Automatic Inference and Enforcement of Kernel Data Structure Invariants

Arati Baliga, Vinod Ganapathy and Liviu Iftode
Department of Computer Science
Rutgers University
Rootkits, the growing threat!

- Computer systems today face a realistic and growing threat from rootkits.
  - 600% increase from 2004-2006 (McAfee Avert Labs)
  - Over 200 rootkits in first quarter of 2008 (antirootkit.com)

- Collection of tools used by the attacker to conceal his presence on the compromised system.

- Rootkits allow the attacker to...
  - Maintain long term control
  - Reuse the system’s resources
  - Spy on the system
  - Involve system in malicious activities
Rootkit hiding trends

**KERNEL SPACE**

- **CONTROL DATA**
  - Virtual File System (VFS) Handlers
  - System call table

- **NON-CONTROL DATA**
  - Process Lists

**USER SPACE**

- **Shared Libraries**
  - /usr/bin/ls
  - /usr/bin/ps
  - /usr/bin/netstat
  - /usr/lib/libc.so

- **User binaries**
  - /usr/bin/login

- **Backdoors, Key Loggers, Log erasers, etc**

**Below the operating system**

- Cloaker

**Hypervisor based rootkits (Subvirt, Blue pill)**
Current Approaches

- **Automated technique, limited in scope**
  - SBCFI [Petroni et al., CCS 2007]

- **Manual specification based techniques**
  - Copilot [Petroni et al., Usenix Security 2004]
  - Specification based architecture [Petroni et al., Usenix Security 2006]

- **Challenge**

<table>
<thead>
<tr>
<th></th>
<th>Location of data</th>
<th>Type of data</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Static</td>
<td>Dynamic</td>
<td>Control</td>
</tr>
<tr>
<td>Copilot</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>Specification based detection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SBCFI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Our approach</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Approach
- Attack examples
- Design and implementation
- Experimental evaluation
- Conclusions
Our approach

- A comprehensive technique to detect rootkits based on automatic invariant inference.

- **Invariant is a property** that holds over an individual object (e.g. variable or struct) or a collection of objects (e.g. arrays or linked lists).

- Learns invariants over a training phase and enforces them during normal operation.

- Works uniformly across control as well as non-control data.
Attacks that violate invariants

- We demonstrate four examples in this talk
  - Two proposed by us [Baliga et al., Oakland 2007]
    - Entropy pool contamination
    - Resource Wastage
  - Two attacks proposed by others
    - Hiding Process (Used by the \textit{fu} rootkit, Butler et al.)
    - Adding binary format (Proposed by Shellcode security research group)
Attack 1 – Entropy pool contamination

**Attack Overview:**
Attack constantly writes zeroes into all three pools and the polynomials used to stir the pools

**Impact:**
All applications that rely on the random number generator such as tcp sequence numbers, session ids are affected
Attack 1 – Invariants violated

Data structures involved.

- `struct poolinfo`. This is a member of the entropy pool data structures of type `struct entropy_store`.

Invariant violated by attack.

- `poolinfo.tap1 ∈ \{26, 103\}`
- `poolinfo.tap2 ∈ \{20, 76\}`
- `poolinfo.tap3 ∈ \{14, 51\}`
- `poolinfo.tap4 ∈ \{7, 25\}`
- `poolinfo.tap5 == 1`
Attack 2 – Resource wastage attack

**Attack Overview:**
Attack manipulates the zone watermarks to create an impression that most of the memory is full.

**Impact:**
Resource wastage and performance degradation

<table>
<thead>
<tr>
<th>Watermark</th>
<th>Original Value</th>
<th>Modified Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pages_min</td>
<td>255</td>
<td>210000</td>
</tr>
<tr>
<td>pages_low</td>
<td>510</td>
<td>215000</td>
</tr>
<tr>
<td>pages_high</td>
<td>765</td>
<td>220000</td>
</tr>
<tr>
<td>total free pages</td>
<td>144681</td>
<td>210065</td>
</tr>
</tbody>
</table>

Total number of pages in zone: 225280
Attack 2 – Invariants violated

Data structures involved.
zone_table[] array. Each element of type struct zone_struct

Invariant violated by attack.
zone_table[1].pages_min == 255
zone_table[1].pages_low == 510
zone_table[1].pages_high == 765

Invariant type
• CONSTANCY invariants over individual OBJECTS

<table>
<thead>
<tr>
<th>Watermark</th>
<th>Original Value</th>
<th>Modified Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pages_min</td>
<td>255</td>
<td>210000</td>
</tr>
<tr>
<td>pages_low</td>
<td>510</td>
<td>215000</td>
</tr>
<tr>
<td>pages_high</td>
<td>765</td>
<td>220000</td>
</tr>
<tr>
<td>total free pages</td>
<td>144681</td>
<td>210065</td>
</tr>
</tbody>
</table>

Total number of pages in zone: 225280
Attack 3 - Hidden process attack

**Data structures involved.**
Process run-list
Process all-tasks list

**Invariant:**
run-list ⊆ all-tasks

**Invariant type**
- SUBSET property over a COLLECTION (LINKED LIST)

**Attack Overview:**
Attack removes malicious process entry from all-tasks list but retains in run-list

**Impact:**
Malicious process is hidden from accounting tools
**Attack Overview:**
Attack adds a new binary format containing a malicious handler.

