

# BackTracker

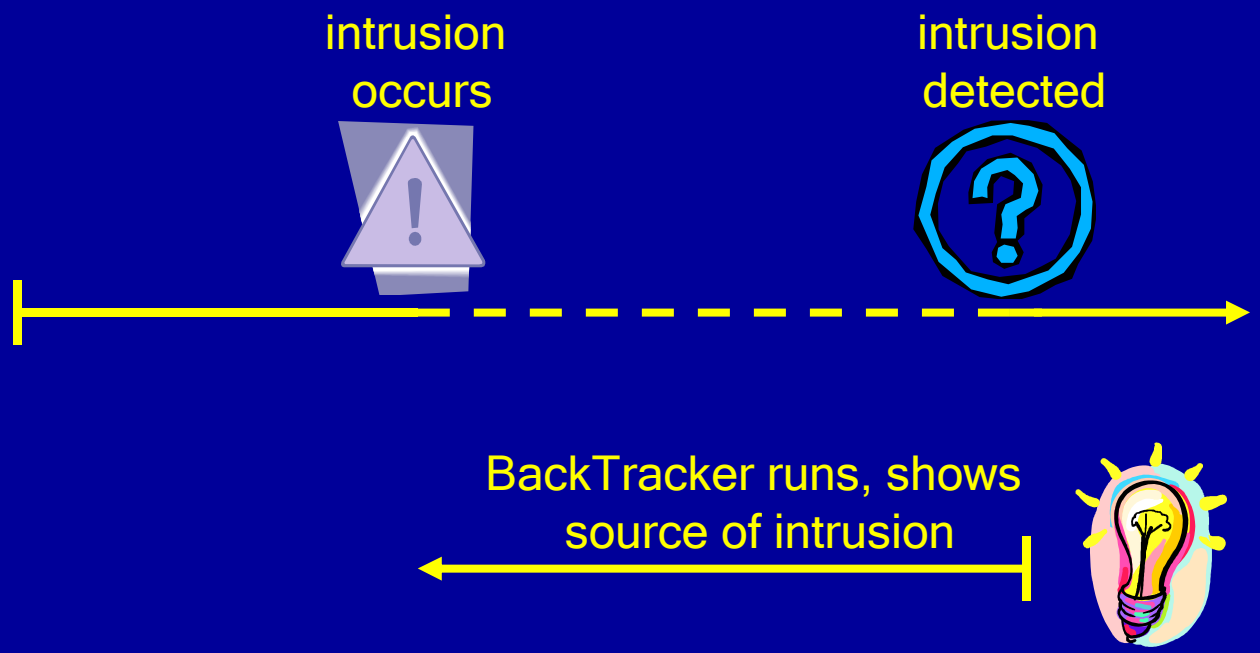
- Can we help figure out what was exploited?
- Track back to exploited application
- Record causal dependencies between objects


Process
File
Socket
Detection point
Fork event
Read/write event
httpd
socket
httpd
forked shell
sh,bash
socket
tar
wget
/tmp/xploit/ptrace
/tmp/pt.tar.gz
ptrace,newgrp
gzip
pipe
pipe
Detection point
ptrace

# Presentation Outline

- BackTracker design
- Evaluation
- Limitations
- Conclusions

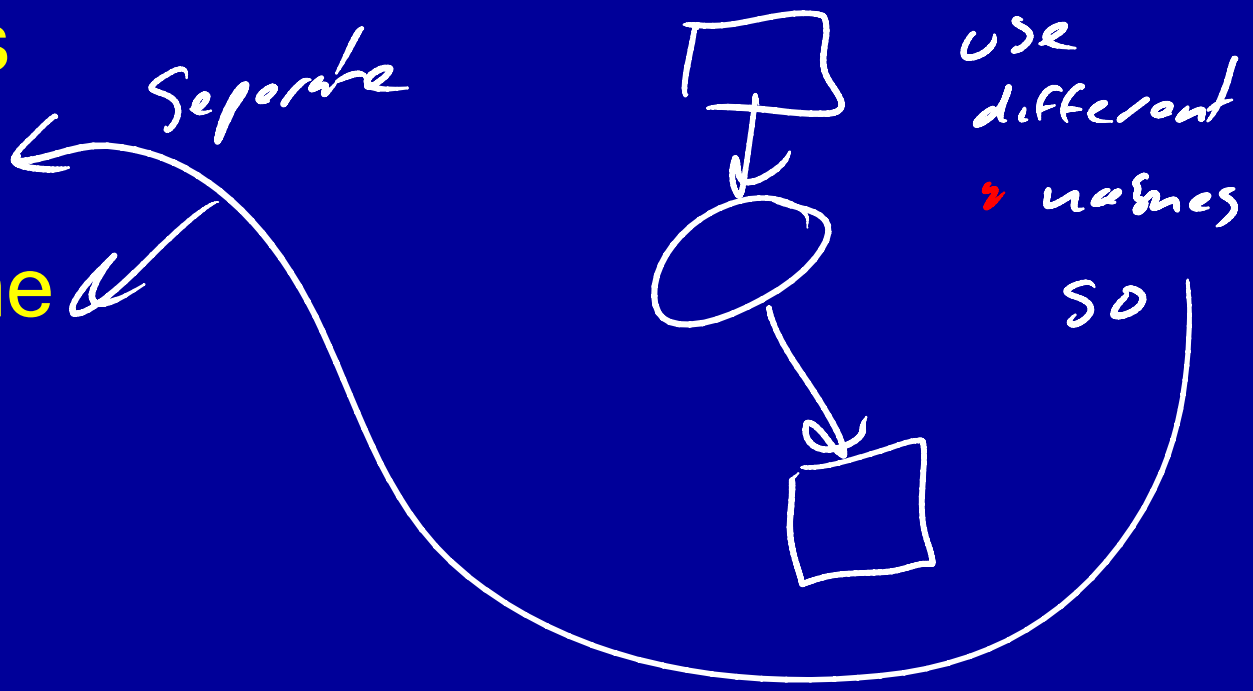
# BackTracker




- Online component, log objects and events
- Offline component to generate graphs

