Multi-Aspect Profiling of Kernel Rootkit Behavior

Ryan Riley, Xuxian Jiang, Dongyan Xu
Purdue University, North Carolina State University

EuroSys 2009
Nürnberg, Germany
Rootkits

- Stealthy malware
- Hide attacker
- Modifying the OS kernel in memory
- Injecting new code

- Threat model:
  - “Root” privileges
  - Full memory access
In the news…

Cyberspeak

Sony: The rootkit of all evil?
Rootkit techniques
Rootkit techniques

**adore-ng**

- Linux 2.4/2.6
- Kernel module
- Adds "custom" functions
Rootkit techniques

adore-ng
- Linux 2.4/2.6
- Kernel module
- Adds “custom” functions

hp
- Linux 2.4
- Kernel module
- Modifies kernel objects
Profiling a rootkit?

• Quickly reveal behavior
• Tool for malware investigators
• Honeypot environment
• This is hard, rootkits are highly privileged!
Profiling: Determining behavior

1. What code does it run?

2. What kernel objects does it modify?

3. How does it modify control flow?

4. What system calls are affected at user-level?
PoKeR: Architecture

Virtual Machine

User-level Applications
Guest Kernel

Right-Before Detection

Virtual Machine Monitor

Logging and Context Tracking

Log

Kernel Symbols & Kernel Object Types

Kernel Object Interpretation

Profile
PoKeR: Architecture

Right-Before Detection → Logging and Context Tracking
“Right before” detection?

Applications

Guest OS

VMM

NICKLE Module

Standard

Shadow
“Right before” detection?

![Diagram showing applications, guest OS, memory access, and NICKLE module with standard and shadow options.]

- VM
- Applications
- Guest OS
- Memory Access
- NICKLE Module

Options:
- Standard
- Shadow
“Right before” detection?

Applications
Guest OS
Memory Access
NICKLE Module
VMM

VM

Guest Kernel Instruction Fetch

Standard
Shadow
“Right before” detection?

![Diagram showing VM, Guest OS, Applications, Memory Access, VMM, NICKLE Module, Standard, and Shadow.]
“Right before” detection?

![Diagram showing the interaction between applications, guest OS, VMM, and memory access modules.]

- Applications
- Guest OS
- VMM
- Memory Access
- NICKLE Module
- Standard
- Shadow
“Right before” detection?
“Right before” detection?

Applications

Guest OS

VM

NICKLE Module

VMM

Memory Access

Other Memory Access

Guest Kernel Instruction Fetch

Standard

Shadow
“Right before” detection?
“Right before” detection?

Applications

Guest OS

VM

Memory Access

VMM

NICKLE Module

Memory Access

Standard

Compare

Shadow
What code does it run?

- Compare standard and shadow memories
  - Extract code as you go
PoKeR: Architecture

Virtual Machine

Kernel Symbols & Kernel Object Types

Logging and Context Tracking

Virtual Machine Monitor

Right-Before Detection

Log

Profile

User-level Applications

Guest Kernel

Virtual Machine Monitor

Kernel Object Interpretation
Kernel Symbols & Kernel Object Types

Logging and Context Tracking

Log

Kernel Object Interpretation
Logging and context tracking

• Logging rootkit code...
  – Execution
  – Reads
  – Writes
What kernel objects does it modify?

• We have memory *writes* from rootkit code
• Use static analysis to build a map
  – Kernel with debug symbols
What about dynamic allocation?

- Some objects are allocated dynamically
What about dynamic allocation?

• Some objects are allocated dynamically

Static Objects

init_task
0xc0300000

pid
0

next_task
0xc11a0000

…

Dynamic Objects

task_struct
0xc11a0000

pid
1

next_task
0xc11b0000

…

task_struct
0xc11b0000

pid
2

next_task
0xc11c0000

…
Simple observation #1

<table>
<thead>
<tr>
<th>Static Objects</th>
<th>Dynamic Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Static Objects" /></td>
<td><img src="image2" alt="Dynamic Objects" /></td>
</tr>
</tbody>
</table>
Simple observation #1

Static Objects

Dynamic Objects
Simple observation #2

- The rootkit is just as ignorant as we are
- It will find dynamic objects by starting at static ones
“Combat tracking”

• Track rootkit reads
• Build a map of dynamic memory
• Reverse VMI
Combat tracking example

Static Objects

init_task
0xc0300000

pid
0

next_task
0xc11a0000

...

Dynamic Objects

task_struct
0xc11a0000

pid
1

next_task
0xc11b0000

...

task_struct
0xc11b0000

pid
2

next_task
0xc11c0000

...

Memory Map

0xc0300000 – task_struct

Output
Combat tracking example

Static Objects

- init_task
  - pid: 0
  - next_task: 0xc11a0000
- ...

Dynamic Objects

- task_struct
  - pid: 1
  - next_task: 0xc11b0000
- ...

- task_struct
  - pid: 2
  - next_task: 0xc11c0000
- ...

