

# VIRTUALIZATION INTROSPECTION SYSTEM ON KVM-BASED CLOUD COMPUTING PLATFORMS

---

## 雲端運算平台之虛擬化偵察系統

李聖瑋

100356010@nccu.edu.tw

Advisor: 郁方

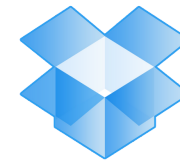
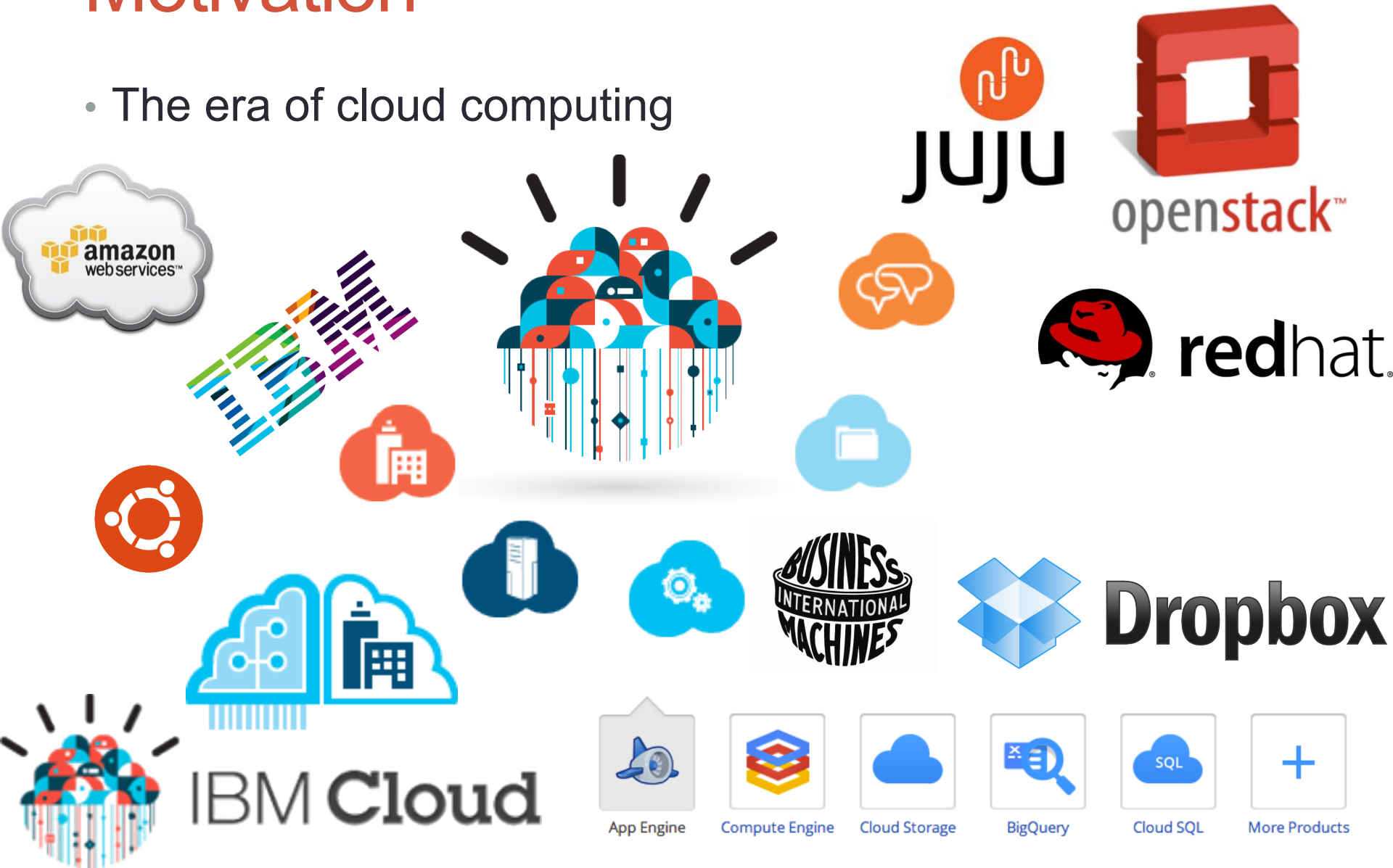
yuf@nccu.edu.tw

Software Security Lab.

國立政治大學資訊管理研究所

# Motivation

- The era of cloud computing



**Dropbox**



**IBM Cloud**



App Engine



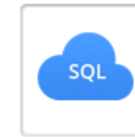
Compute Engine



Cloud Storage



BigQuery



Cloud SQL



More Products

# Motivation

- In the era of cloud computing, security threats could be a major stunning block.



# KVM

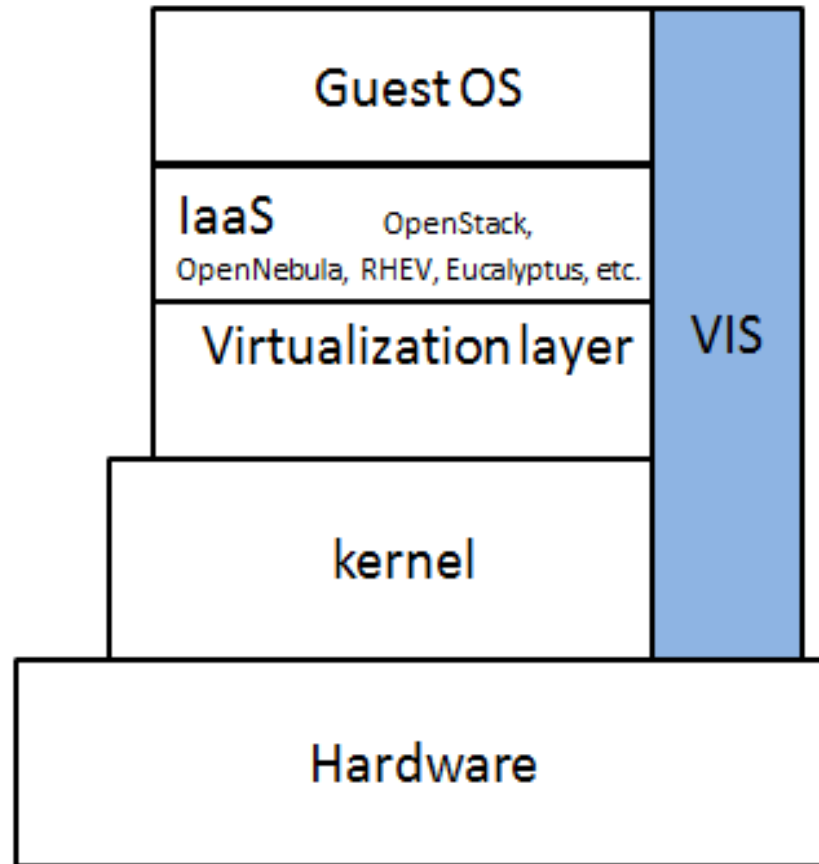
- Linux Kernel Virtual Machine (KVM) is one of the most common commodity hypervisor driver deployed in the IaaS layer of clouds.
- KVM provides a full-virtualization environment that emulates hardware as much as possible including CPU(s), network interfaces and mother-board chips.

# Attacking VM Hypervisor

- An intruder can attack the KVM hypervisor by exploiting a software defect in its kernel module and can get the Host privilege with which the intruder can take over the hypervisor
  - E.g., Cloudburst Attack that exploits the software vulnerability CVE-2011-1751 (N. Elhage, 2011)

# Objective

- We implement a system called **Virtualization Introspection System(VIS)** that detects and intercepts attacks from VMs by monitoring their status.
  - Detect VMs that attack Hypervisor
  - Detect VMs that attack other VMs
  - Detect VMs that have been compromised
- **VIS** can be deployed on most cloud operating systems based on KVM such as OpenStack and OpenNebula.

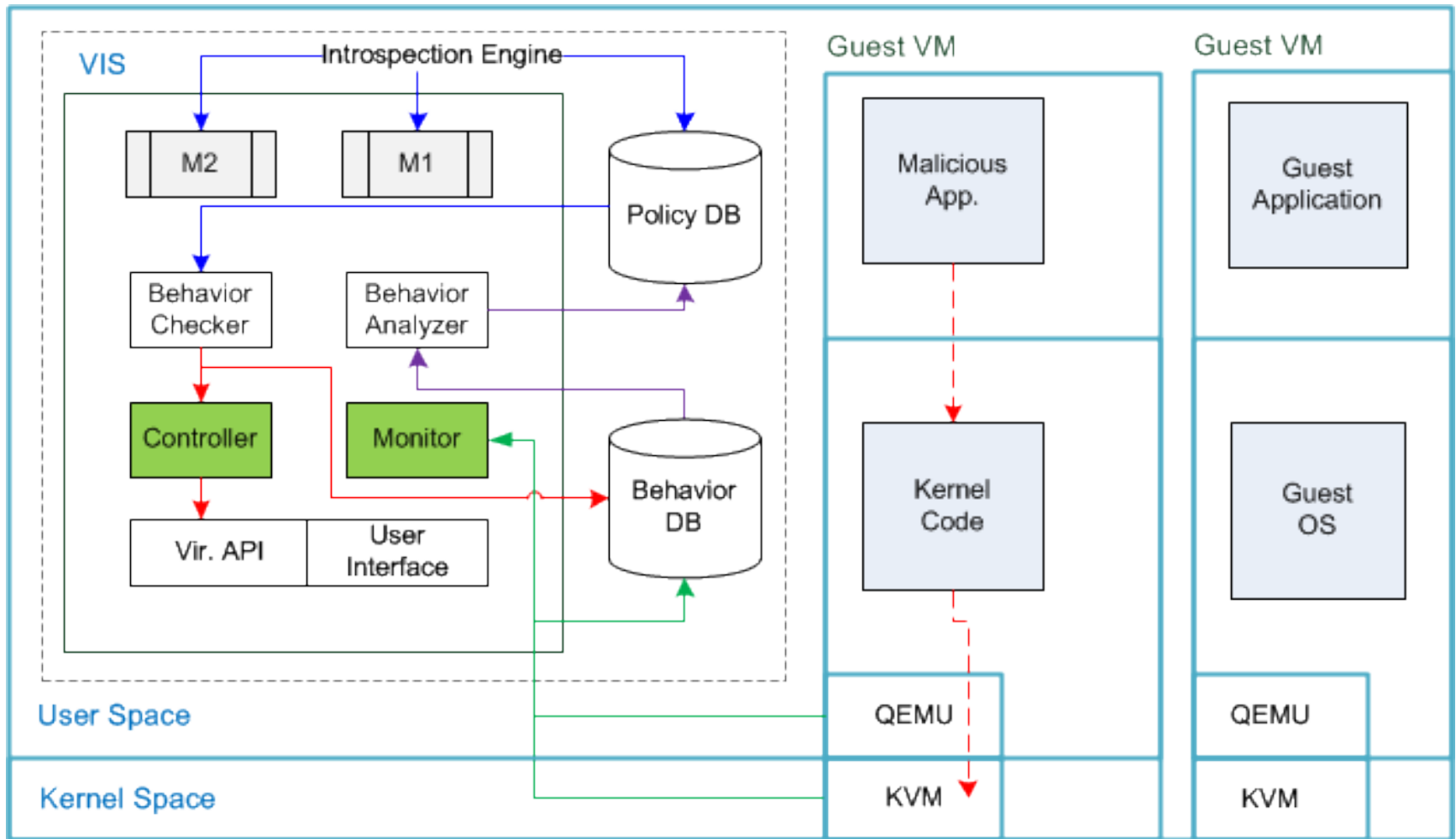


VIS with IaaS and Cloud middle ware

# Monitor VM status

- We collect both static and dynamic information to characterize VM behaviors
  - Run-time status: using “strace” to collect underlying system calls
  - Static status: using “qemu-monitor” to check installed VM image (hardware).