# BackTracker Objects

- Process
- File
- Filename



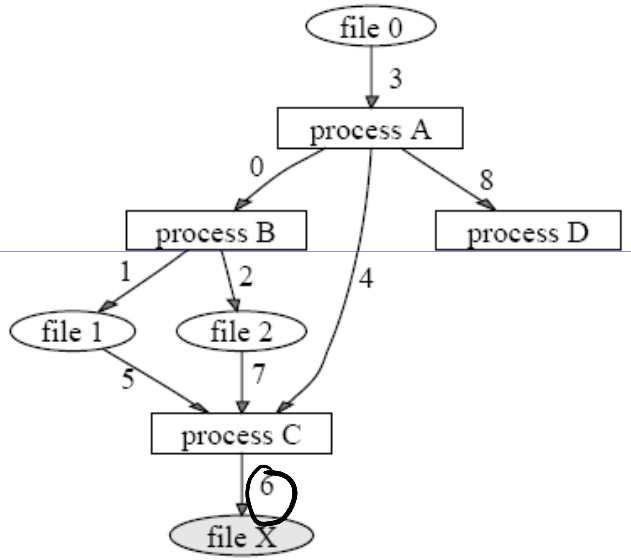
# Dependency-Forming Events

- Process / Process
  - fork, clone, vfork
- Process / File
  - read, write, mmap, exec
- Process / Filename
  - open, creat, link, unlink, mkdir, rmdir, stat, chmod, …

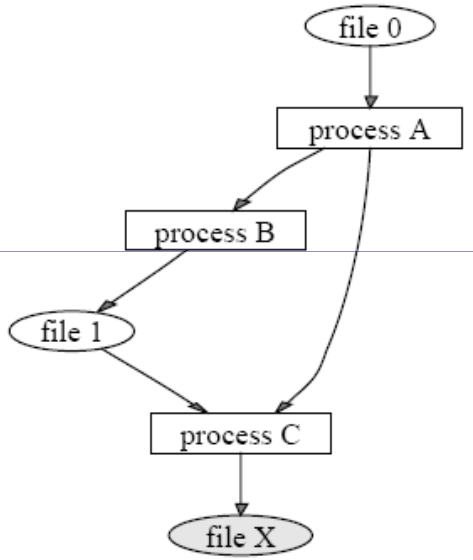
# GraphGen

time 0: process A creates process B
time 1: process B writes file 1
time 2: process B writes file 2
time 3: process A reads file 0
time 4: process A creates process C
time 5: process C reads file 1
time 6: process C writes file X
time 7: process C reads file 2
time 8: process A creates process D

