# Towards Automatic Inference of Kernel Object Semantics from Binary Code

Junyuan Zeng, and **Zhiqiang Lin**

Department of Computer Science
University of Texas at Dallas

RAID 2015

# Kernel Data Structure (or Object) Semantics

- Concerning the meaning and the behavior of kernel data structures
  - task_struct: process descriptor
  - mm_struct: memory address space descriptor

# Kernel Data Structure (or Object) Semantics

- Concerning the meaning and the behavior of kernel data structures
  - task_struct: process descriptor
  - mm_struct: memory address space descriptor

- Useful for a number of security applications.
  - Virtual machine introspection [GR03]
  - Kernel function reverse engineering

# Why This is Challenging

## Challenges

1. Semantics concern the meaning, which is even vague for human beings.
2. Kernel tends to have a large number of kernel objects.
    - Up to tens of thousands of dynamically created kernel objects.
    - Hundreds of different semantics types.

# Why This is Challenging

## Challenges

1. Semantics concern the meaning, which is even vague for human beings.
2. Kernel tends to have a large number of kernel objects.
   - Up to tens of thousands of dynamically created kernel objects.
   - Hundreds of different semantics types.

## Current Practice

Merely relying on human beings to manually inspect kernel source code, kernel symbols, or kernel APIs to derive and annotate the semantics of the kernel objects.

# Introducing ARGOS

ARGOS: **A**utomatic **R**everse en**G**ineering of kernel **O**bject **S**emantics

# Introducing ARGOS

ARGOS: **A**utomatic **R**everse en**G**ineering of kernel **O**bject **S**emantics

## Key Features

1. Recognizing and uncovering important kernel data structures with semantics, directly from binary code
2. General, working with a variety of (Linux) operating system kernels.

# Introducing ArgOS

ArgOS: **A**utomatic **R**everse en**G**ineering of kernel **O**bject **S**emantics

## Key Features

1. Recognizing and uncovering important kernel data structures with semantics, directly from binary code
2. General, working with a variety of (Linux) operating system kernels.

## Key Principle

**Data use tells data semantics**

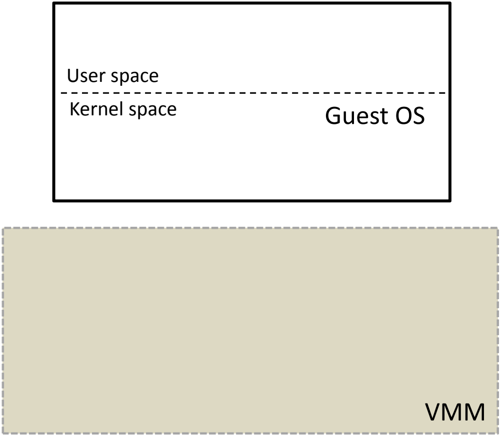
# Key Insights

1. Starting from well-known knowledge
   - User level system call (syscall for short) specification
   - Kernel level exported API specification
2. Using execution context differencing
   - e.g., `task_struct` vs. `mm_struct`
3. Encoding the semantics using a bit-vector
   - Which syscall (e.g., `fork`, `open`, `mmap`) accessed
   - How the object was accessed:

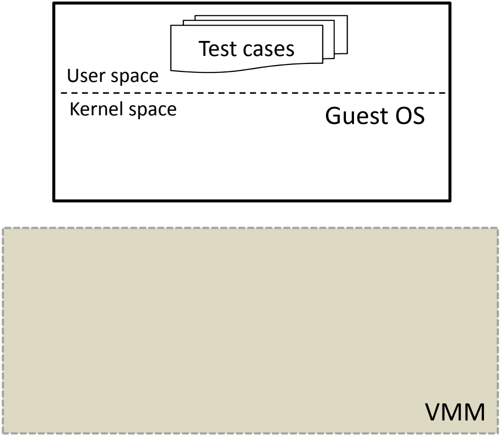
# Key Insights

1. Starting from well-known knowledge
   - User level system call (syscall for short) specification
   - Kernel level exported API specification
2. Using execution context differencing
   - e.g., `task_struct` vs. `mm_struct`
3. Encoding the semantics using a bit-vector
   - Which syscall (e.g., `fork`, `open`, `mmap`) accessed
   - How the object was accessed:
     - read
     - write
     - create
     - destroy