**Impact:**
Malicious code invoked each time a new process is created on the system.

**Data structures involved:**
- formats list

**Invariant:**
\[ \text{len(formats)} = 2 \]
Gibraltar architecture

Page Fetcher

Data Structure Extractor

Root Symbols
Kernel Data Definitions

Invariant Templates

Invariant Generator

Training Enforcement

Monitor

Physical Memory Address

010101
010101
010000
110011

run-list ⊆ all-tasks

run-list ⊆ all-tasks?
Prototype (Gibraltar)

- Fetches remote memory pages from the target continuously
Invariants automatically inferred

<table>
<thead>
<tr>
<th>Template</th>
<th>Object</th>
<th>Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Membership</td>
<td>643,622</td>
<td>422</td>
</tr>
<tr>
<td>Non-zero</td>
<td>49,058</td>
<td>266</td>
</tr>
<tr>
<td>Bounds</td>
<td>16,696</td>
<td>600</td>
</tr>
<tr>
<td>Length</td>
<td>NA</td>
<td>4,696</td>
</tr>
<tr>
<td>Subset</td>
<td>NA</td>
<td>3,580</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>709,376</td>
<td>9,564</td>
</tr>
</tbody>
</table>

Total 718,940 invariants inferred by Gibraltar. These invariants are used as data structure integrity specifications during enforcement.
Detection Accuracy

- **Test suite**
  - Fourteen publicly available kernel rootkits
  - Six advanced stealth attacks on the kernel (previously discussed)

- **Results**
  - All of them detected (No false negatives)

- **False positive evaluation**
  - Benign workload run for half an hour consisting of combination of tasks
  - 0.65% false positive rate

<table>
<thead>
<tr>
<th>Template</th>
<th>Object</th>
<th>Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Membership</td>
<td>0.71%</td>
<td>1.18%</td>
</tr>
<tr>
<td>Non-zero</td>
<td>0.17%</td>
<td>2.25%</td>
</tr>
<tr>
<td>Bounds</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Length</td>
<td>NA</td>
<td>0.66%</td>
</tr>
<tr>
<td>Subset</td>
<td>NA</td>
<td>0%</td>
</tr>
</tbody>
</table>

Average false positive rate: 0.65%

- # Copying the Linux kernel source code from one folder to another.
- # Editing a text document
- # Compiling the Linux kernel
- # Downloading eight video files from the Internet.
- # Perform file system operations using the IOZone benchmark
Performance Evaluation

- **Training Time**
  - 25 mins for snapshot collection, 31 minutes for invariant inference (Total of 56 minutes).

- **Detection Time**
  - Ranges from 15 seconds up to 132 seconds. Large variance depending on the number of objects found in memory.
  - Number of objects varies depending on the workload running on the system and system uptime.

- **PCI Overhead**
  - DMA access creates contention for the memory bus.
  - 0.49% (Results of the stream benchmark)
Conclusions and future work

- Our approach automatically infers invariants over kernel control and non-control data.

- Gibraltar could automatically detect publicly available rootkits and advanced stealth attacks using automatically inferred invariants.

- As future work, we plan to investigate
  - Improvement of false positive rate (filtering, feedback)
  - Quality of invariants generated
  - Portability of invariants across reboots.
Questions ?

Thank you !
Data structure extractor

BFS Queue

Static data

Root 1
Root 2
Root 3
... 
Root n

![Diagram showing BFS Queue and static data structure]

```
struct foo {
    struct bar * b1;
    struct list_head * p;
}
```

```
struct list_head {
    struct list_head * next;
    struct list_head * prev;
}
```

Linked list of objects of type “struct foo”
Invariant generator

- We leverage Daikon’s invariant inference engine to extract invariants over kernel snapshots.

- Daikon is a tool for dynamic invariant inference over application programs.

- We focus on the following five templates
  - Membership template (var ∈ {a, b, c}).
  - Non-zero template (var ≠ 0).
  - Bounds template (var < const), (var > const).
  - Length template (length(var) == const).
  - Subset template (list1 ⊆ list2).