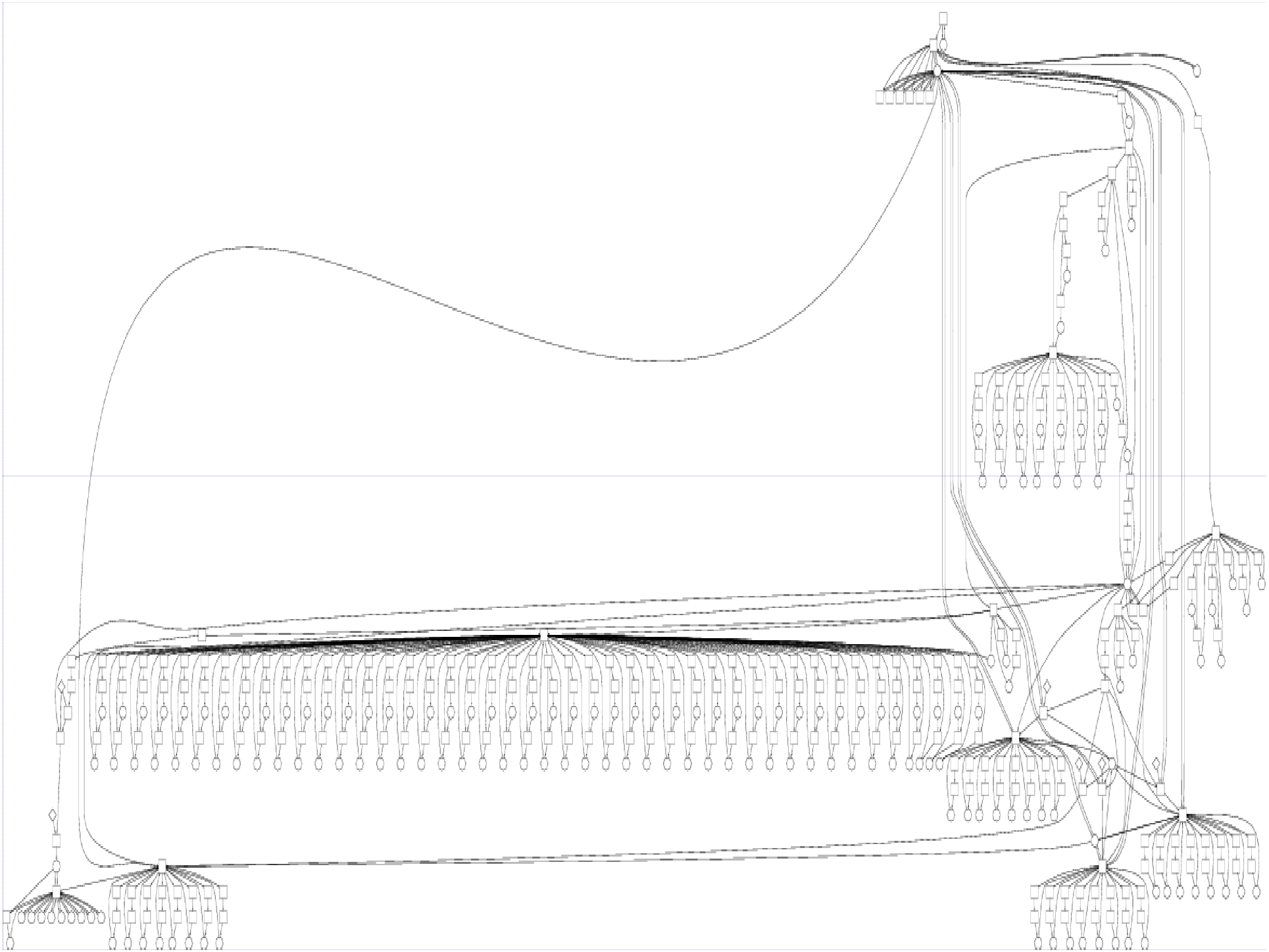
(a) event log




(b) dependency graph for complete event log

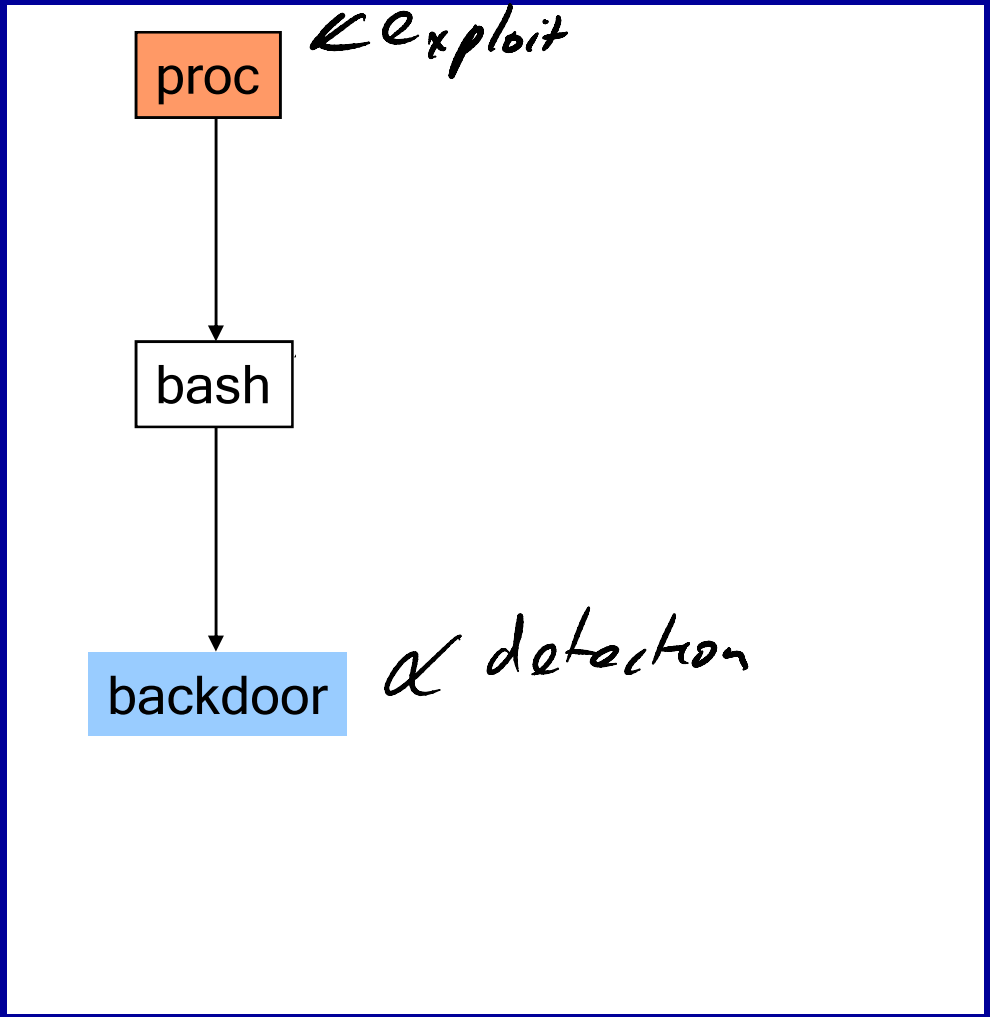



(c) dependency graph generated by GraphGen



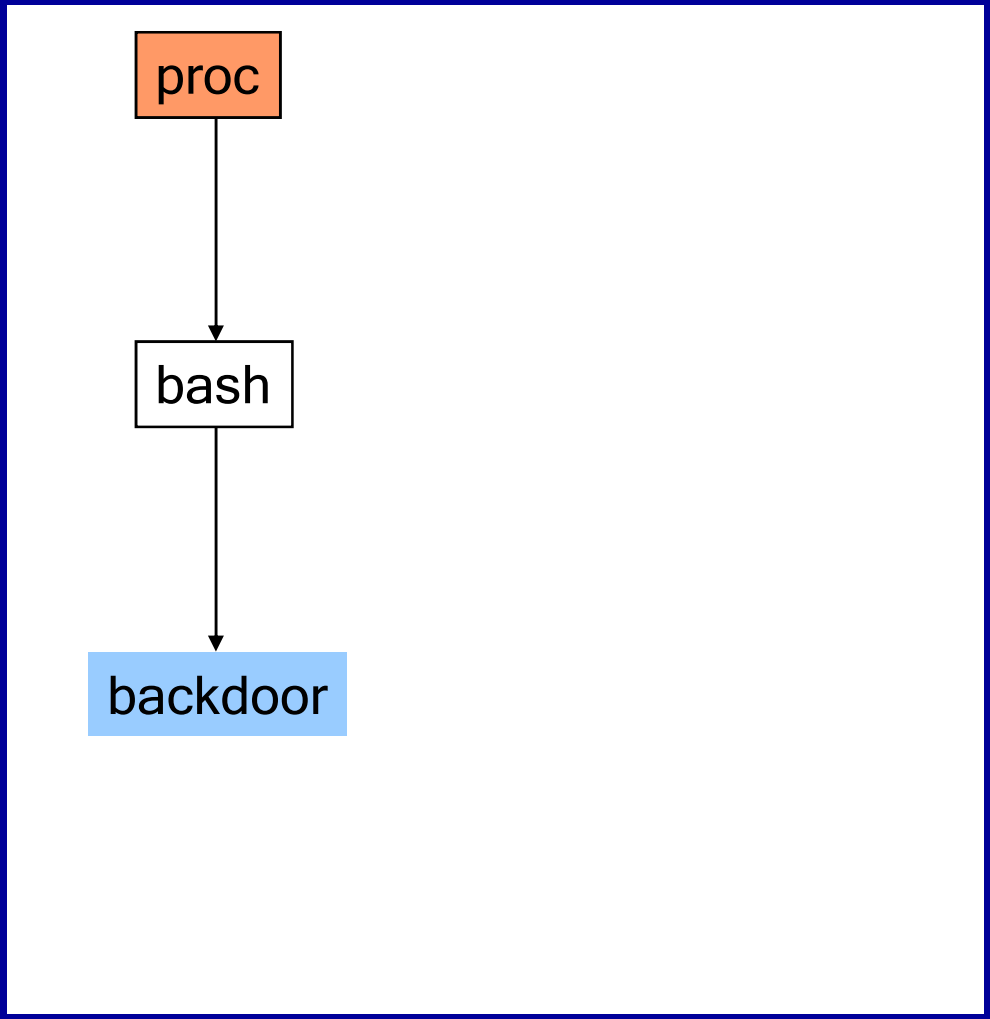
# Prioritizing Dependency Graphs

- Hide read-only files
- Eliminate helper processes
- Filter “low-control” events