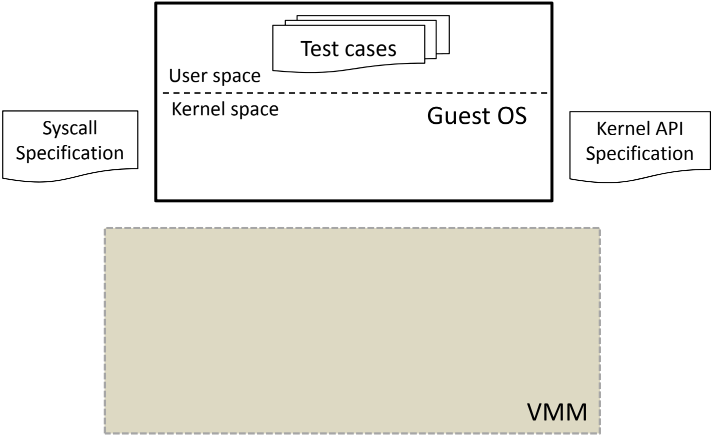
# How ARGOS Works



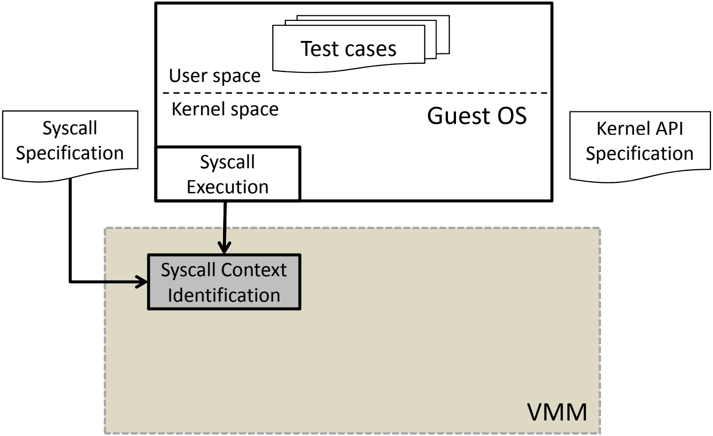
# How ARGOS Works



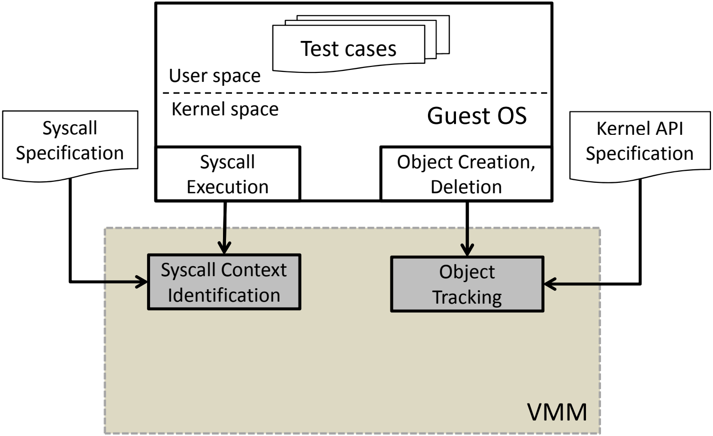
# How ARGOS Works



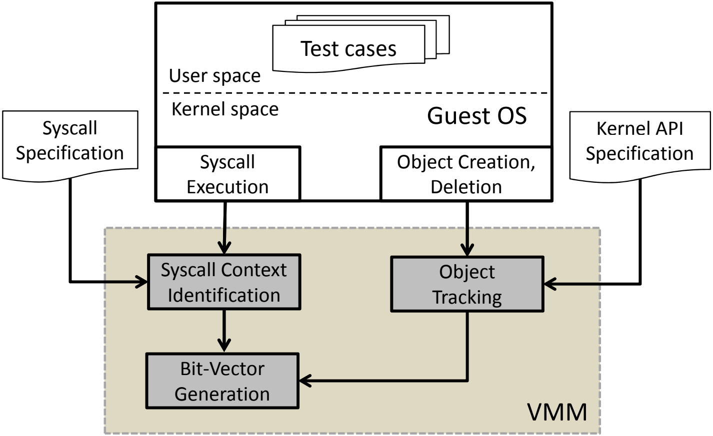
# How ARGOS Works



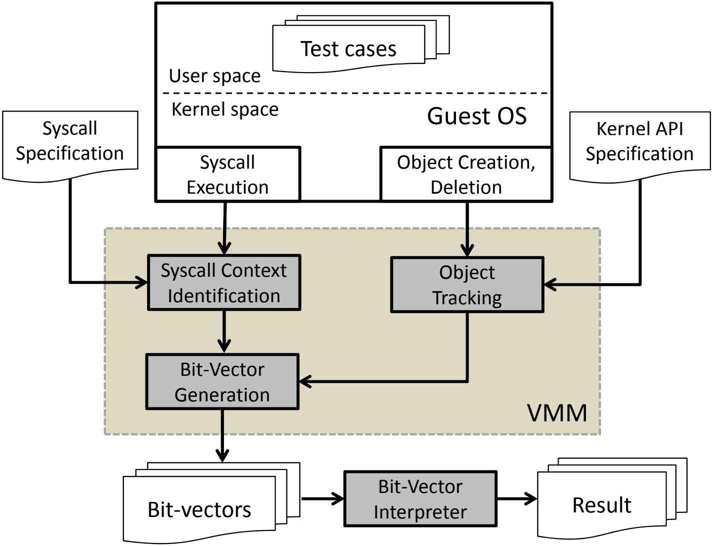
# How ARGOS Works



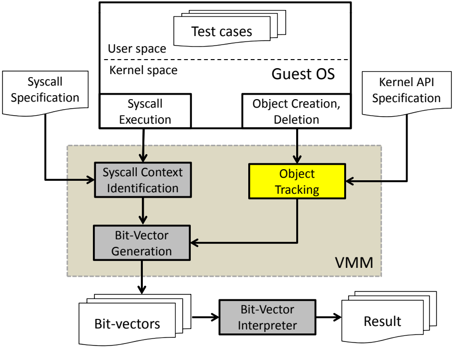
# How ARGOS Works



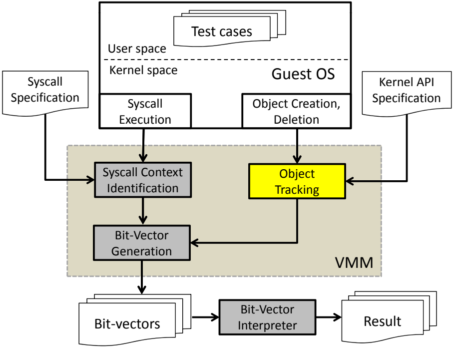
# How ARGOS Works



# Object Tracking



# Object Tracking




1. Tracking the object life time.
2. Assigning a static type to the dynamic object.
3. Tracking the object size.
4. Tracking object relations.

# Object Tracking: Object Life Time

An easy problem by hooking the corresponding kernel APIs

1. Creation
   - `kmem_cache_alloc`
   - `kmalloc`
   - `vmalloc`
2. Deletion
   - `kmem_cache_free`
   - `kfree`
   - `vfree`

# Object Tracking: Object Life Time

An easy problem by hooking the corresponding kernel APIs

1. Creation
   - `kmem_cache_alloc`
   - `kmalloc`
   - `vmalloc`
2. Deletion
   - `kmem_cache_free`
   - `kfree`
   - `vfree`

We will use `kmalloc/kfree` to denote these functions.

# Object Tracking: Assigning a Static Type

## The problem

- What we observe: each **dynamic** data structure (object) instance and their virtual addresses
- What we want: a **static** type associated to each instance

# Object Tracking: Assigning a Static Type

## The problem

- What we observe: each **dynamic** data structure (object) instance and their virtual addresses
- What we want: a **static** type associated to each instance

## Typical approaches

1. Using the call-site-chain from the top callers to `kmalloc` (e.g., $f \rightarrow g \rightarrow h \rightarrow kmalloc$)