Memory Map

- 0xc0300000 – task_struct

Output
Combat tracking example

**Static Objects**

- **init_task**
  -pid 0
  -next_task 0xc11a0000
  -...

**Dynamic Objects**

- **task_struct**
  -pid 1
  -next_task 0xc11b0000
  -output

- **task_struct**
  -pid 2
  -next_task 0xc11c0000
  -output

**Memory Map**

- 0xc0300000 – task_struct

**Output**

-
Combat tracking example

Static Objects
- init_task 0xc0300000
  - pid 0
  - next_task 0xc11a0000
    - ...

Dynamic Objects
- task_struct 0xc11a0000
  - pid 1
  - next_task 0xc11b0000
    - ...
- task_struct 0xc11b0000
  - pid 2
  - next_task 0xc11c0000
    - ...

Memory Map
- 0xc0300000 – task_struct

Output
Combat tracking example

**Static Objects**

- `init_task 0xc0300000`
  - `pid 0`
  - `next_task 0xc11a0000`
  - ...

**Dynamic Objects**

- `task_struct 0xc11a0000`
  - `pid 1`
  - `next_task 0xc11b0000`
  - ...

- `task_struct 0xc11b0000`
  - `pid 2`
  - `next_task 0xc11c0000`
  - ...

**Memory Map**

- `0xc0300000 – task_struct`
- `0xc11a0000 – task_struct`

**Output**
Combat tracking example

Static Objects

init_task 0xc0300000
  pid 0
  next_task 0xc11a0000
  ...

Dynamic Objects

task_struct 0xc11a0000
  pid 1
  next_task 0xc11b0000
  ...
task_struct 0xc11b0000
  pid 2
  next_task 0xc11c0000
  ...

Memory Map

0xc0300000 – task_struct
0xc11a0000 – task_struct

Output
Combat tracking example

**Static Objects**

- init_task
  - pid 0
  - next_task 0xc11a0000
  - ...

**Dynamic Objects**

- task_struct
  - pid 1
    - next_task 0xc11b0000
    - ...
  - pid 2
    - next_task 0xc11c0000
    - ...

**Memory Map**

- 0xc0300000 – task_struct
- 0xc11a0000 – task_struct

**Output**
Combat tracking example

**Static Objects**

- `init_task`
  - `0xc0300000`
  - `pid`
    - `0`
  - `next_task`
    - `0xc11a0000`
    - `...`

**Dynamic Objects**

- `task_struct`
  - `0xc11a0000`
  - `pid`
    - `1`
  - `next_task`
    - `0xc11b0000`
    - `...`

- `task_struct`
  - `0xc11b0000`
  - `pid`
    - `2`
  - `next_task`
    - `0xc11c0000`
    - `...`

**Memory Map**

- `0xc0300000` – `task_struct`
- `0xc11a0000` – `task_struct`

**Output**
Combat tracking example

**Static Objects**

- `init_task` 0xc0300000
  - `pid` 0
  - `next_task` 0xc11a0000
    - ...

**Dynamic Objects**

- `task_struct` 0xc11a0000
  - `pid` 1
    - `next_task` 0xc11b0000
      - ...

- `task_struct` 0xc11b0000
  - `pid` 2
    - `next_task` 0xc11c0000
      - ...

**Memory Map**

- 0xc0300000 – `task_struct`
- 0xc11a0000 – `task_struct`

**Output**
Combat tracking example

Static Objects

init_task 0xc0300000
  pid 0
  next_task 0xc11a0000 ...

Dynamic Objects

task_struct 0xc11a0000
  pid 1
  next_task 0xc11b0000 ...

task_struct 0xc11b0000
  pid 2
  next_task 0xc11c0000 ...

Memory Map

0xc0300000 – task_struct
0xc11a0000 – task_struct
0xc11b0000 – task_struct

Output
Combat tracking example

**Static Objects**

- init_task: 0xc0300000
  - pid: 0
  - next_task: 0xc11a0000

**Dynamic Objects**

- task_struct: 0xc11a0000
  - pid: 1
    - next_task: 0xc11b0000

- task_struct: 0xc11b0000
  - pid: 2
    - next_task: 0xc11c0000

**Memory Map**

- 0xc0300000 – task_struct
- 0xc11a0000 – task_struct
- 0xc11b0000 – task_struct

**Output**

- next_task: 0xc11a0000
- pid: 0
- init_task: 0xc0300000

- next_task: 0xc11b0000
- pid: 1
- task_struct: 0xc11a0000

- next_task: 0xc11c0000
- pid: 2
- task_struct: 0xc11b0000
**Combat tracking example**

**Static Objects**

- `init_task` 0xc0300000
  - pid 0
  - next_task 0xc11a0000
    - ...

**Dynamic Objects**

- `task_struct` 0xc11a0000
  - pid 1
  - next_task 0xc11b0000
    - ...

- `task_struct` 0xc11b0000
  - pid 2
  - next_task 0xc11c0000
    - ...