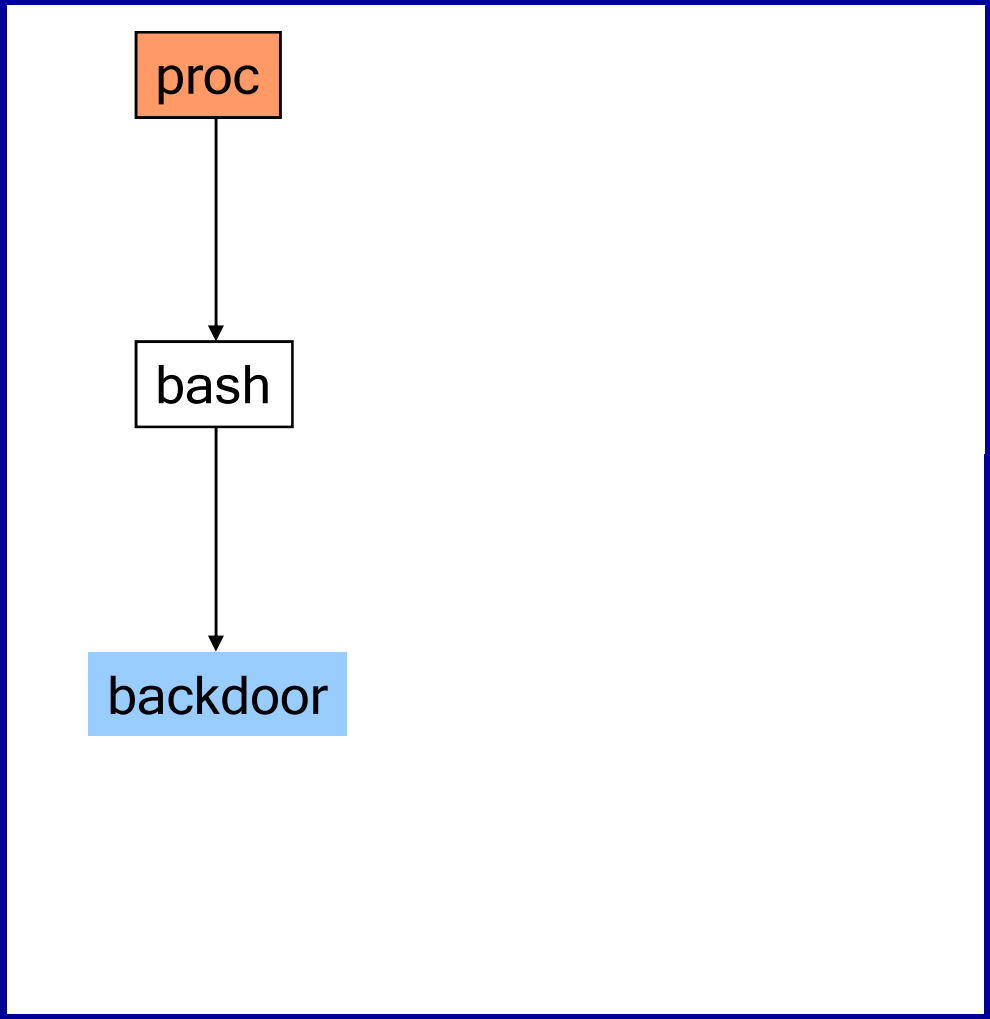
# Prioritizing Dependency Graphs

- Hide read-only files
- Eliminate helper processes
- Filter “low-control” events

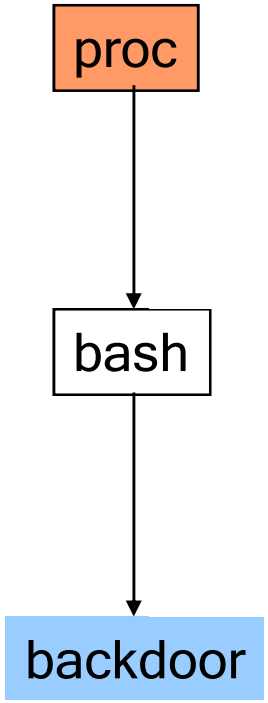


# Prioritizing Dependency Graphs

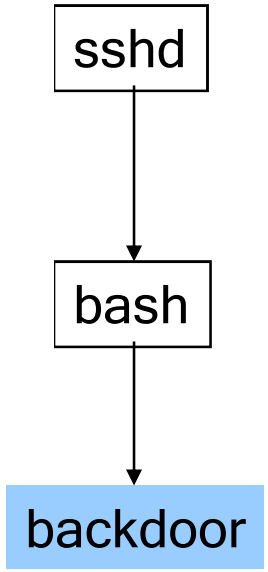
- Hide read-only files
- Eliminate helper processes
- Filter “low-control” events