   - May over-classify an object type
2. Using the program counter (PC) that invokes `kmalloc` (i.e., $PC_{kmalloc}$)
   - May under-classify an object type (because of wrapper)

# Object Tracking: Assigning a Static Type

## $PC_{kmalloc}$ approach

1. A single kernel object (e.g., `task_struct`) can often be allocated in different calling contexts (e.g., `vfork`, `clone`) $\rightarrow$ over-classify
2. Experimental data
   - 80.3% of the kernel objects have a direct mapping with $PC_{kmalloc}$ approach
   - 97.5% of the objects over-classified with call-chain approach

# Object Tracking: the Object Size

**The problem**

No size argument to many other kernel object allocation functions (e.g., `kmem_cache_alloc`)

# Object Tracking: the Object Size

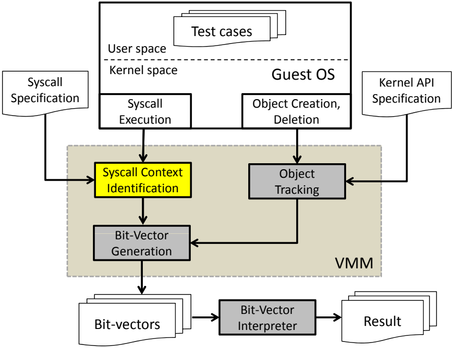
## The problem

No size argument to many other kernel object allocation functions (e.g., `kmem_cache_alloc`)

## Our observation

- Right after executing `kmalloc`, `eax` holds the base address $v$ of the allocated object
- Further access to the field of the object must start from $v$, or the propagation of $v$ (e.g., `mov eax, ebx`) (Taint Analysis)
- By observing how $v$ gets used, we infer the size

# Syscall Context Identification




## Goal

Identify the specific syscall execution context, when a kernel object got accessed.

## Challenges

1. Context switches
2. Interrupts (bottom half, top half)

# Syscall Context Identification

## Observations

1. Tracking `sysenter/int 0x80/sysexit/iret`, and the `eax`
2. Context switches lead to kernel stack (`esp`) exchange
3. Interrupt handler
   - Top half execution (of an interrupt handler) can be identified by `iret`
   - Bottom half execution also has (`esp`) exchange