**Memory Map**

- 0xc0300000 – task_struct
- 0xc11a0000 – task_struct
- 0xc11b0000 – task_struct

**Output**
Combat tracking example

**Static Objects**

- init_task
  - pid: 0
  - next_task: 0xc11a0000

**Dynamic Objects**

- task_struct
  - pid: 1
    - next_task: 0xc11b0000
  - task_struct
    - pid: 2
      - next_task: 0xc11c0000

**Memory Map**

- 0xc0300000 – task_struct
- 0xc11a0000 – task_struct
- 0xc11b0000 – task_struct

**Output**

...
Combat tracking example

**Static Objects**

- **init_task**: 0xc0300000
  - **pid**: 0
  - **next_task**: 0xc11a0000
    - ...

**Dynamic Objects**

- **task_struct**: 0xc11a0000
  - **pid**: 1
    - **next_task**: 0xc11b0000
      - **pid**: 2
        - **next_task**: 0xc11c0000
          - *

**Memory Map**

- 0xc0300000 – task_struct
- 0xc11a0000 – task_struct
- 0xc11b0000 – task_struct

**Output**
Combat tracking example

Static Objects

init_task 0xc0300000

pid 0

next_task 0xc11a0000 ...

Dynamic Objects

task_struct 0xc11a0000

pid 1

next_task 0xc11b0000 ...

task_struct 0xc11b0000

pid 2

next_task 0xc11c0000 ...

Memory Map

0xc0300000 – task_struct
0xc11a0000 – task_struct
0xc11b0000 – task_struct

Output

Write to 0xc11b0056
Combat tracking example

**Static Objects**

- **init_task**
  - 0xc0300000
- **pid**
  - 0
- **next_task**
  - 0xc11a0000
  - ...

**Dynamic Objects**

- **task_struct**
  - 0xc11a0000
    - **pid**
      - 1
    - **next_task**
      - 0xc11b0000
      - ...
- **task_struct**
  - 0xc11b0000
    - **pid**
      - 2
    - **next_task**
      - 0xc11c0000
      - ...

**Memory Map**

- 0xc0300000 – task_struct
- 0xc11a0000 – task_struct
- 0xc11b0000 – task_struct

**Output**

Write to 0xc11b0056

```
task_struct->euid
```
How does it modify control flow?

• Kernel hooks
  – Function pointers
  – Part of existing data objects
  – Could be statically or dynamically allocated
  – This is a subset of the previous point…
## Results – adore

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Kernel Objects Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>adore</td>
<td>0.42</td>
<td>sys_call_table[2,4,5,6,18,37,39,84,106]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sys_call_table[107,120,141,195,196,220]</td>
</tr>
<tr>
<td>adore</td>
<td>0.53</td>
<td>sys_call_table[1,2,6,26,37,39,120,141,220]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>proc_net-&gt;subdir-&gt;next-&gt;(...)-&gt;next-&gt;get_info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>proc_root_inode_operations-&gt;lookup</td>
</tr>
<tr>
<td>adore-ng</td>
<td>0.56</td>
<td>proc_net-&gt;subdir-&gt;next-&gt;(...)-&gt;next-&gt;get_info</td>
</tr>
<tr>
<td></td>
<td></td>
<td>proc_root_inode_operations-&gt;lookup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>proc_root_operations-&gt;readdir</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ext3_dir_operations-&gt;readdir</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ext3_file_operations-&gt;write</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unix_dgram_ops-&gt;recvmsg</td>
</tr>
</tbody>
</table>
## Results – hp rootkit

<table>
<thead>
<tr>
<th>Action</th>
<th>Value</th>
<th>Kernel Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0xc677c000</td>
<td>pidhash[600]</td>
</tr>
<tr>
<td>R</td>
<td>0x0000025a</td>
<td>pidhash[600]-&gt;pid</td>
</tr>
<tr>
<td>R</td>
<td>0xc76d8000</td>
<td>pidhash[600]-&gt;next_task</td>
</tr>
<tr>
<td>R</td>
<td>0xc6780000</td>
<td>pidhash[600]-&gt;prev_task</td>
</tr>
<tr>
<td>W</td>
<td>0xc6780000</td>
<td>pidhash[600]-&gt;next_task-&gt;prev_task</td>
</tr>
<tr>
<td>W</td>
<td>0xc76d8000</td>
<td>pidhash[600]-&gt;prev_task-&gt;next_task</td>
</tr>
</tbody>
</table>
Limitations

• Lack of formal completeness
• Cannot reveal the reason for modifications
• Combat tracking evasion
• Assume VMM isolation
• Kernel rootkits only
Related work

- Panorama
  - CCS ‘07
- HookFinder
  - NDSS ‘08
- HookMap
  - RAID ‘08
- K-Tracer
  - NDSS ‘09
Your three take aways…

• PoKeR: Virtualization based rootkit profiler

• Combat Tracking allows us to track dynamic data objects

• Tells what a rootkit does in order to help an expert determine why it does it