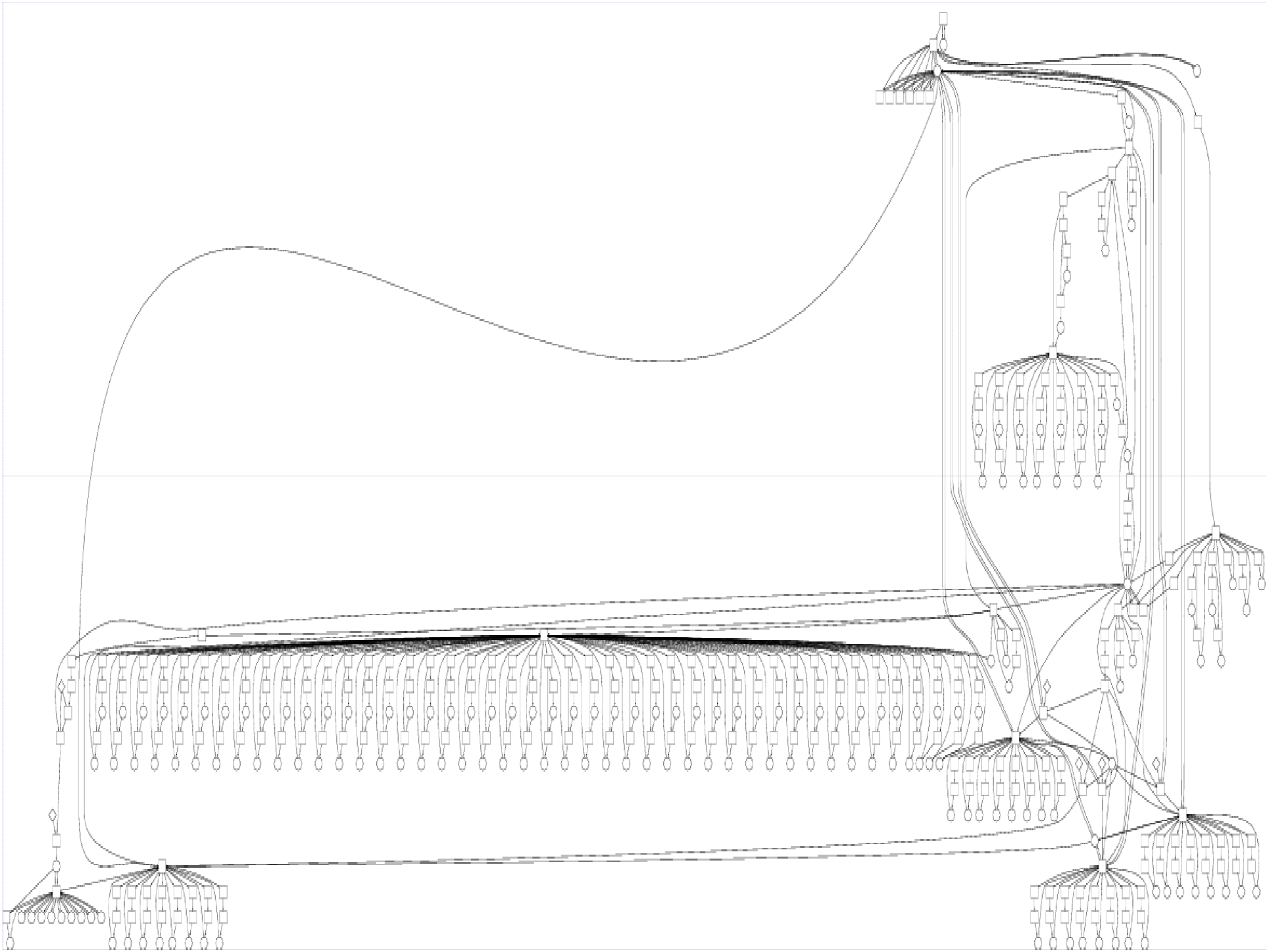


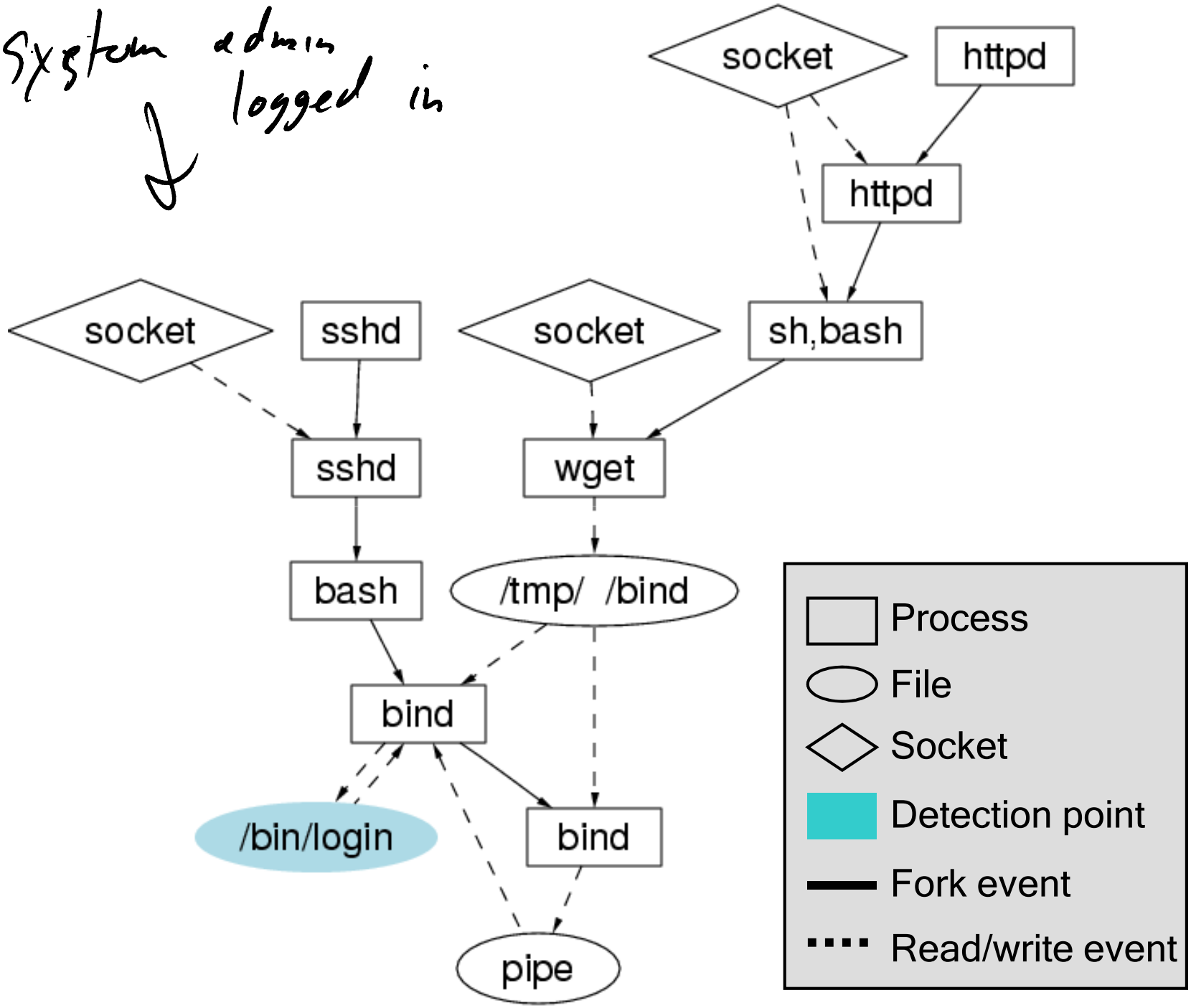
# Filtering “Low-Control” Events



# Filtering “Low-Control” Events

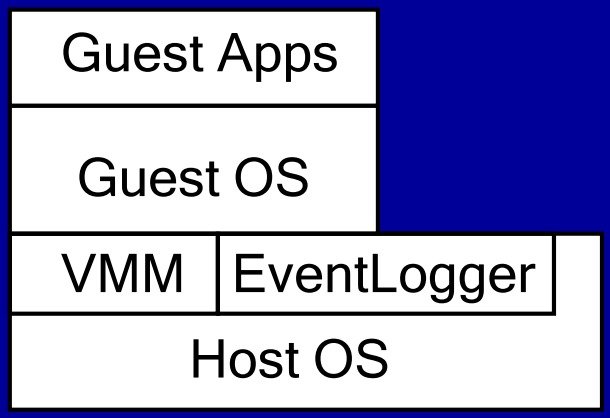




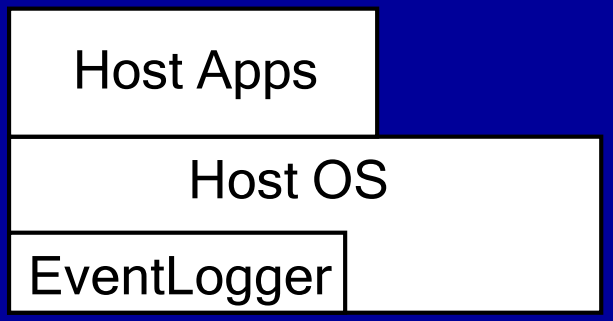

System admin logged in
socket
httpd
httpd
socket
sshd
socket
sh,bash
sshd
wget
bash
/tmp/ /bind
bind
/bin/login
bind
pipe
Process
File
Socket
Detection point
Fork event
Read/write event

# Implementation

- Prototype built on Linux 2.4.18
- Both stand-alone and virtual machine
- Hook system call handler
- Inspect state of OS directly