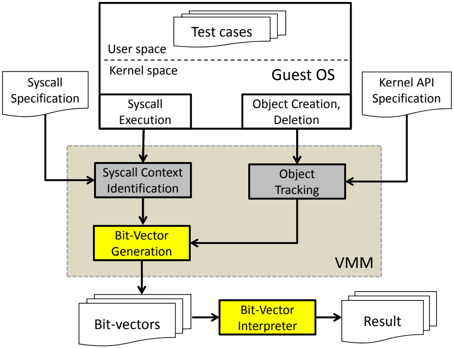
# Syscall Context Identification

## Observations

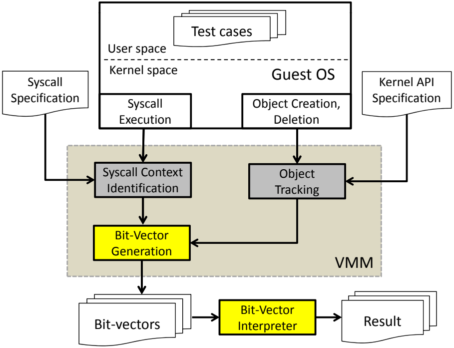
1. Tracking `sysenter/int 0x80/sysexit/iret`, and the `eax`
2. Context switches lead to kernel stack (`esp`) exchange
3. Interrupt handler
   - Top half execution (of an interrupt handler) can be identified by `iret`
   - Bottom half execution also has (`esp`) exchange

By tracking the `sysenter/int 0x80/sysexit/iret` instructions, as well as kernel `esp`, we can uniquely identify kernel syscall context [FL12, FL13]

# Bit-Vector Generation and Interpretation



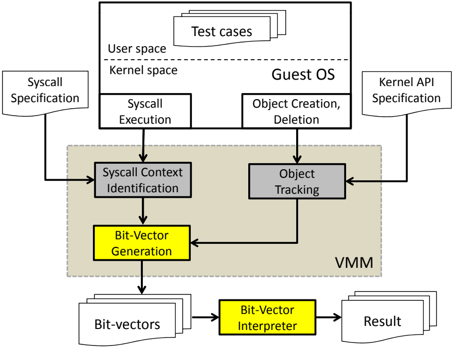
# Bit-Vector Generation and Interpretation




## Goal

Associate the kernel object semantics with the captured execution context

# Bit-Vector Generation and Interpretation




## Goal

Associate the kernel object semantics with the captured execution context

## Challenges

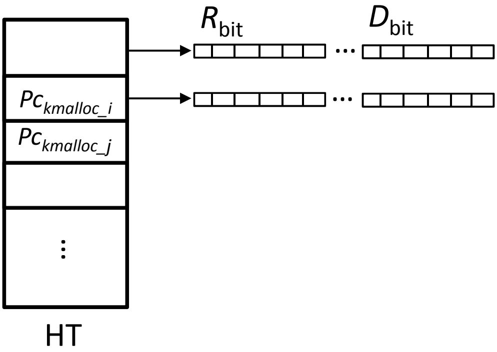
1. How to represent such information (Bit-Vector).
2. How to interpret it (Bit-Vector Interpreter).

# Bit-Vector Generation

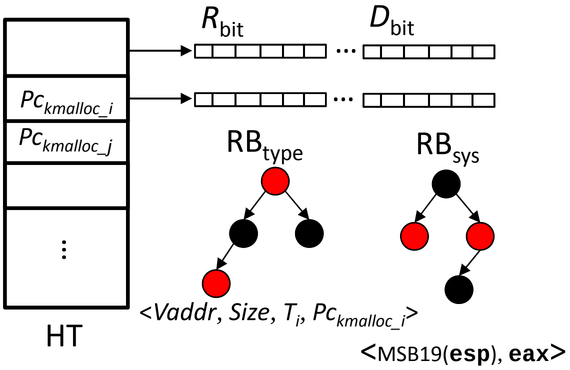
## What information does the Bit-Vector contain

- Each object is associated with one bit-vector of length 4*N where N is the number of syscall.
- For each syscall, four bits are presented
  - *C-bit*: whether this syscall created the object;
  - *R-bit*: whether this syscall read the object;
  - *W-bit*: whether this syscall wrote the object ;
  - *D-bit*: whether this syscall destroyed the object.

# Bit-Vector Generation - All Involved Data Structures



# Bit-Vector Generation - All Involved Data Structures




e.g., `mov %ecx, (%ebx)` → resolve the vaddr of `ebx`, locate the syscall context by using kernel `esp`.

# Bit-Vector Interpreter

## How to interpret Bit-Vector

- Bit-Vector can be viewed as:
  - What are these syscalls that have contributed to the meaning of the object.
  - How these syscalls contributed (recorded in our *R*, *W*, *C*, *D*-bits).

# Bit-Vector Interpreter

## How to interpret Bit-Vector

- Bit-Vector can be viewed as:
  - What are these syscalls that have contributed to the meaning of the object.
  - How these syscalls contributed (recorded in our *R*, *W*, *C*, *D*-bits).

## Current Design

- Deriving the rules based on the general syscall and kernel knowledge.
  - e.g., `task_struct` must be created by `fork`-family syscall, and accessed by `getpid` syscall.

# Experiment Setup

## Experiment Environment

- Guest OS
  - `Linux-2.6.32` with `debian-6.0`
  - `Linux-3.2.58` with `debian-7`
- Host OS: `ubuntu-12.04` with `3.5.0-51-generic`.