Cloudburst Attack Path

# The VIS Architecture

# VIS

- Monitor
  - Monitor running behaviors/status of VMs
    - Strace, Qemu-monitor
  - Store the data into Behavior Databases
  - Visualize running status
- Behavior Analyzer
  - Derive policy/rules for known malicious behaviors
  - For each rule, we implement an introspection module

# VIS

- **Introspection Modules**
  - Each is an independent python module that can be loaded dynamically to detect malicious VM on a specific behavior
- **Policy Database**
  - Rules for the actions on malicious VMs
- **Behavior Database**
  - Store the previously analyzed patten of malicious behavior of VMs
  - Save the category data as  
Role → Period → Program → System call

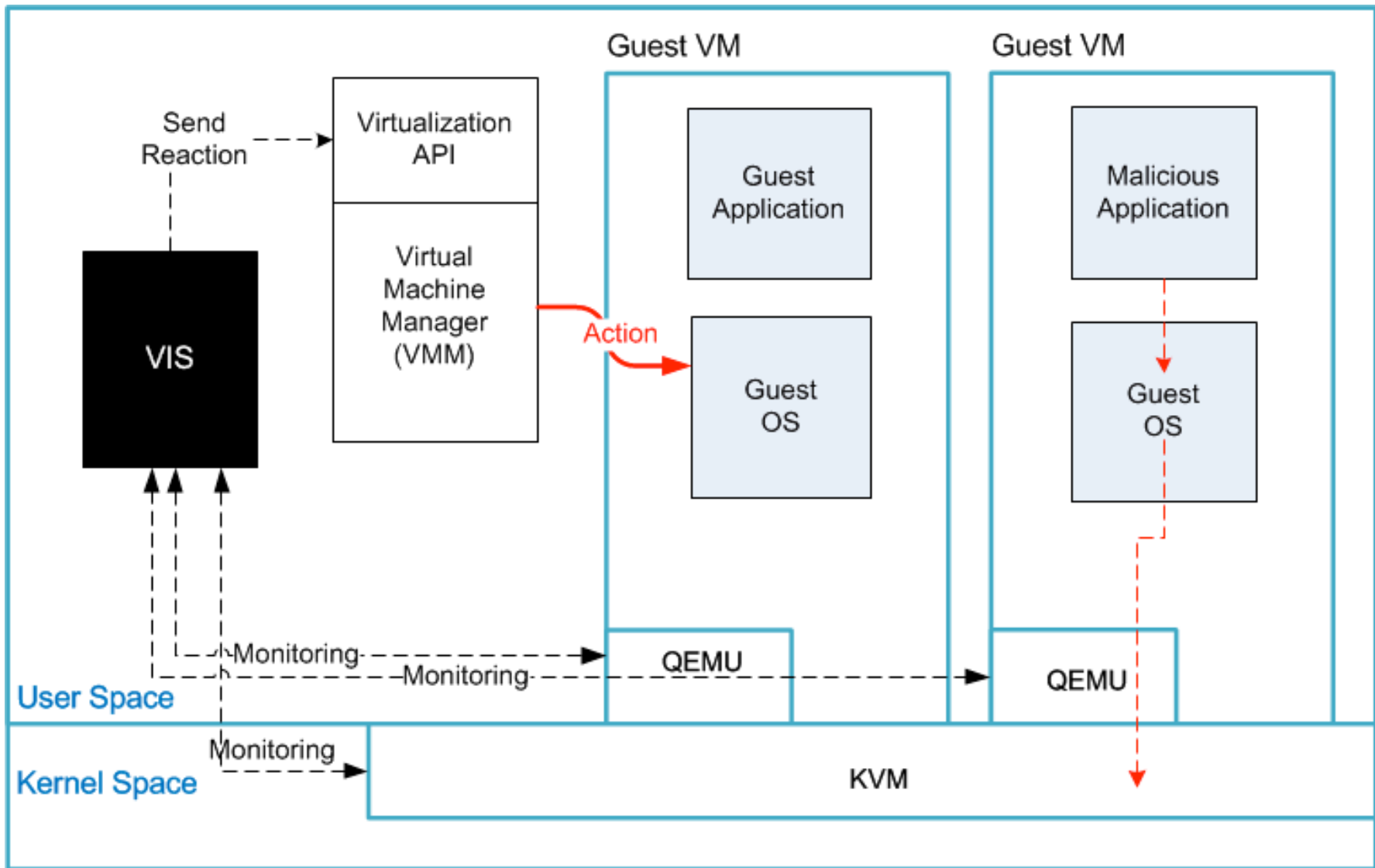
# VIS

- Behavior Checker

- Executes the modules to compare the behaviors of underlying VMs with policy rules
- Identifies VMs that are (1) executing malicious programs/system calls or (2) in compromised status
- Sends the domain action message to controller

## Controller

- Executes commands from Behavior Checker: destroy, shutdown, migrate etc.
- This can be done by passing the message to cloud middleware (e.g. OpenStack, OpenNebula)
- In our current implementation, we use libvirt and virsh to control the compromised and malicious VMs.



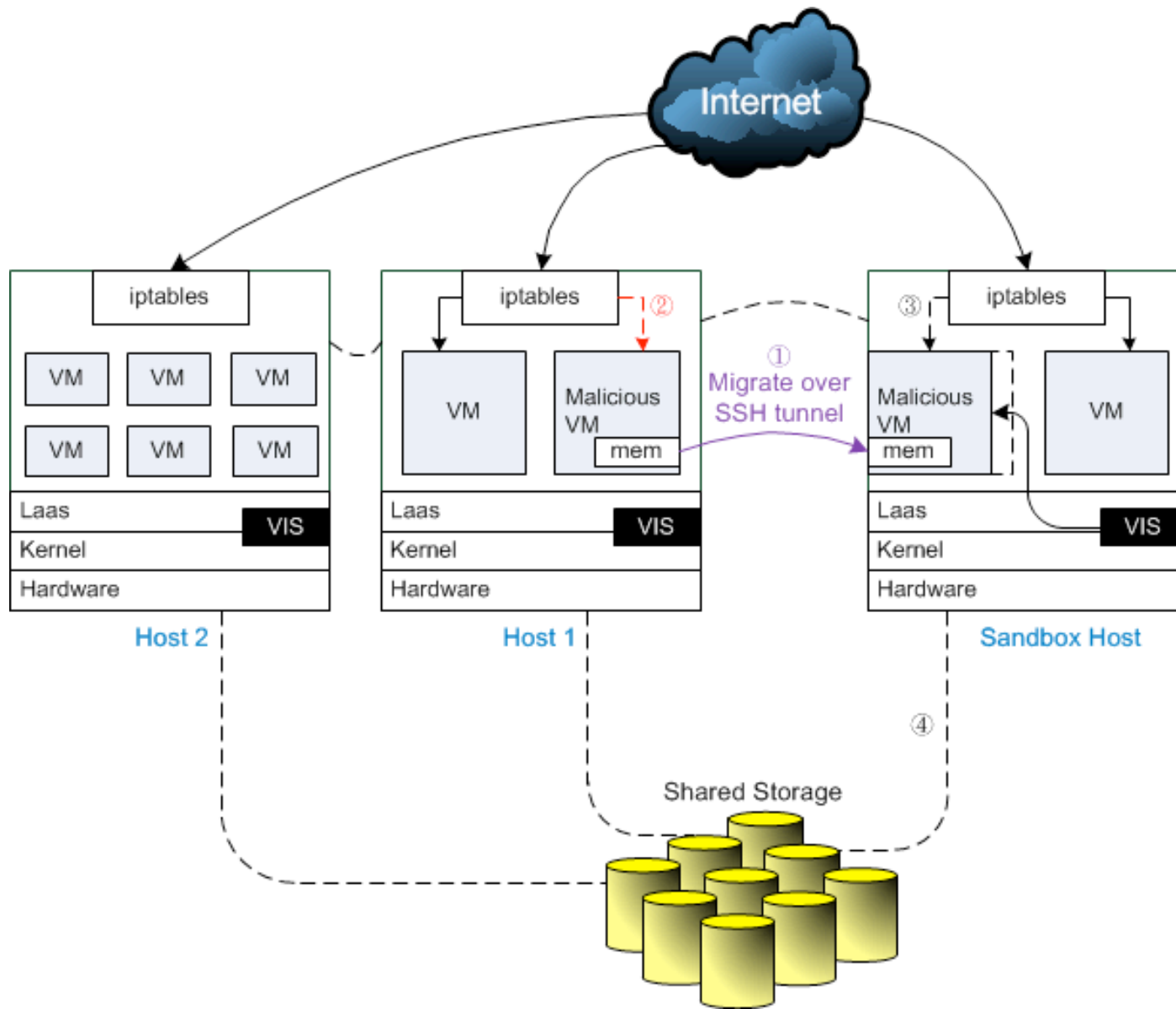
Termination: Shutdown the attack VMs

# VIS Defense Operation

- Termination
  - Direct shutdown and offline migration
  - VMs that are confirmed with severe attacks
- Isolation
  - Online migration (to a physical isolated place)
  - Potential vulnerable VMs, e.g., VMs that are identified been compromised

- Isolation:

  - Migrate Malicious VMs and Redirect iptables

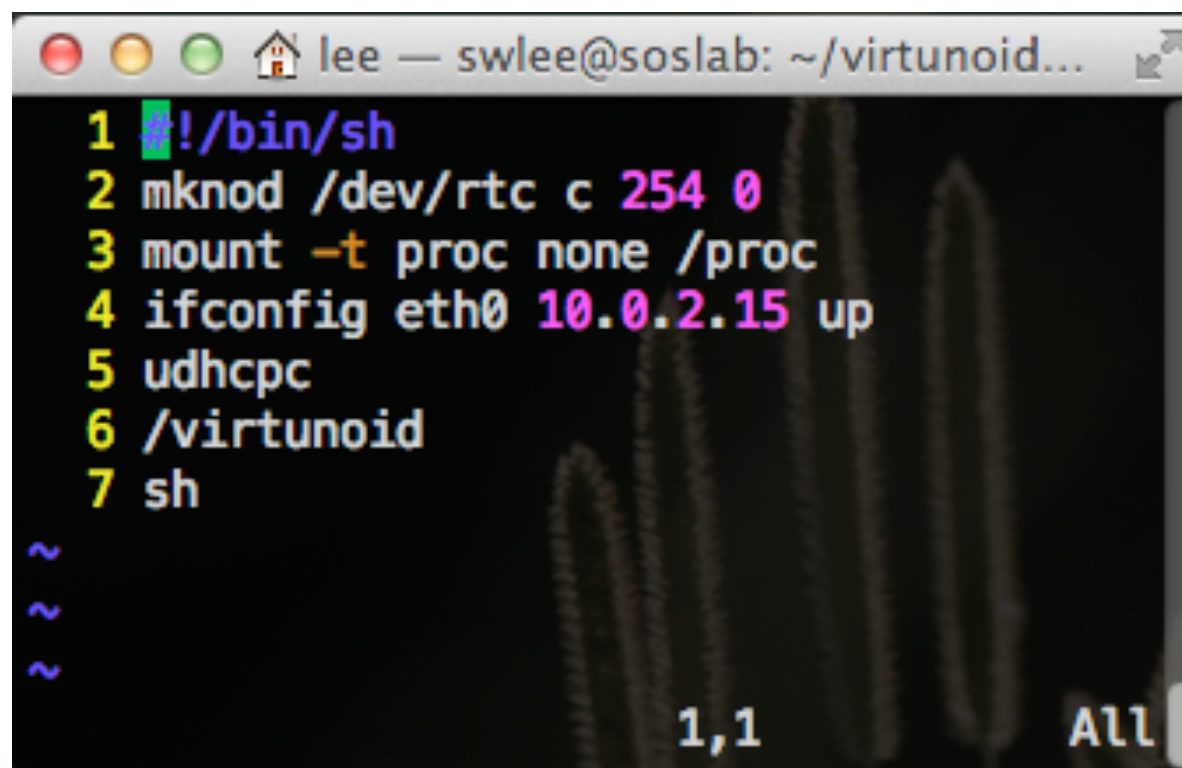


# Evaluation

- Detect Cloudburst Attack
- Detect Social Engineering Attack



# Detect Cloudburst Attack



```
lee — swlee@soslab: ~/virtunoid...  
1 #!/bin/sh  
2 mknod /dev/rtc c 254 0  
3 mount -t proc none /proc  
4 ifconfig eth0 10.0.2.15 up  
5 udhcpc  
6 /virtunoid  
7 sh  
~  
~  
~  
1,1 All
```

Build a VM to execute the cloudburst attack

# Detect Cloudburst Attack

- The attack exploits the KVM software defect (CVE-2011-1751: a pointer leaking that is triggered by unplugging the PIIX4\_PM device)

```
43,101d42
<     dev: PIIX4_PM, id ""
<         dev-prop: smb_io_base = 45312
<         bus-prop: addr = 01.3
<         bus-prop: romfile = <null>
<         bus-prop: rombar = 1
<         bus-prop: multifunction = off
<         bus-prop: command_serr_enable = on
<         class Bridge, addr 00:01.3, pci id 8086:7113 (sub 1af4:1100)
<         bus: i2c
```