Virtual Machine Implementation

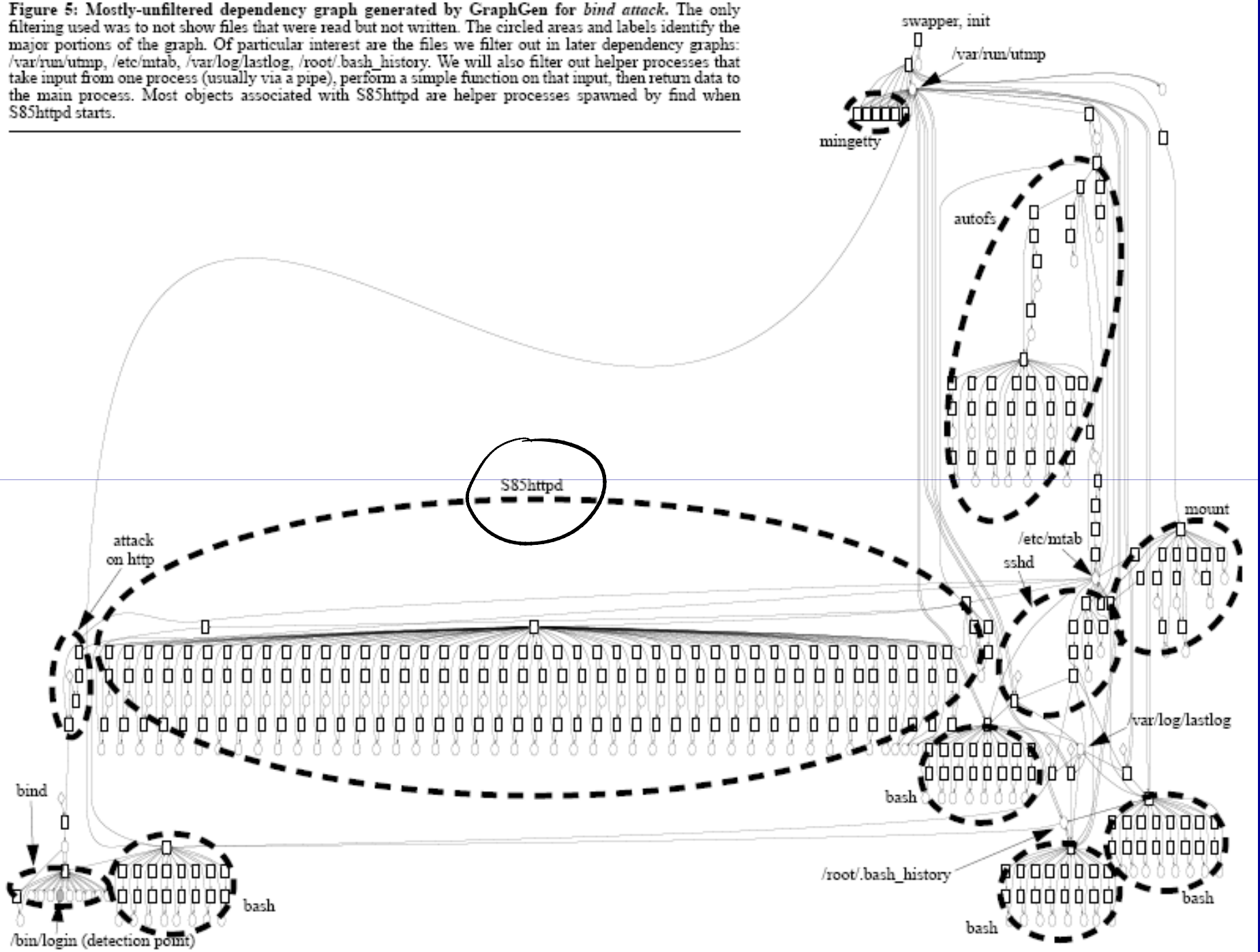


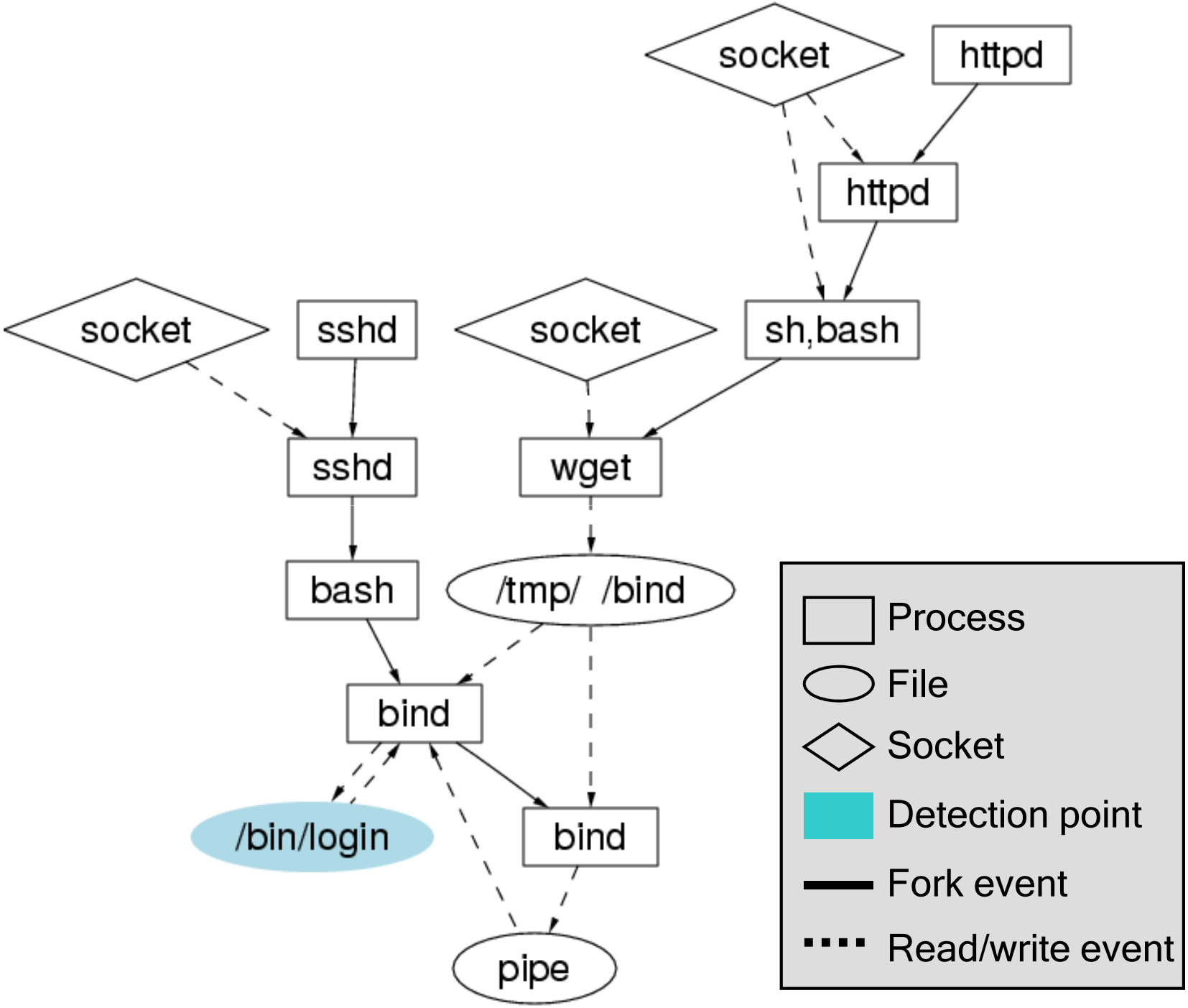

Stand-Alone Implementation

# Evaluation

- Determine effectiveness of Backtracker
- Set up Honeypot virtual machine
- Intrusion detection using standard tools
- Attacks evaluated with six default filtering rules