## System Input

1. Syscall Specification
2. Kernel API Specification
3. Test Suites:
   - Linux Kernel Test Suite: `ltp-20140115`
   - User Level: `spec2006`, `lmbench-2alpha8`

# Rules to Infer the Semantics

| Rule Num | Detailed Rules | Data Structure |
|---|---|---|
| I | sys_clone[*C*] ∩ sys_getpid[*R*] | task_struct, pid |
| II | ((sys_clone[*C*] - sys_vfork[*C*]) ∩ sys_brk[*RW*]) ∩ sys_munmap[*D*] | vm_area_struct |
| III | ((sys_clone[*C*] - sys_vfork[*C*]) ∩ sys_brk[*RW*]) - sys_munmap[*D*] | mm_struct |
| IV | sys_open[*C*] ∩ sys_lseek[*W*] ∩ sys_dup[*R*] | file |
| V | sys_clone[*C*] - sys_clone[*C*](CLONE_FS) | fs_struct |
| VI | sys_clone[*C*] - sys_clone[*C*](CLONE_FILES) | files_struct |
| VII | sys_mount[*C*] ∩ sys_umount[*D*] | vfs_mount |
| VIII | sys_socketcall[*C*](SYS_SOCKET) ∩ sys_socketcall[*W*] (SYS_SETSOCKOPT) | sock |
| IX | sys_clone[*C*] - sys_clone[*C*](CLONE_SIGHAND) | sighand_struct |
| X | sys_capget[*R*] ∩ sys_capset[*W*] | credential |

# Rules to Infer the Semantics

| Rule Num | Detailed Rules | Data Structure |
|---|---|---|
| I | sys_clone[*C*] ∩ sys_getpid[*R*] | task_struct, pid |
| II | ((sys_clone[*C*] - sys_vfork[*C*]) ∩ sys_brk[*RW*]) ∩ sys_munmap[*D*] | vm_area_struct |
| III | ((sys_clone[*C*] - sys_vfork[*C*]) ∩ sys_brk[*RW*]) - sys_munmap[*D*] | mm_struct |
| IV | sys_open[*C*] ∩ sys_lseek[*W*] ∩ sys_dup[*R*] | file |
| V | sys_clone[*C*] - sys_clone[*C*](CLONE_FS) | fs_struct |
| VI | sys_clone[*C*] - sys_clone[*C*](CLONE_FILES) | files_struct |
| VII | sys_mount[*C*] ∩ sys_umount[*D*] | vfs_mount |
| VIII | sys_socketcall[*C*](SYS_SOCKET) ∩ sys_socketcall[*W*] (SYS_SETSOCKOPT) | sock |
| IX | sys_clone[*C*] - sys_clone[*C*](CLONE_SIGHAND) | sighand_struct |
| X | sys_capget[*R*] ∩ sys_capset[*W*] | credential |

# Statistics of the Bit-Vector

| Rule Num | Kernel Version | Symbol Name | Traced Size | Statistics of the R/W Bit Vector | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | P | F | M | T | G | S | N | I | D | O |
| I | 2.6.32 | pid | 44 | 25 | 16 | 4 | 0 | 3 | 0 | 1 | 3 | 1 | 0 |
| | | task_struct | 1072 | 47 | 48 | 5 | 0 | 12 | 0 | 1 | 1 | 2 | 0 |
| | 3.2.58 | pid | 64 | 28 | 24 | 3 | 0 | 3 | 0 | 1 | 3 | 1 | 0 |
| | | task_struct | 1072 | 73 | 109 | 13 | 6 | 19 | 1 | 2 | 7 | 2 | 0 |
| II | 2.6.32 | vm_area_struct | 88 | 4 | 17 | 12 | 0 | 3 | 0 | 0 | 1 | 1 | 0 |
| | 3.2.58 | vm_area_struct | 88 | 3 | 5 | 12 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| III | 2.6.32 | mm_struct | 420 | 15 | 6 | 5 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| | 3.2.58 | mm_struct | 448 | 15 | 9 | 6 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| IV | 2.6.32 | file | 128 | 41 | 93 | 12 | 0 | 10 | 0 | 1 | 7 | 2 | 0 |
| | 3.2.58 | file | 160 | 35 | 97 | 12 | 0 | 11 | 0 | 1 | 7 | 2 | 0 |
| V | 2.6.32 | fs_struct | 32 | 4 | 50 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| | 3.2.58 | fs_struct | 64 | 4 | 51 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| VI | 2.6.32 | files_struct | 224 | 11 | 73 | 3 | 0 | 4 | 0 | 1 | 6 | 1 | 0 |
| | 3.2.58 | files_struct | 256 | 39 | 84 | 5 | 0 | 6 | 0 | 1 | 6 | 1 | 0 |
| VII | 2.6.32 | vfs_mount | 128 | 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | 3.2.58 | vfs_mount | 160 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| VII | 2.6.32 | sock | 1216 | 19 | 55 | 8 | 0 | 9 | 1 | 6 | 6 | 2 | 0 |
| | 3.2.58 | sock | 1248 | 28 | 74 | 7 | 0 | 9 | 1 | 1 | 6 | 2 | 0 |
| IX | 2.6.32 | sighand_struct | 1288 | 15 | 5 | 0 | 0 | 12 | 0 | 1 | 1 | 1 | 0 |
| | 3.2.58 | sighand_struct | 1312 | 15 | 7 | 0 | 0 | 12 | 0 | 1 | 1 | 1 | 0 |
| X | 2.6.32 | cred | 128 | 51 | 72 | 8 | 3 | 3 | 1 | 2 | 4 | 2 | 0 |
| | 3.2.58 | cred | 128 | 53 | 75 | 7 | 3 | 2 | 1 | 2 | 4 | 2 | 0 |