# Detect Cloudburst Attack

- Checking the change of QEMU device

```
43,101d42
< dev: PIIX4_PM, id ""
<   dev-prop: smb_io_base = 45312
<   bus-prop: addr = 01.3
<   bus-prop: romfile = <null>
<   bus-prop: rombar = 1
<   bus-prop: multifunction = off
<   bus-prop: command_serr_enable = on
< class Bridge, addr 00:01.3, pci id 8086:7113 (sub 1af4:1100)
< bus: i2c
```

# Detect Social Engineering Attack

- We replay social engineering attacks on VMs
  - Hacker VM that executes the attacks
  - Victim VM that is compromised
  - Normal VM that has the same operation system as Hacker VM

# Detect Social Engineering Attack

Hacker VM (BackTrack 5 R3)

- Period Initial: do nothing
- Period Prepare: setup the social engineering attack (send fishing emails)
- Period Compromise: Victim clicks malicious url to build ssh channel
- Period Attack I: Hacker kills the process inside Victim
- Period Attack II: Hacker keystrokes the Victim for Password

# Detect Social Engineering Attack

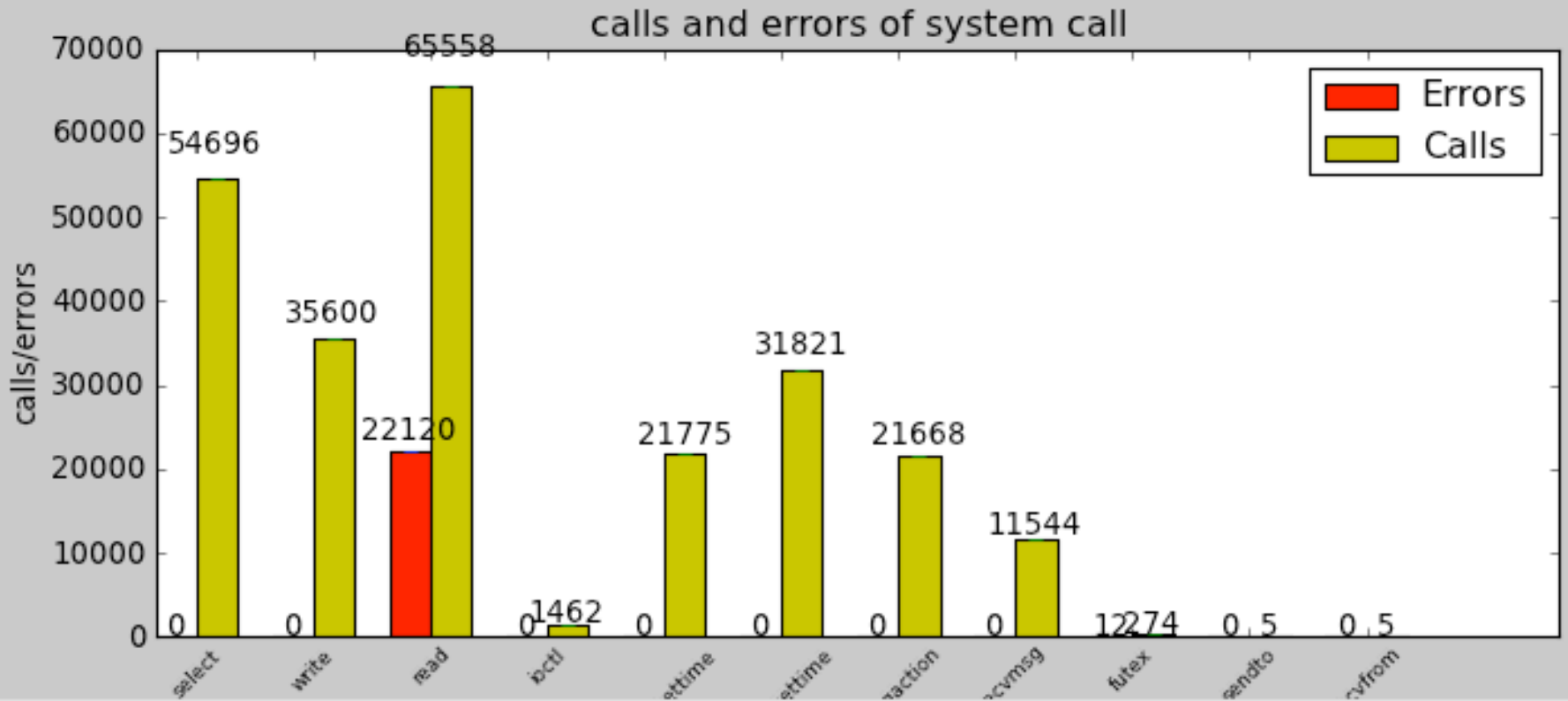
## Victim VM (Windows 7)

- Period Initial: Having Firewall and Anti-virus installed
- Period Normal: Receive fishing email with malicious url from Hacker
- Period Compromise: Click the malicious url
- Period under Attack I
- Period under Attack II

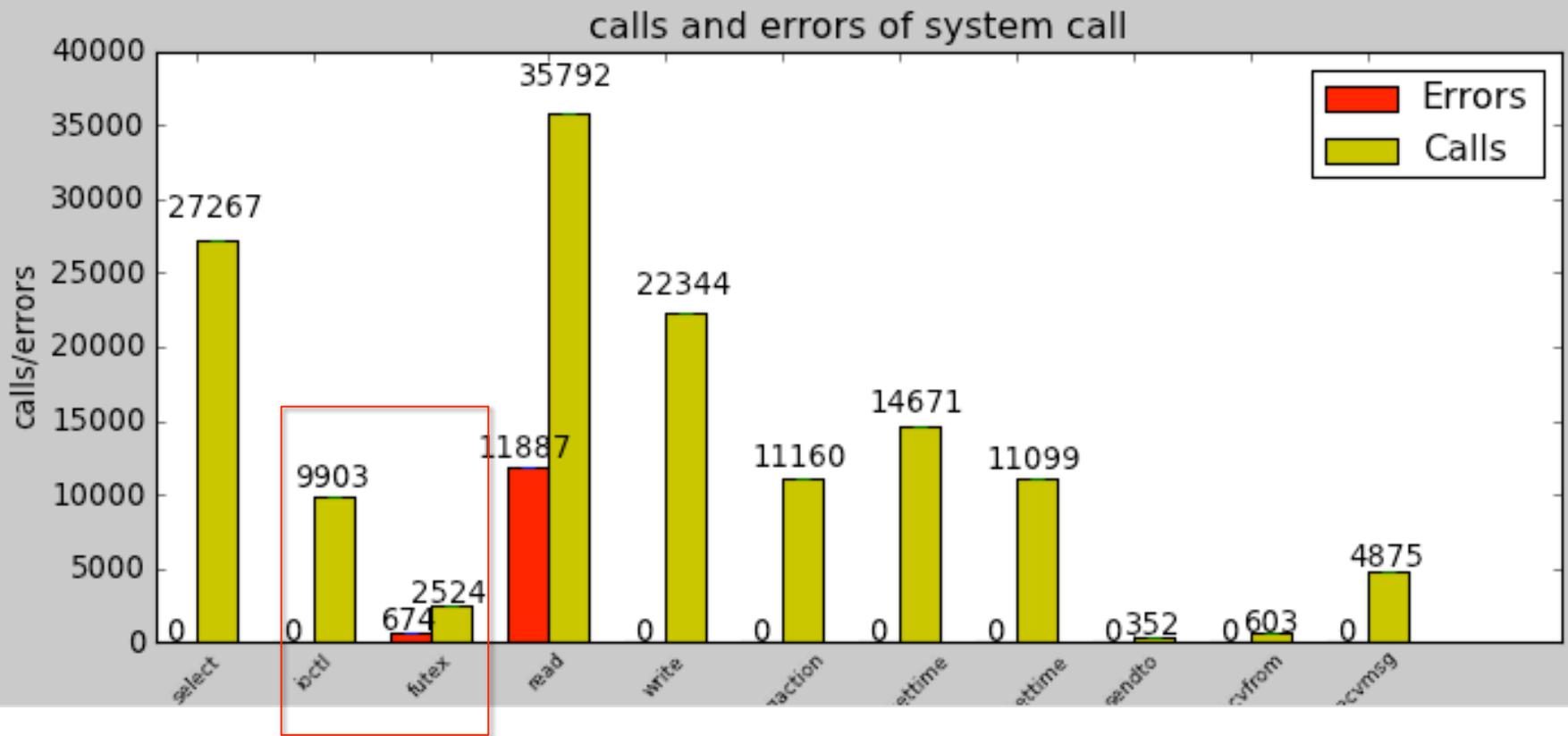
## Normal VM (BackTrack 5 R3)

- Do nothing.