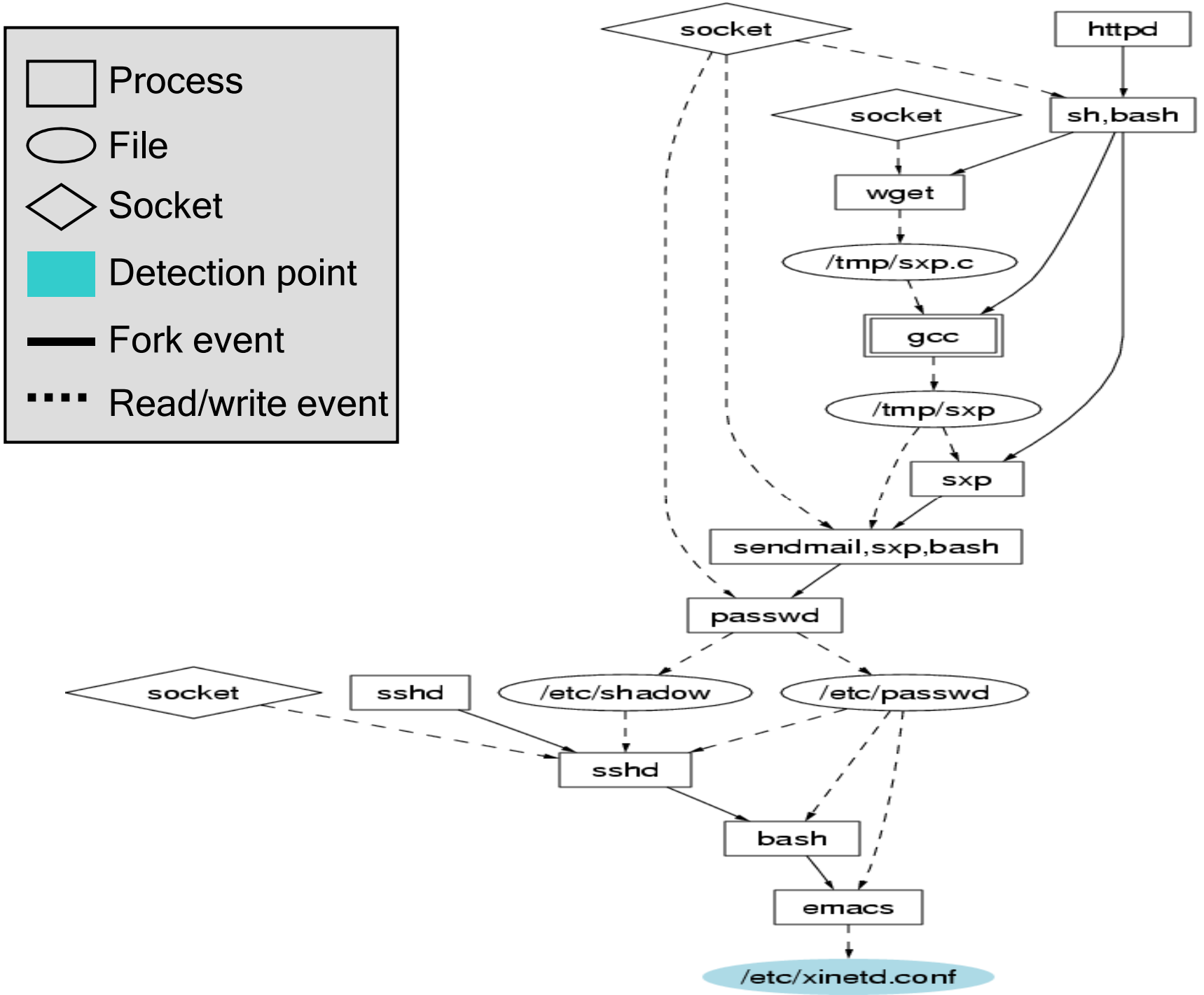
**Figure 5: Mostly-unfiltered dependency graph generated by GraphGen for *bind attack*.** The only filtering used was to not show files that were read but not written. The circled areas and labels identify the major portions of the graph. Of particular interest are the files we filter out in later dependency graphs: /var/run/utmp, /etc/mtab, /var/log/lastlog, /root/.bash_history. We will also filter out helper processes that take input from one process (usually via a pipe), perform a simple function on that input, then return data to the main process. Most objects associated with S85httpd are helper processes spawned by find when S85httpd starts.




socket
httpd
httpd
socket
sshd
socket
sh,bash
sshd
wget
bash
/tmp/ /bind
bind
/bin/login
bind
pipe
Process
File
Socket
Detection point
Fork event
Read/write event

# Distinguishing Related Events

- Ran Spec benchmarks
- Concurrently performed attack based on sendmail exploit


Process
File
Socket
Detection point
Fork event
Read/write event
socket
httpd
socket
sh,bash
wget
/tmp/sxp.c
gcc
/tmp/sxp
sxp
sendmail,sxp,bash
passwd
socket
sshd
/etc/shadow
/etc/passwd
sshd
bash
emacs
/etc/xinetd.conf

# Performance

| | *bind* (Fig 5-6) | *ptrace (Fig 1)* | *openssl-too (Fig 7)* | *self (Fig 8)* |
|---|---|---|---|---|
| time period being analyzed | 24 hours | | 61 hours | 24 hours |
| # of objects and events in log | 155,344 objects<br>1,204,166 events | | 77,334 objects<br>382,955 events | 2,187,963 objects<br>55,894,869 events |
| # of objects and events in unfiltered dependency graph | 5,281 objects<br>9,825 events | 552 objects<br>2,635 events | 495 objects<br>2,414 events | 717 objects<br>3,387 events |
| # of objects and events in filtered dependency graph | 24 objects<br>28 events | 20 objects<br>25 events | 28 objects<br>41 events | 56 (36) objects<br>81 (49) events |
| growth rate of EventLogger's log | 0.017 GB/day | | 0.002 GB/day | 1.2 GB/day |
| time overhead of EventLogger | 0% | | 0% | 9% |

**Table 1: Statistics for BackTracker's analysis of attacks.** This table shows results for three real attacks and one simulated attack. Event counts include only the first event from a source object to a sink object. GraphGen and the filtering rules drastically reduce the amount of information that an administrator must peruse to understand an attack. Results related to EventLogger's log are combined for the *bind* and *ptrace* attacks because these attacks are intermingled in one log. Object and events counts for the *self* attack are given for two different levels of filtering.

# BackTracker Limitations

- Layer-below attack
- Use “low control” events or filtered objects to carry out attack
- Hidden channels
- Create large dependency graph
  - Perform a large number of steps
  - Implicate innocent processes

# Future Work

- Department system administrators currently evaluating BackTracker
- Use different methods of dependency tracking
- Forward tracking

# Conclusions

- Tracking causality through system calls can backtrack intrusions
- Dependency tracking
  - Reduce events and objects by 100x
  - Still effective even when same application exploited many times
- Filtering
  - Further reduce events and objects