# Statistics of the Bit-Vector

| Rule Num | Kernel Version | Symbol Name | Traced Size | Statistics of the R/W Bit Vector | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | P | F | M | T | G | S | N | I | D | O |
| I | 2.6.32 | pid | 44 | 25 | 16 | 4 | 0 | 3 | 0 | 1 | 3 | 1 | 0 |
| | | task_struct | 1072 | 47 | 48 | 5 | 0 | 12 | 0 | 1 | 1 | 2 | 0 |
| | 3.2.58 | pid | 64 | 28 | 24 | 3 | 0 | 3 | 0 | 1 | 3 | 1 | 0 |
| | | task_struct | 1072 | 73 | 109 | 13 | 6 | 19 | 1 | 2 | 7 | 2 | 0 |
| II | 2.6.32 | vm_area_struct | 88 | 4 | 17 | 12 | 0 | 3 | 0 | 0 | 1 | 1 | 0 |
| | 3.2.58 | vm_area_struct | 88 | 3 | 5 | 12 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| III | 2.6.32 | mm_struct | 420 | 15 | 6 | 5 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| | 3.2.58 | mm_struct | 448 | 15 | 9 | 6 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| IV | 2.6.32 | file | 128 | 41 | 93 | 12 | 0 | 10 | 0 | 1 | 7 | 2 | 0 |
| | 3.2.58 | file | 160 | 35 | 97 | 12 | 0 | 11 | 0 | 1 | 7 | 2 | 0 |
| V | 2.6.32 | fs_struct | 32 | 4 | 50 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| | 3.2.58 | fs_struct | 64 | 4 | 51 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| VI | 2.6.32 | files_struct | 224 | 11 | 73 | 3 | 0 | 4 | 0 | 1 | 6 | 1 | 0 |
| | 3.2.58 | files_struct | 256 | 39 | 84 | 5 | 0 | 6 | 0 | 1 | 6 | 1 | 0 |
| VII | 2.6.32 | vfs_mount | 128 | 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| | 3.2.58 | vfs_mount | 160 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| VII | 2.6.32 | sock | 1216 | 19 | 55 | 8 | 0 | 9 | 1 | 6 | 6 | 2 | 0 |
| | 3.2.58 | sock | 1248 | 28 | 74 | 7 | 0 | 9 | 1 | 1 | 6 | 2 | 0 |
| IX | 2.6.32 | sighand_struct | 1288 | 15 | 5 | 0 | 0 | 12 | 0 | 1 | 1 | 1 | 0 |
| | 3.2.58 | sighand_struct | 1312 | 15 | 7 | 0 | 0 | 12 | 0 | 1 | 1 | 1 | 0 |
| X | 2.6.32 | cred | 128 | 51 | 72 | 8 | 3 | 3 | 1 | 2 | 4 | 2 | 0 |
| | 3.2.58 | cred | 128 | 53 | 75 | 7 | 3 | 2 | 1 | 2 | 4 | 2 | 0 |

# The Syscall Classification

| Syscall Type | Short Name | #Syscalls Linux-2.6.32 | #Syscalls Linux-3.2.58 |
|---|---|---|---|
| Process | P | 90 | 92 |
| File | F | 152 | 156 |
| Memory | M | 19 | 21 |
| Time | T | 13 | 13 |
| Signal | G | 25 | 25 |
| Security | S | 3 | 3 |
| Network | N | 2 | 4 |
| IPC | I | 7 | 7 |
| Module | D | 4 | 4 |
| Other | O | 3 | 3 |
| Total | - | 317 | 328 |

# The Syscall Classification

| Syscall Type | Short Name | #Syscalls Linux-2.6.32 | #Syscalls Linux-3.2.58 |
|---|---|---|---|
| Process | P | 90 | 92 |
| File | F | 152 | 156 |
| Memory | M | 19 | 21 |
| Time | T | 13 | 13 |
| Signal | G | 25 | 25 |
| Security | S | 3 | 3 |
| Network | N | 2 | 4 |
| IPC | I | 7 | 7 |
| Module | D | 4 | 4 |
| Other | O | 3 | 3 |
| Total | - | 317 | 328 |