# System Call Distribution (via Strace)

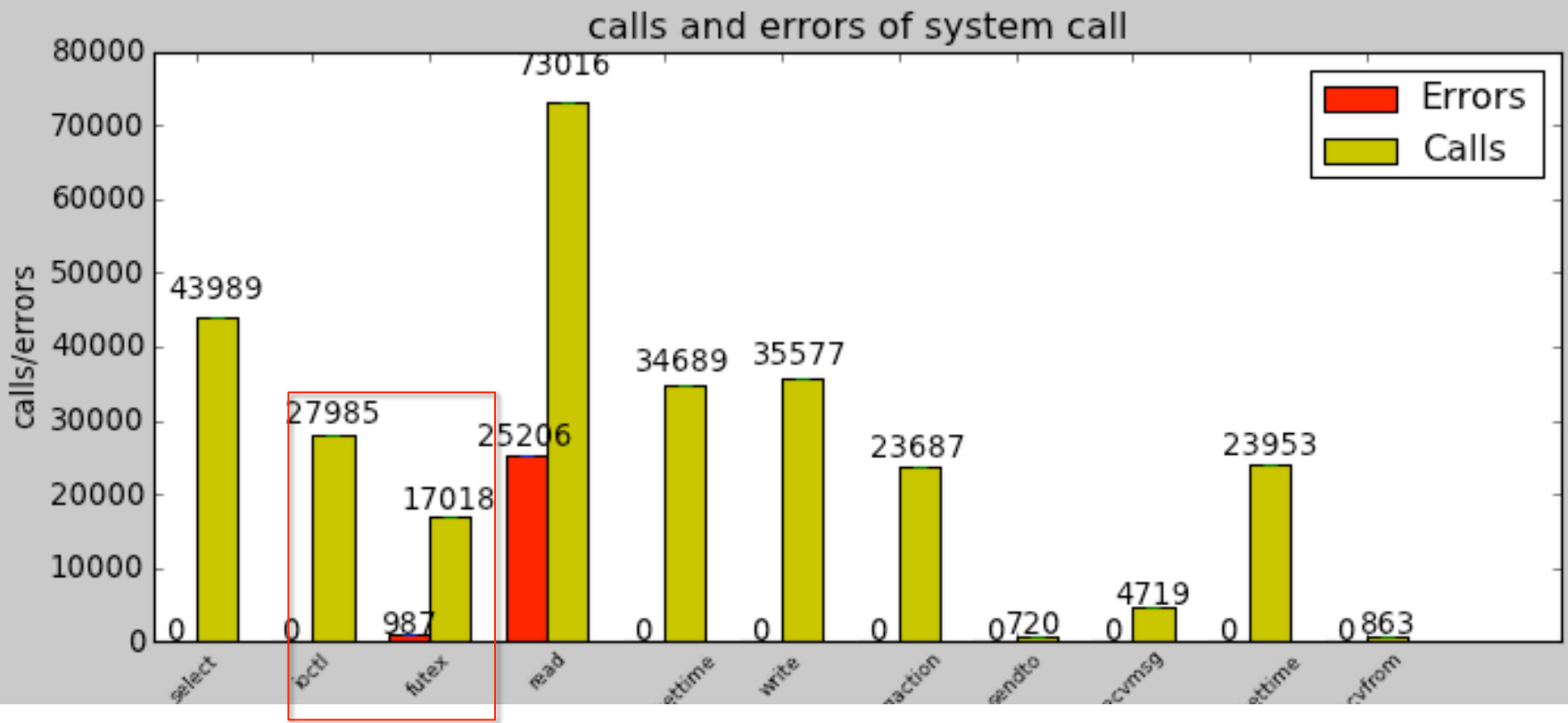


Normal VM- the same as Hacker VM: Period initial



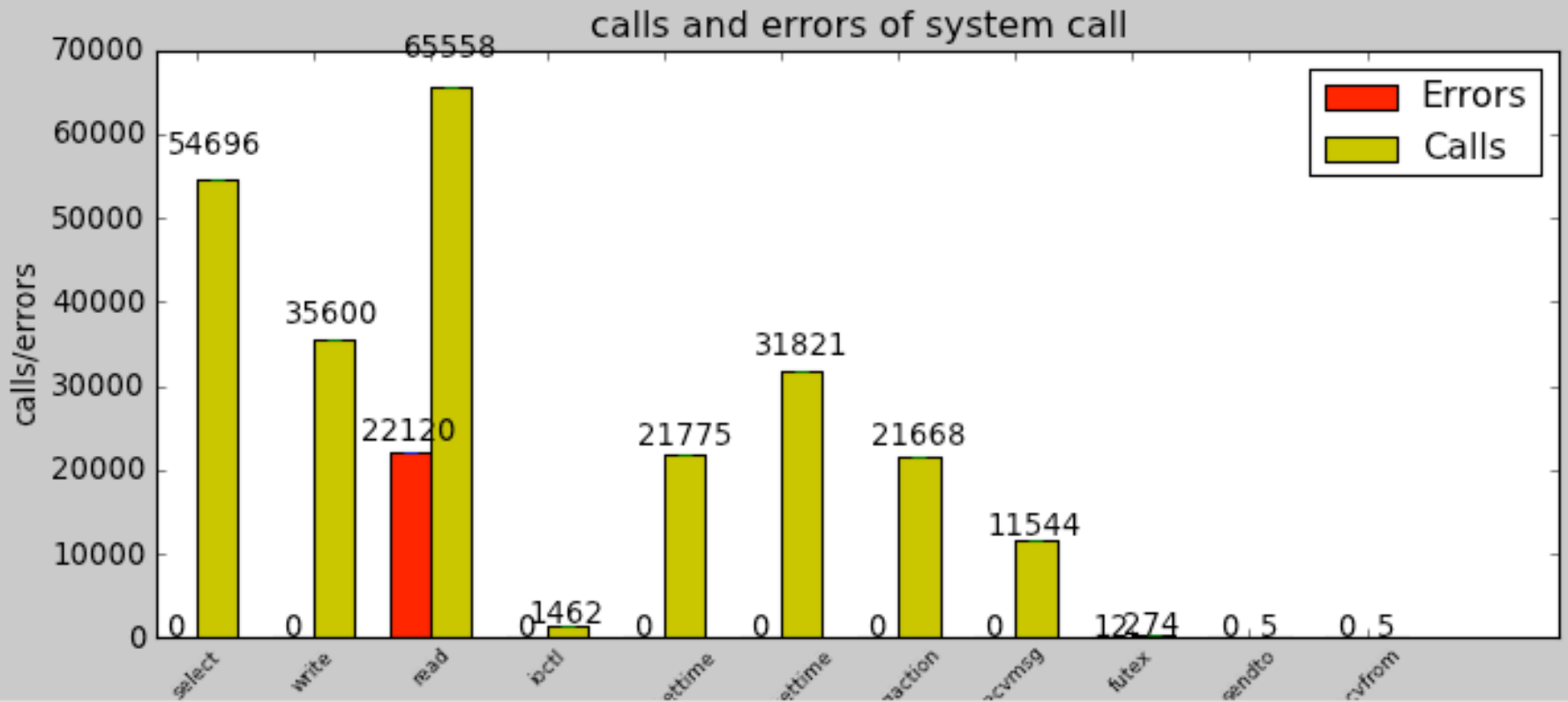
Hacker VM: Period Compromise





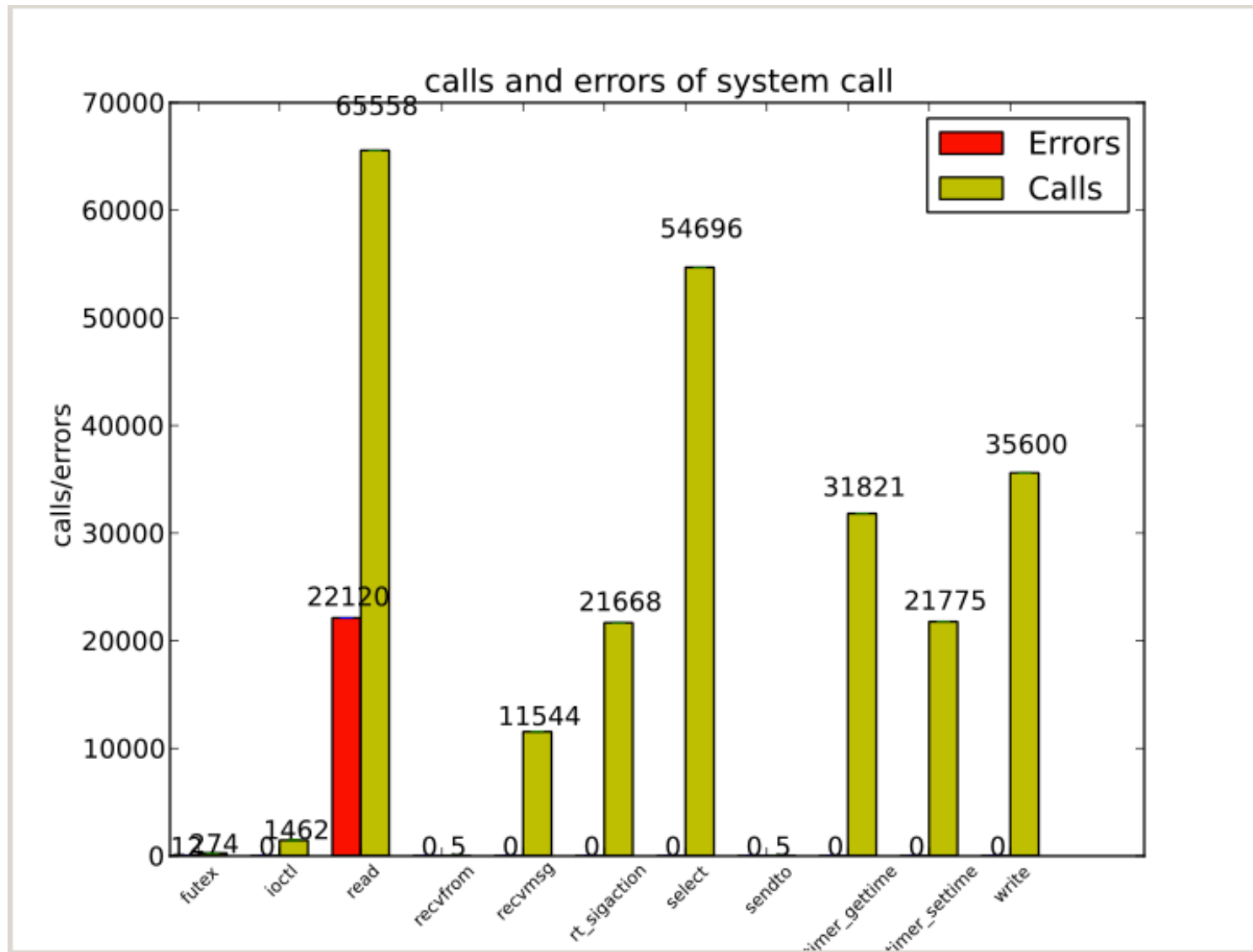
Victim VM: Period Compromise

# Run the testing

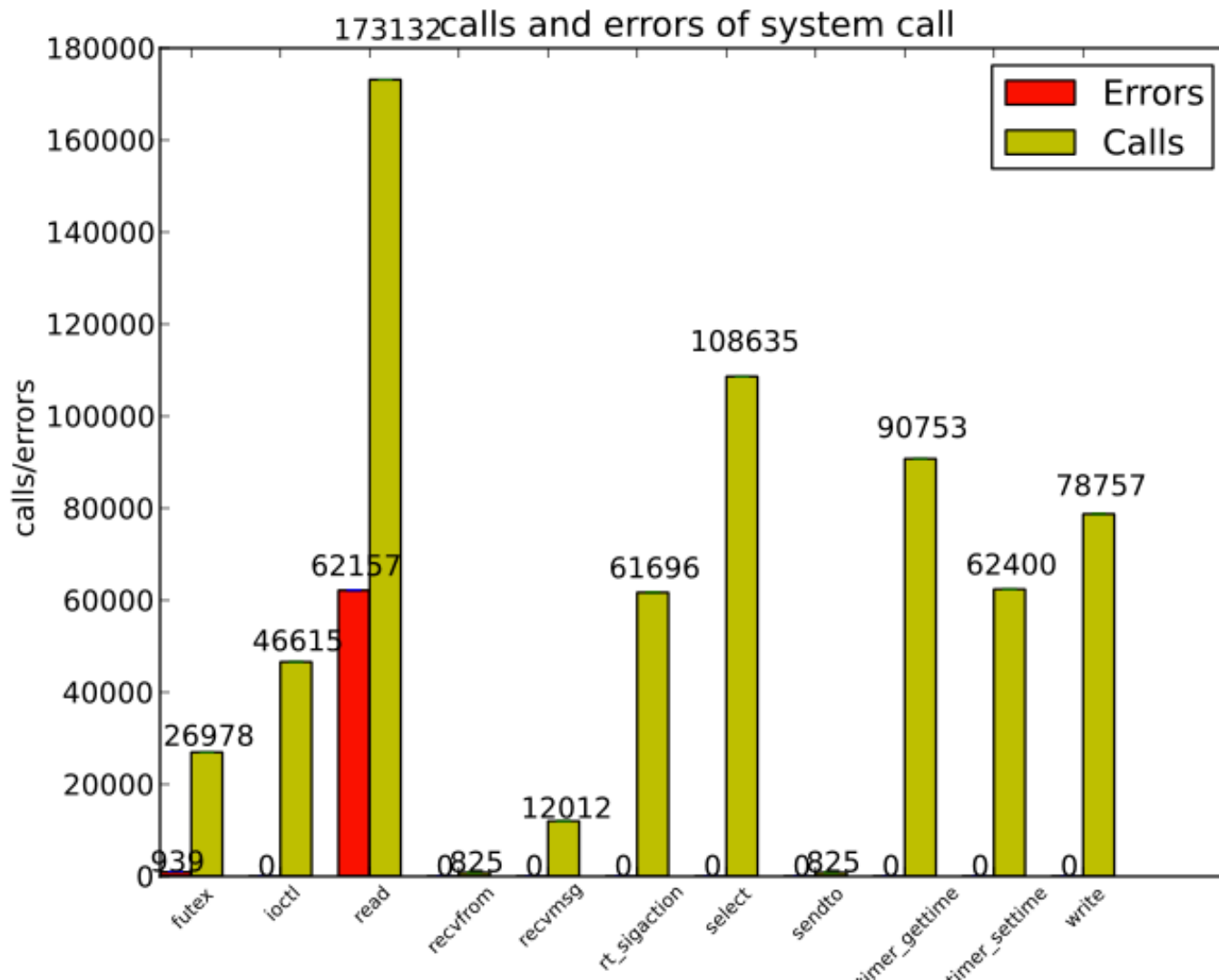


Hacker VM : Period initial

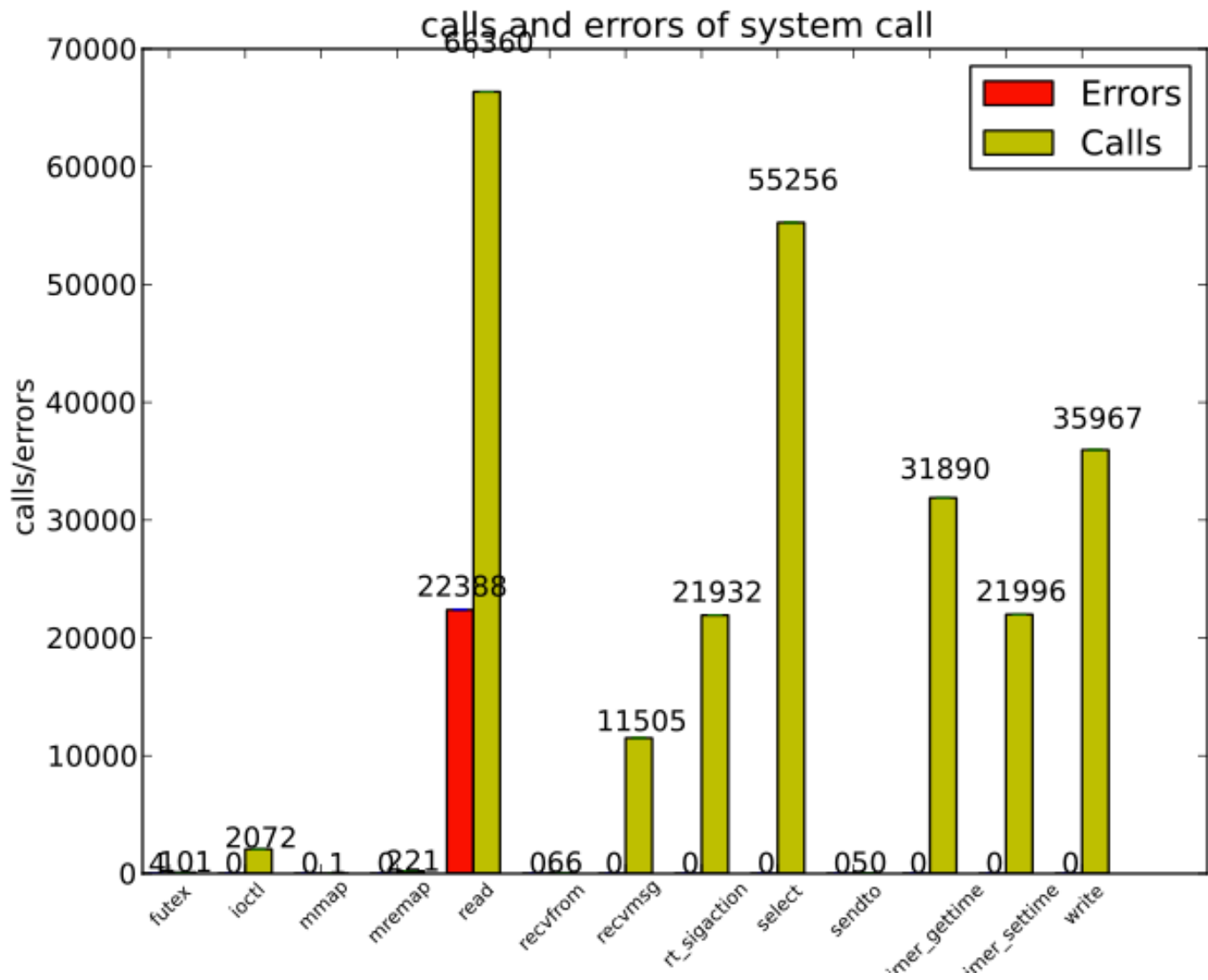
# 01-01\_normal-hacker



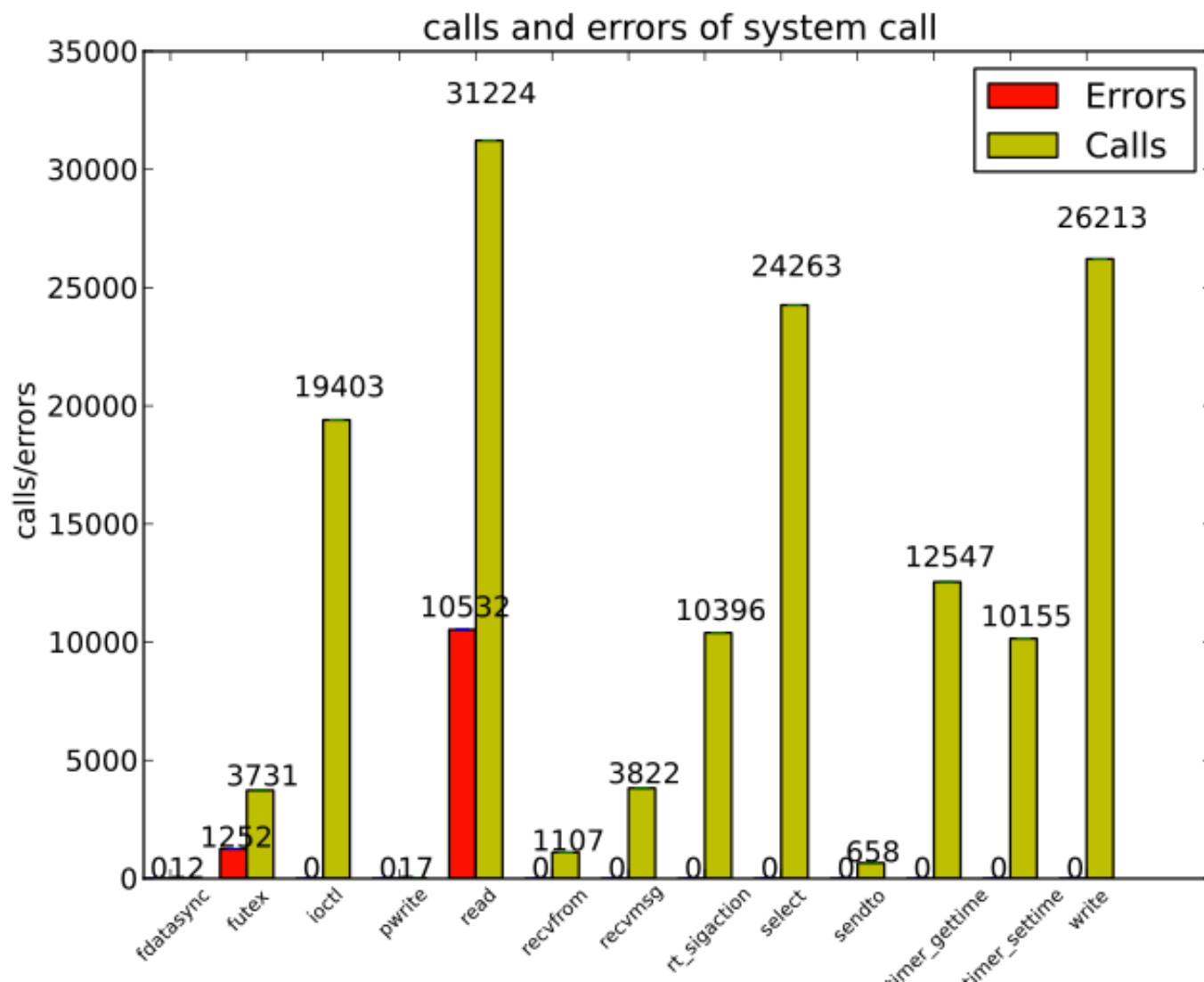
# 01-02\_normal-victim



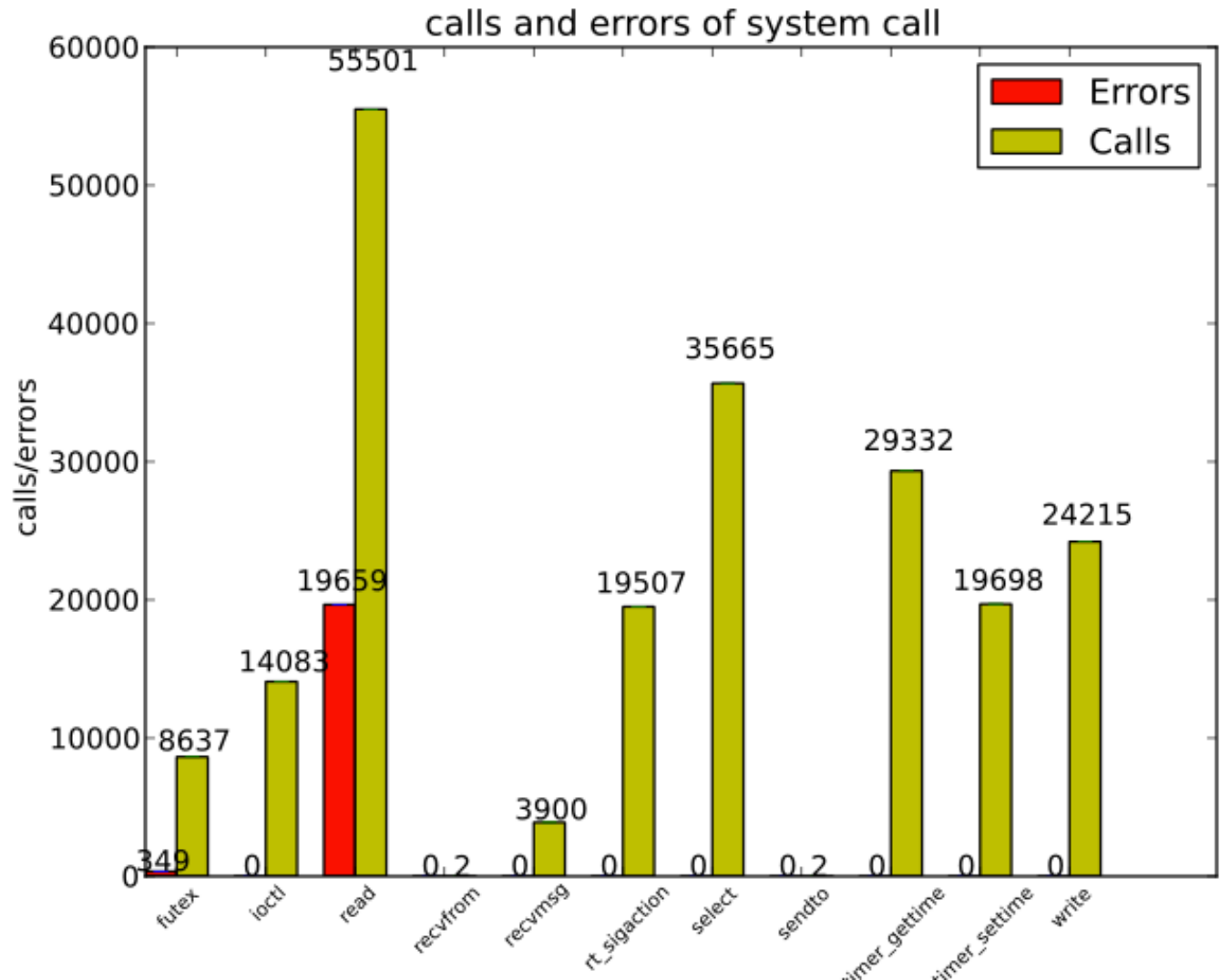