# Application: Inference of Kernel Internal Functions

| Type | Version | Creation Function | | Deletion Function | |
|---|---|---|---|---|---|
| | | PC | Symbol | PC | Symbol |
| pid | 2.6.32 | c10414d0 | alloc_pid | c10413de | put_pid |
| | 3.2.58 | c104bb02 | alloc_pid | c104b969 | put_pid |
| task_struct | 2.6.32 | c102daaf | copy_process | c102da55 | free_task |
| | 3.2.58 | c103719d | copy_process | c10368a7 | free_task |
| vm_area_struct | 2.6.32 | c102d730 | dup_mm | c109d387 | remove_vma |
| | 3.2.58 | c1036d97 | dup_mm | c10b13d7 | remove_vma |
| mm_struct | 2.6.32 | c102d730 | dup_mm | c102d3dc | __mmdrop |
| | 3.2.58 | c1036d97 | dup_mm | c1036a58 | __mmdrop |
| file | 2.6.32 | c10b230d | get_empty_filp | c10b2030 | file_free_rcu |
| | 3.2.58 | c10cee78 | get_empty_filp | c10ceba0 | file_free_rcu |
| fs_struct | 2.6.32 | c10cac50 | copy_fs_struct | c10cae5b | free_fs_struct |
| | 3.2.58 | c10eaac4 | copy_fs_struct | c10eaa55 | free_fs_struct |
| files_struct | 2.6.32 | c10c1839 | dup_fd | c1030a32 | put_files_struct |
| | 3.2.58 | c10df2ab | dup_fd | c103b16d | put_files_struct |
| vfs_mount | 2.6.32 | c10c3a35 | alloc_vfsmnt | c10c30ba | free_vfsmnt |
| | 3.2.58 | c10dfd23 | alloc_vfsmnt | c10dfe36 | free_vfsmnt |
| sighand_struct | 2.6.32 | c102daaf | copy_process | c102d148 | __cleanup_sighand |
| | 3.2.58 | c103719d | copy_process | c103717b | __cleanup_sighand |
| sock | 2.6.32 | c11cd7a5 | sk_prot_alloc | c11cc884 | __sk_free |
| | 3.2.58 | c12146e5 | sk_prot_alloc | c1214d46 | __sk_free |
| cred | 2.6.32 | c1047923 | prepare_creds | c1047d00 | put_cred_rcu |
| | 3.2.58 | c10525fe | prepare_creds | c105239b | put_cred_rcu |

# Application: Inference of Kernel Internal Functions

| Type | Version | Creation Function | | Deletion Function | |
|---|---|---|---|---|---|
| | | PC | Symbol | PC | Symbol |
| pid | 2.6.32 | c10414d0 | alloc_pid | c10413de | put_pid |
| | 3.2.58 | c104bb02 | alloc_pid | c104b969 | put_pid |
| task_struct | 2.6.32 | c102daaf | copy_process | c102da55 | free_task |
| | 3.2.58 | c103719d | copy_process | c10368a7 | free_task |
| vm_area_struct | 2.6.32 | c102d730 | dup_mm | c109d387 | remove_vma |
| | 3.2.58 | c1036d97 | dup_mm | c10b13d7 | remove_vma |
| mm_struct | 2.6.32 | c102d730 | dup_mm | c102d3dc | __mmdrop |
| | 3.2.58 | c1036d97 | dup_mm | c1036a58 | __mmdrop |
| file | 2.6.32 | c10b230d | get_empty_filp | c10b2030 | file_free_rcu |
| | 3.2.58 | c10cee78 | get_empty_filp | c10ceba0 | file_free_rcu |
| fs_struct | 2.6.32 | c10cac50 | copy_fs_struct | c10cae5b | free_fs_struct |
| | 3.2.58 | c10eaac4 | copy_fs_struct | c10eaa55 | free_fs_struct |
| files_struct | 2.6.32 | c10c1839 | dup_fd | c1030a32 | put_files_struct |
| | 3.2.58 | c10df2ab | dup_fd | c103b16d | put_files_struct |
| vfs_mount | 2.6.32 | c10c3a35 | alloc_vfsmnt | c10c30ba | free_vfsmnt |
| | 3.2.58 | c10dfd23 | alloc_vfsmnt | c10dfe36 | free_vfsmnt |
| sighand_struct | 2.6.32 | c102daaf | copy_process | c102d148 | __cleanup_sighand |
| | 3.2.58 | c103719d | copy_process | c103717b | __cleanup_sighand |
| sock | 2.6.32 | c11cd7a5 | sk_prot_alloc | c11cc884 | __sk_free |
| | 3.2.58 | c12146e5 | sk_prot_alloc | c1214d46 | __sk_free |
| cred | 2.6.32 | c1047923 | prepare_creds | c1047d00 | put_cred_rcu |
| | 3.2.58 | c10525fe | prepare_creds | c105239b | put_cred_rcu |

# Limitation and Future Work

1. Only semantics, no syntax (the layout, field)
2. Unable to track the inlined `kmalloc` execution
3. Only demonstrated our techniques for Linux Kernel
4. ...

# Related Work on Data Structure Analysis

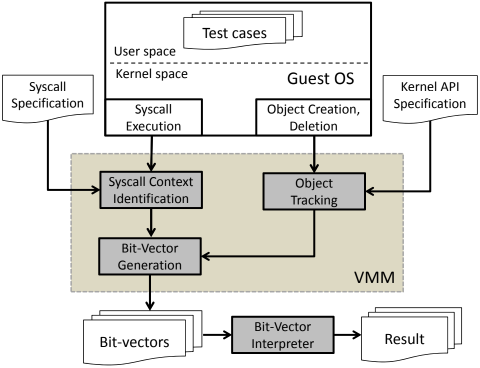
## Static Analysis

1. **Aggregate structure identification (ASI)** [RFT99]
2. **Value set analysis (VSA)** [BR04, RB08]
3. TIE [LAB11]

## Dynamic Analysis

1. **Protocol Reverse Engineering**: Polyglot [CS07], AutoFormat [LJXZ08], ANP [WMKK08], Tupni [CPC+08], ReFromat [WJC+09], Dispatcher [CPKS09]
2. **Data Structure Reverse Engineering**: Rewards [LZX10], Howard [SSB11], PointerScope [ZPL+12], Laika [CSXK08]

# Summary: ARGOS




1. The first system to infer kernel object semantics
2. Starting from syscall and kernel API knowledge
3. Tracking the instruction execution and using bit-vector
4. Evaluated w/ Linux kernel

# Thank you

# References I


- Gogul Balakrishnan and Thomas Reps, Analyzing memory accesses in x86 executables, CC, Mar. 2004.
- Weidong Cui, Marcus Peinado, Karl Chen, Helen J. Wang, and Luis Irun-Briz, Tupni: Automatic reverse engineering of input formats, Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08) (Alexandria, Virginia, USA), October 2008, pp. 391–402.
- Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song, Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering, Proceedings of the 16th ACM Conference on Computer and and Communications Security (CCS'09) (Chicago, Illinois, USA), 2009, pp. 621–634.
- Juan Caballero and Dawn Song, Polyglot: Automatic extraction of protocol format using dynamic binary analysis, Proceedings of the 14th ACM Conference on Computer and and Communications Security (CCS'07) (Alexandria, Virginia, USA), 2007, pp. 317–329.
- Anthony Cozzie, Frank Stratton, Hui Xue, and Samuel T. King, Digging for data structures, Proceeding of 8th Symposium on Operating System Design and Implementation (OSDI'08) (San Diego, CA), December, 2008, pp. 231–244.
- Yangchun Fu and Zhiqiang Lin, Space traveling across vm: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection, Proceedings of $33^{rd}$ IEEE Symposium on Security and Privacy, May 2012.
- ______, Exterior: Using a dual-vm based external shell for guest-os introspection, configuration, and recovery, Proceedings of the Ninth Annual International Conference on Virtual Execution Environments (Houston, TX), March 2013.

# References II


- Tal Garfinkel and Mendel Rosenblum, A virtual machine introspection based architecture for intrusion detection, Proceedings Network and Distributed Systems Security Symposium (NDSS'03), February 2003, pp. 38–53.
- JongHyup Lee, Thanassis Avgerinos, and David Brumley, Tie: Principled reverse engineering of types in binary programs, Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS'11) (San Diego, CA), February 2011.
- Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang, Automatic protocol format reverse engineering through context-aware monitored execution, Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08) (San Diego, CA), February 2008.
- Zhiqiang Lin, Xiangyu Zhang, and Dongyan Xu, Automatic reverse engineering of data structures from binary execution, Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS'10) (San Diego, CA), February 2010.
- Thomas W. Reps and Gogul Balakrishnan, Improved memory-access analysis for x86 executables, Proceedings of International Conference on Compiler Construction (CC'08), 2008, pp. 16–35.
- G. Ramalingam, John Field, and Frank Tip, Aggregate structure identification and its application to program analysis, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of programming languages (POPL'99) (San Antonio, Texas), ACM, 1999, pp. 119–132.
- Asia Slowinska, Traian Stancescu, and Herbert Bos, Howard: A dynamic excavator for reverse engineering data structures, Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS'11) (San Diego, CA), February 2011.

# References III


- Zhi Wang, Xuxian Jiang, Weidong Cui, Xinyuan Wang, and Mike Grace, Reformat: Automatic reverse engineering of encrypted messages, Proceedings of the 14th European Conference on Research in Computer Security (Saint-Malo, France), ESORICS'09, Springer-Verlag, 2009, pp. 200–215.
- Gilbert Wondracek, Paolo Milani, Christopher Kruegel, and Engin Kirda, Automatic network protocol analysis, Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08) (San Diego, CA), February 2008.
- Mingwei Zhang, Aravind Prakash, Xiaolei Li, Zhenkai Liang, and Heng Yin, Identifying and analyzing pointer misuses for sophisticated memory-corruption exploit diagnosis, Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS'12) (San Diego, CA), February 2012.