# 01-03\_normal-normal



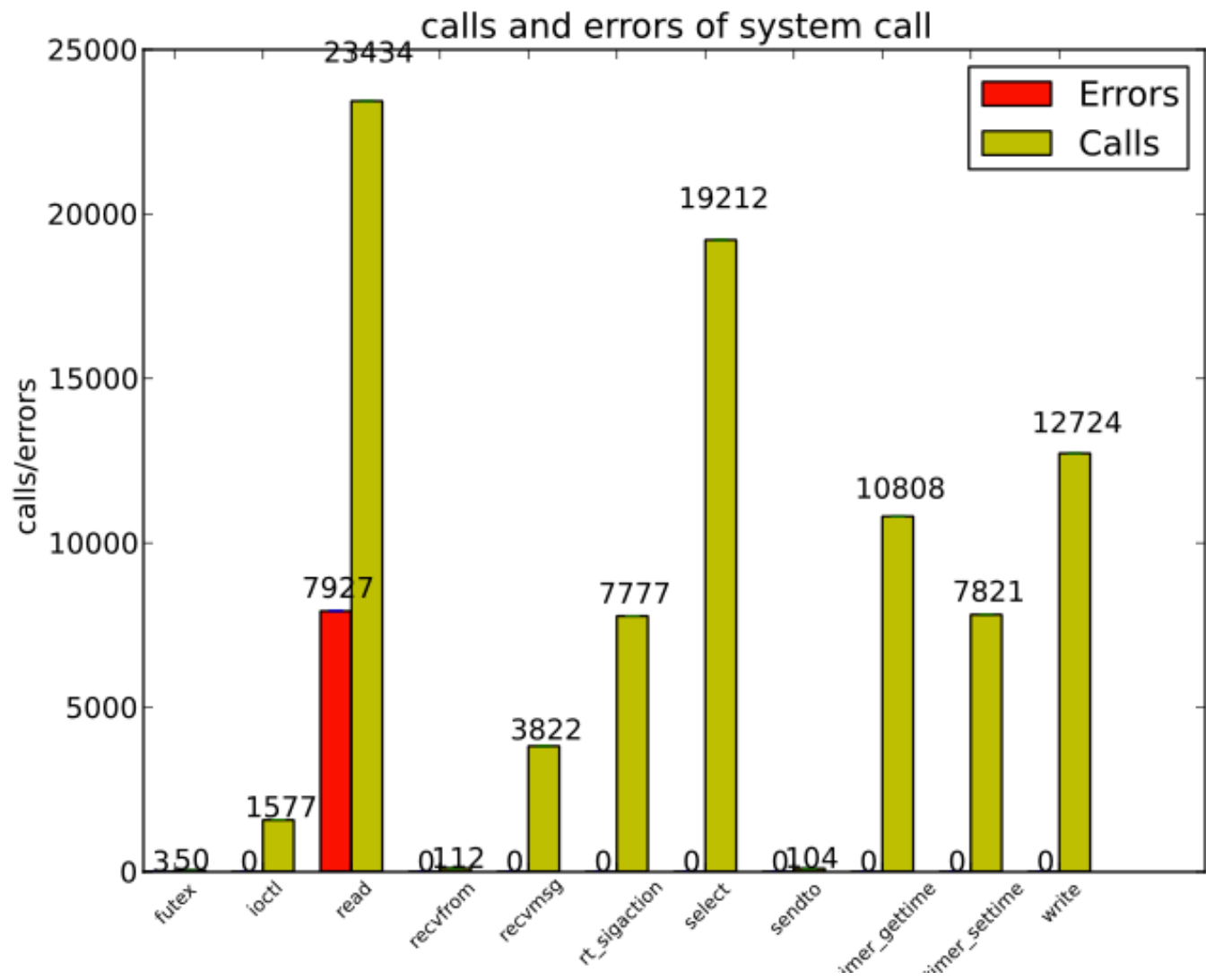
# 02-01\_set-atk-hacker



# 02-02\_set-atk-victim

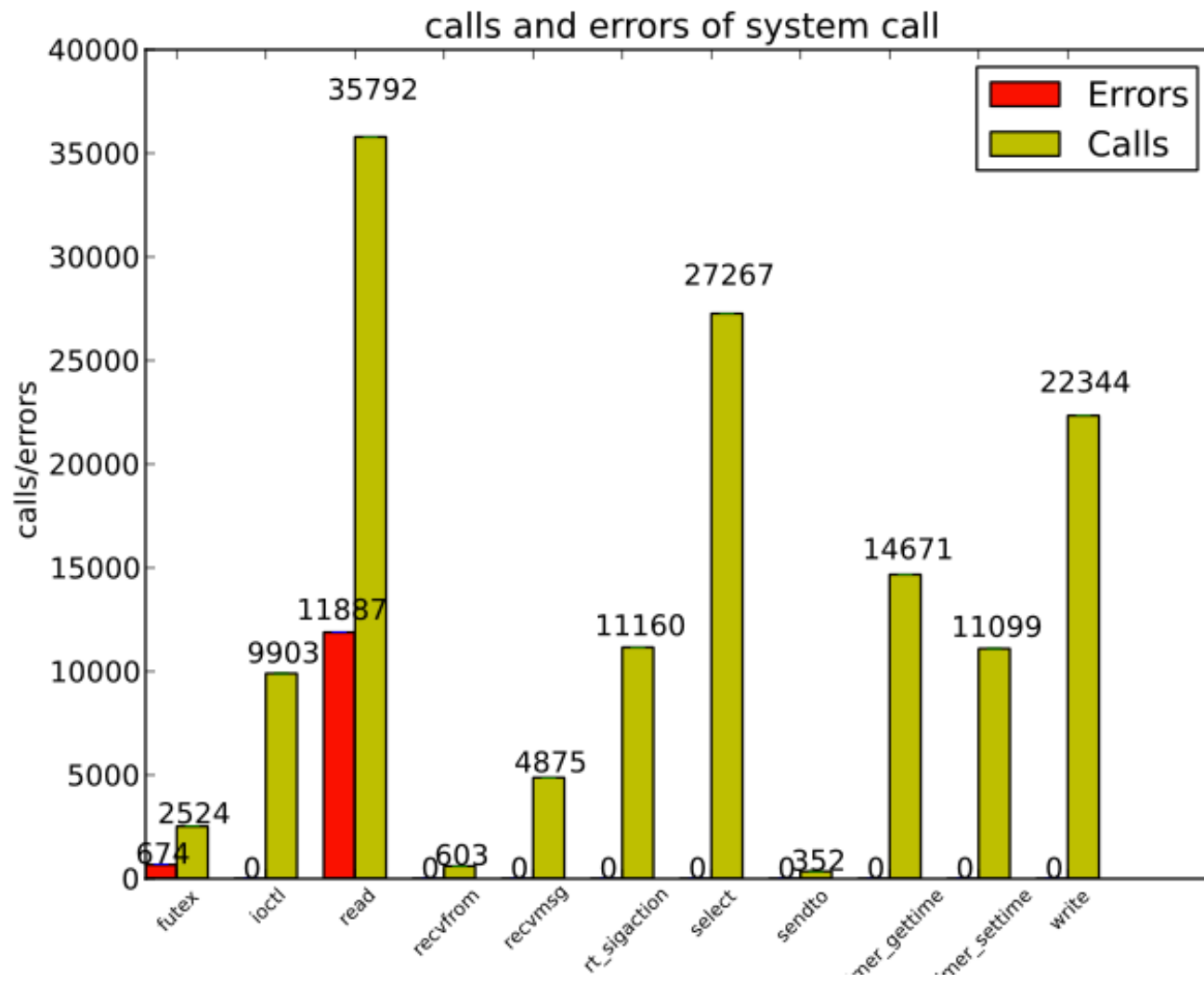


# 02-03\_set-atk-normal

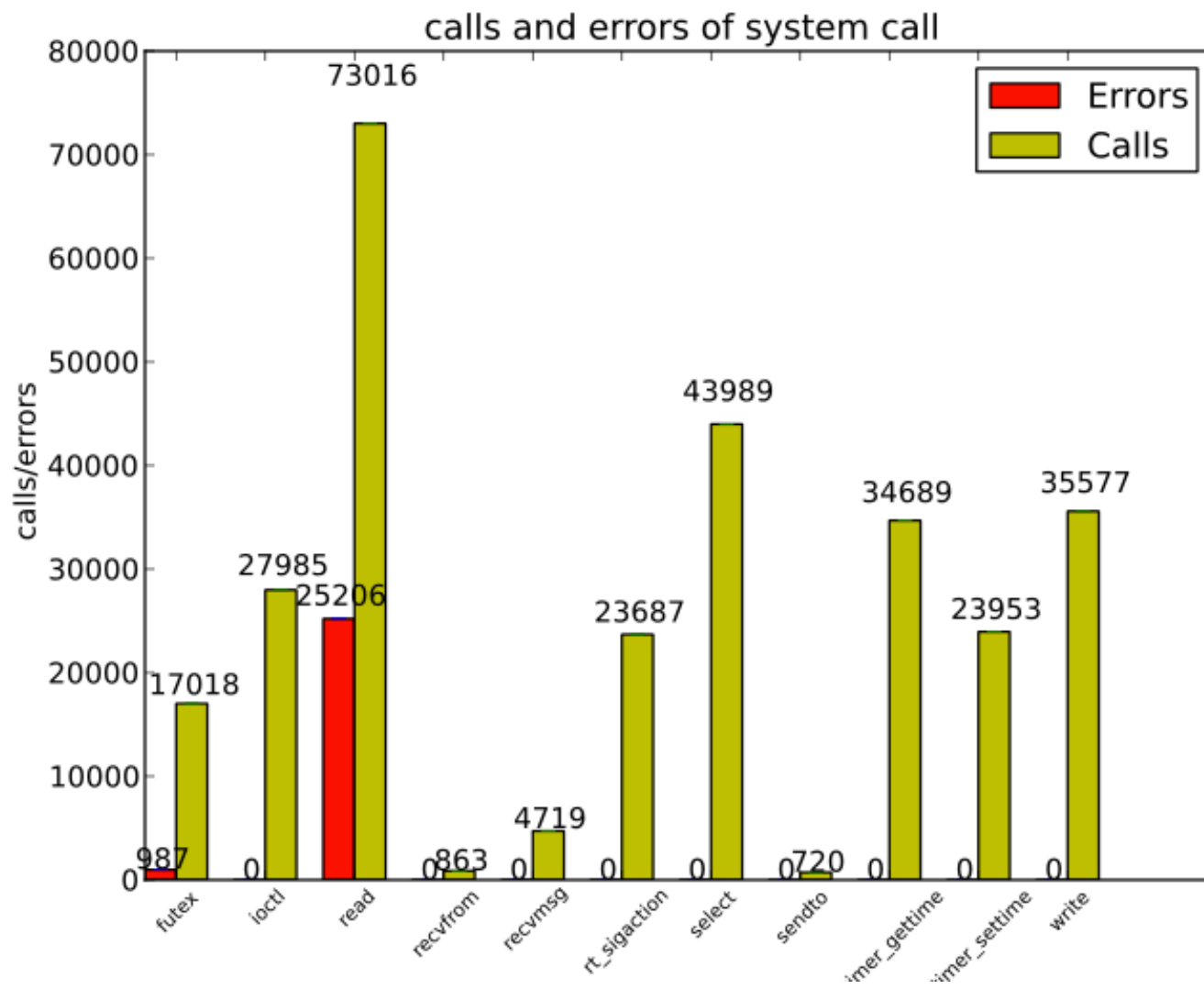




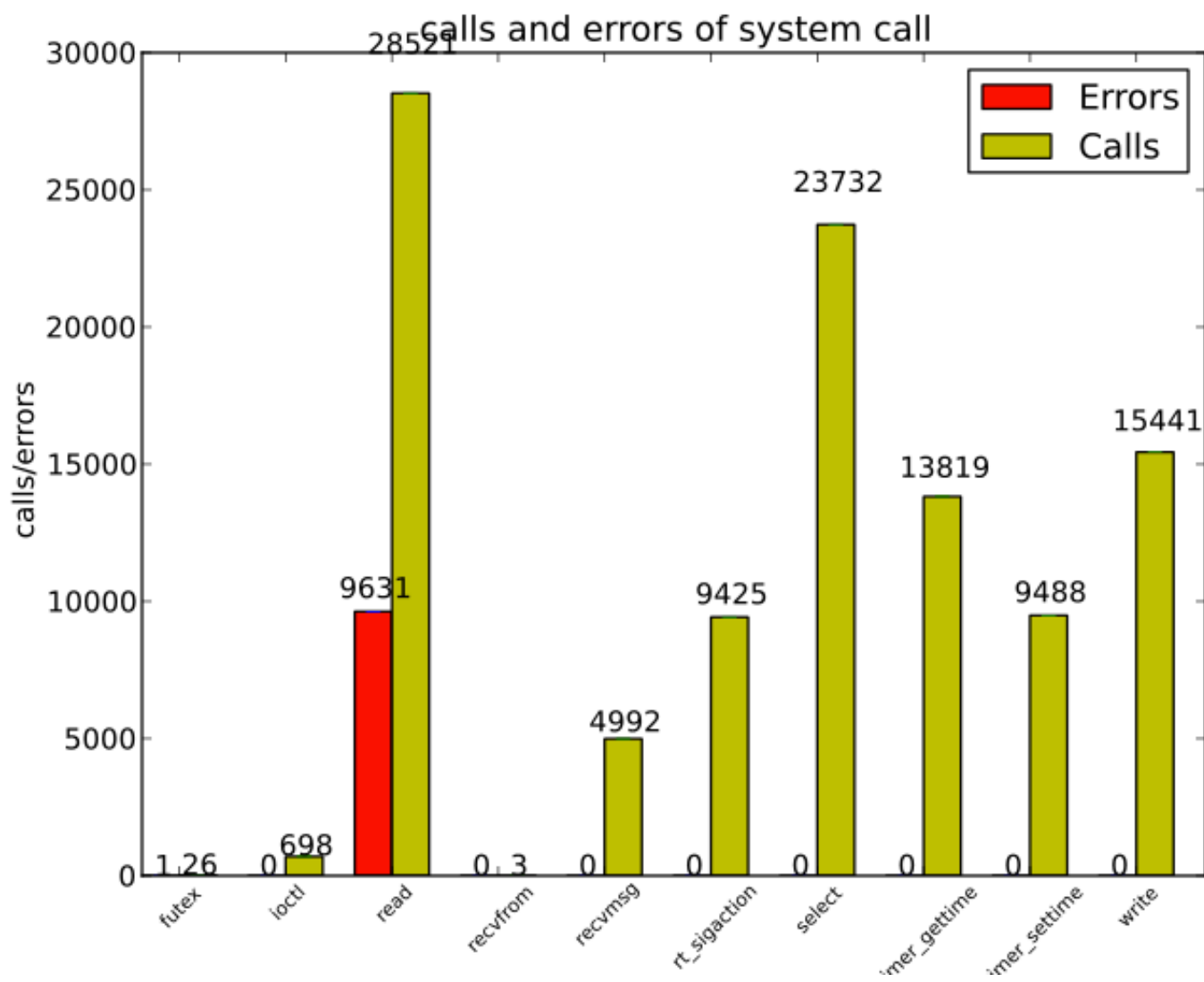
# 03-01\_atking-hacker



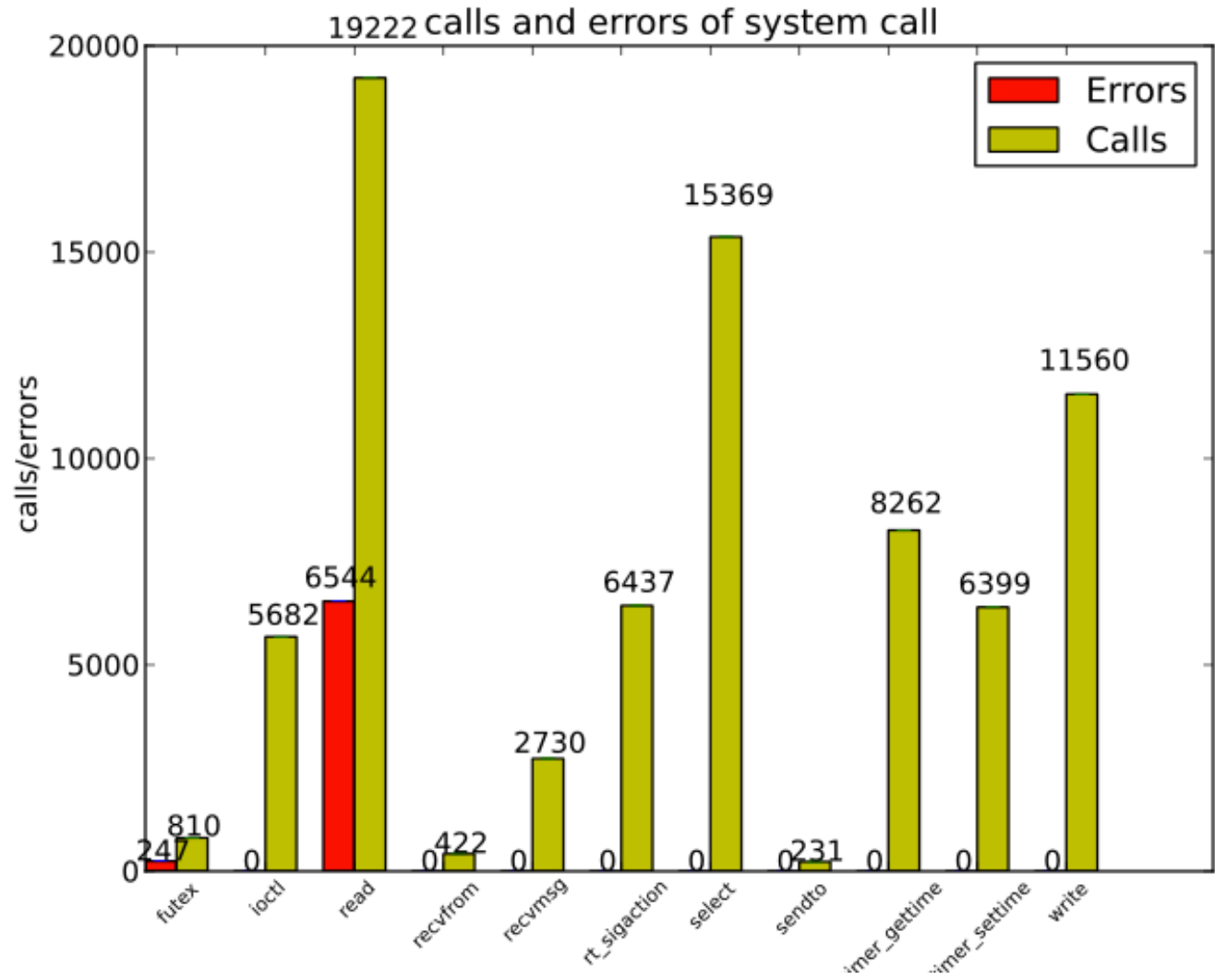
# 03-02\_atking-victim



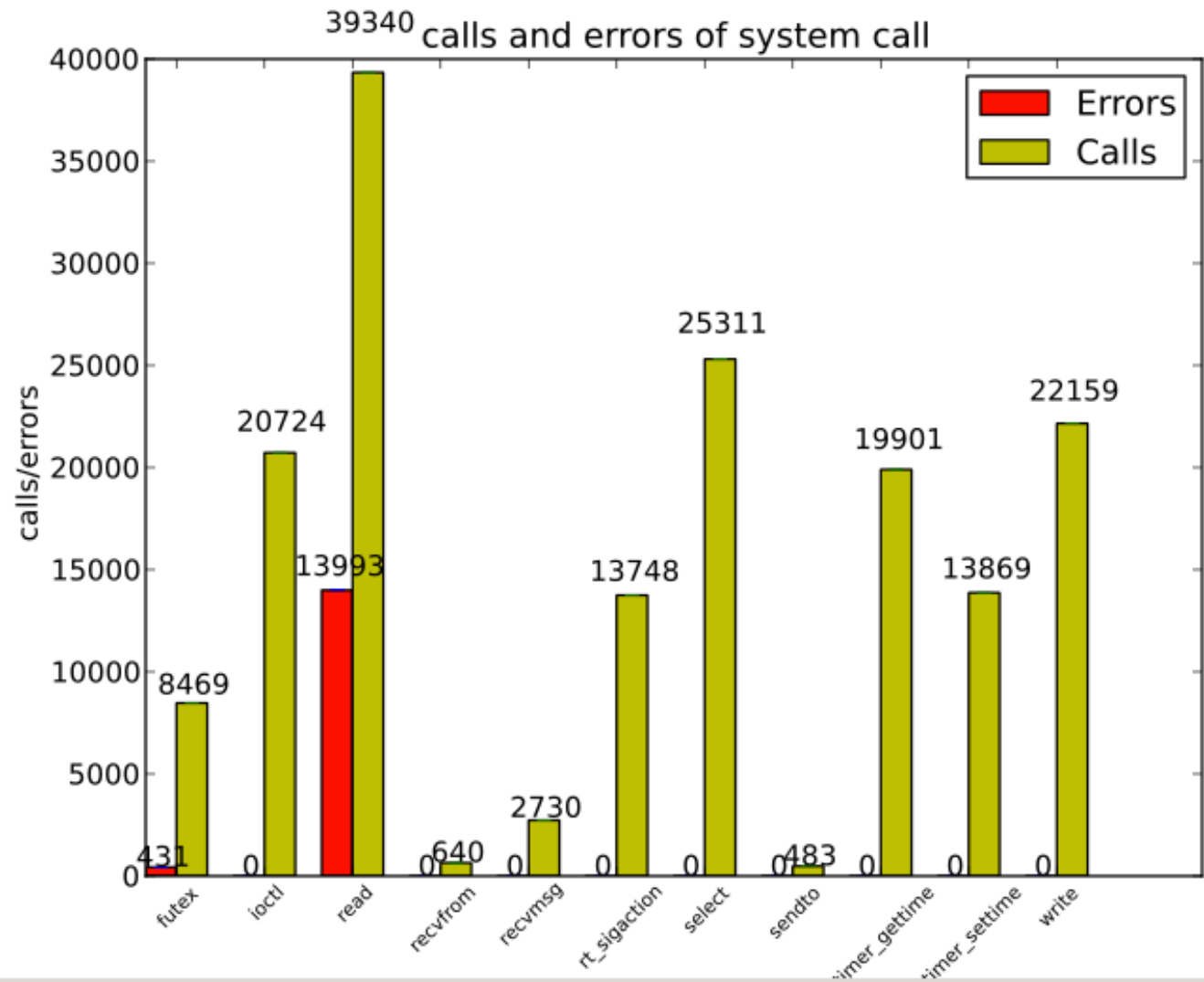
# 03-03\_atking-normal



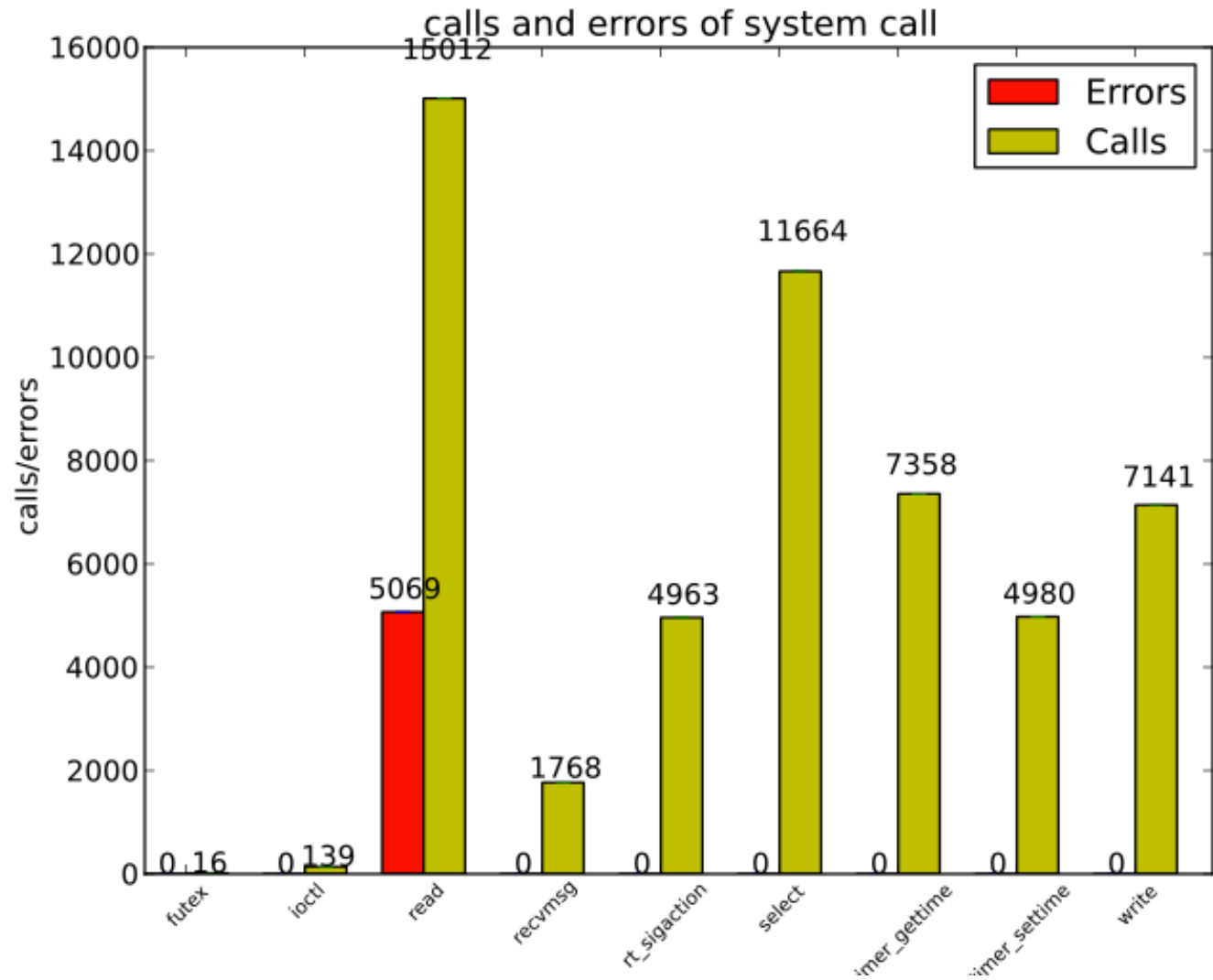
# 04-01\_key\_storke-hacker



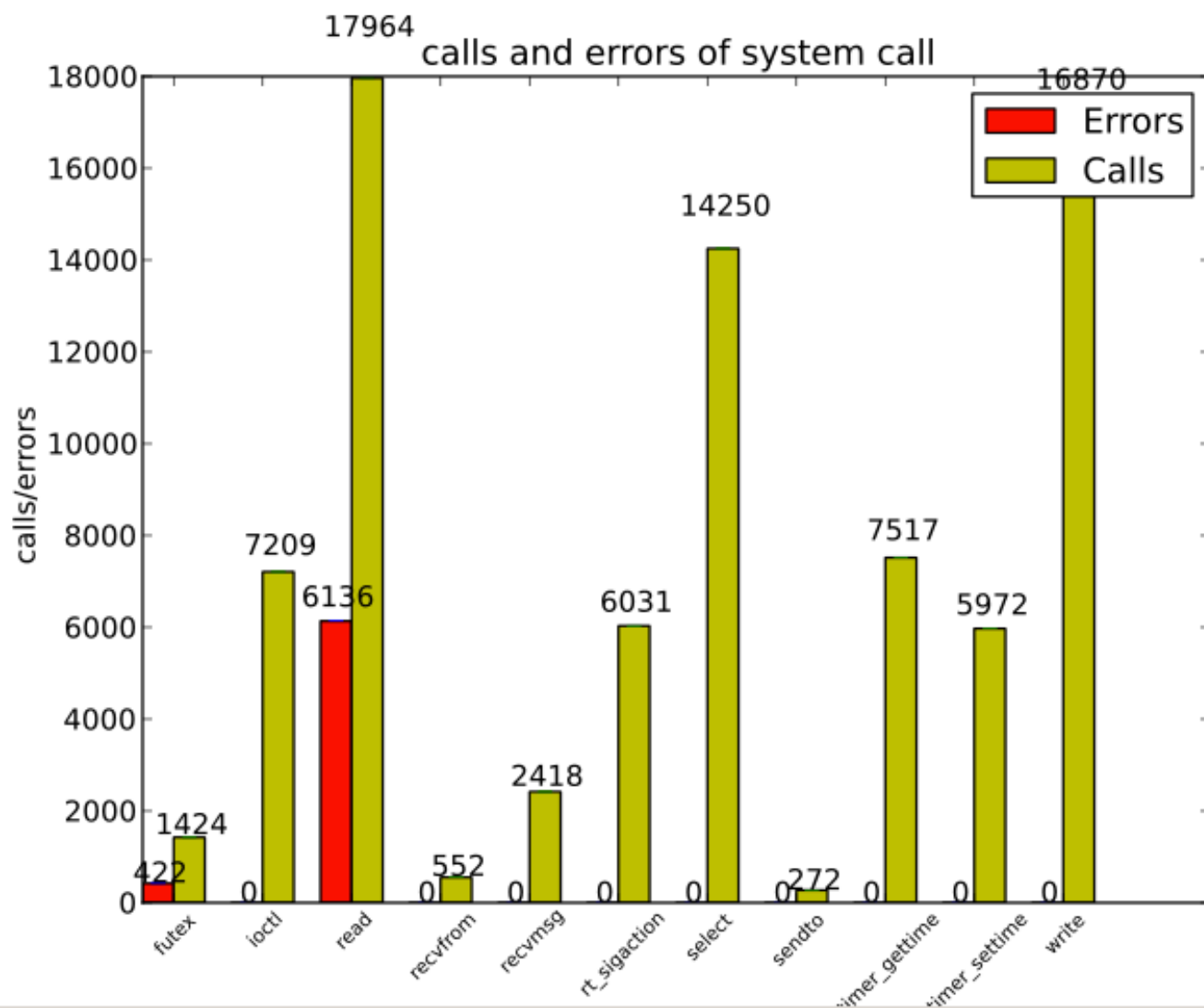
# 04-02\_key\_storke-victim



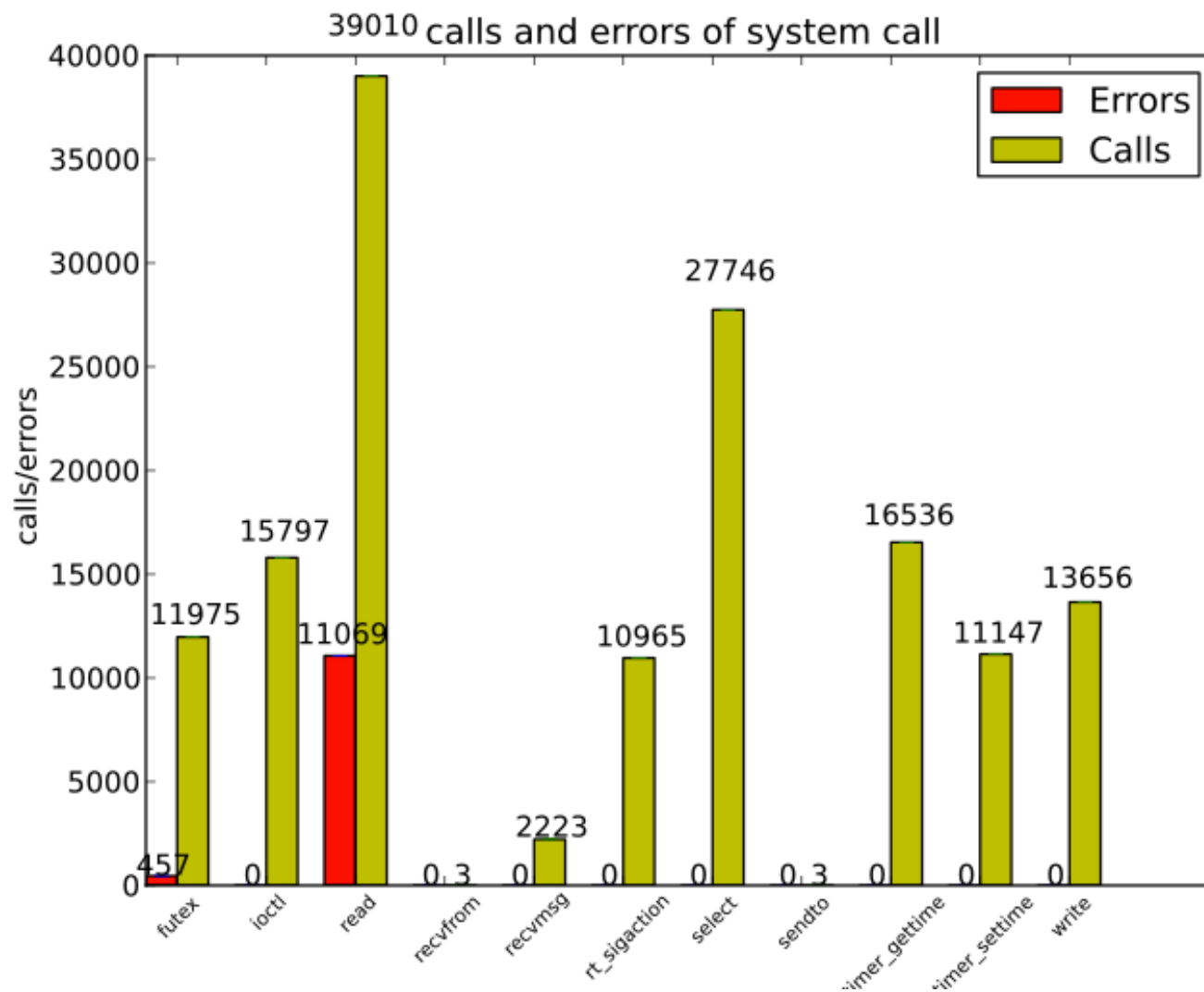
# 04-03\_key\_storke-normal



# 05-01\_kill\_Proc-hacker

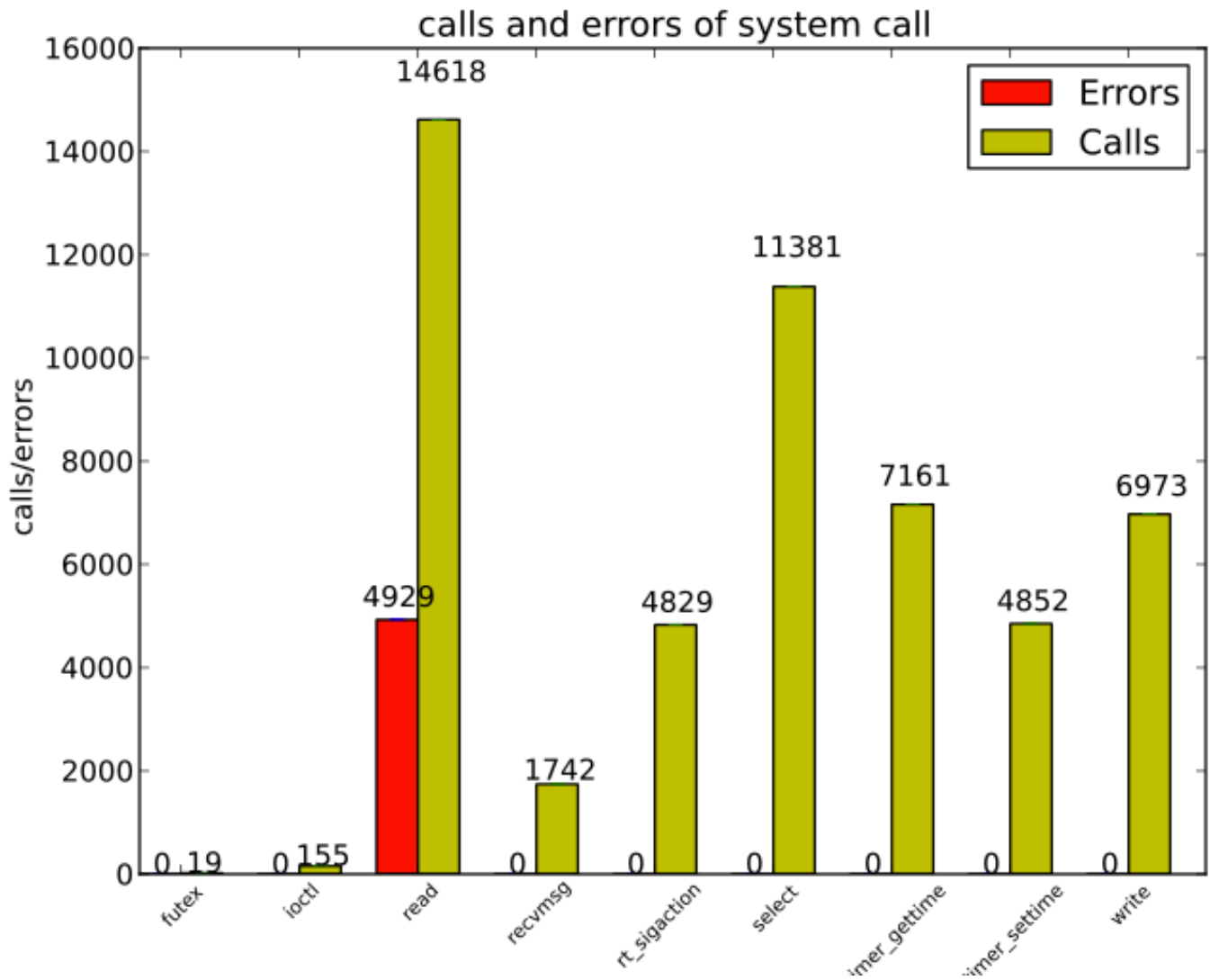


# 05-02\_kill\_Proc-victim





# 05-03\_kill\_Proc-normal



# Clustering result

normal-victim.txt	key_torque-victim.txt set-atk-hacker.txt kill_proc-victim.txt atking-hacker.txt				
normal-normal.txt set-atk-victim.txt normal-hacker.txt atking-victim.txt	<table border="1"><tr><td data-bbox="1025 839 1296 1082">kill_proc-hacker.txt key_torque-hacker.txt</td><td data-bbox="1296 839 1572 1082">key_torque-normal.txt kill_proc-normal.txt</td></tr><tr><td data-bbox="1025 1082 1296 1325">set-atk-normal.txt atking-normal.txt</td><td data-bbox="1296 1082 1572 1325"></td></tr></table>	kill_proc-hacker.txt key_torque-hacker.txt	key_torque-normal.txt kill_proc-normal.txt	set-atk-normal.txt atking-normal.txt	
kill_proc-hacker.txt key_torque-hacker.txt	key_torque-normal.txt kill_proc-normal.txt				
set-atk-normal.txt atking-normal.txt					

# Conclusion

- We propose VIS, a virtualization introspection system for KVM-based cloud platforms
- We monitor both dynamic and static VM status
- We replay and characterize various attacks
  - Detect VMs that attack VM Hypervisor
  - Detect VMs that attack other VMs
  - Detect VMs that are compromised
- VIS can do termination and online migration

# Limitation

- VIS is limited to protection on rules that have been established
  - Need to collect more attack patterns
- The rules are derived by heuristics
  - False positives and negatives
  - Need more sophisticated analysis, e.g., system call sequences

# Q & A

- Thank you for your attention.