1. Problem

Profiling a kernel rootkit’s behavior in a “live” system (e.g., honeypot) in multiple aspects: (1) hooking behavior, (2) manipulated kernel objects, (3) affected system calls, and (4) malicious kernel code executed.

2. PoKeR: A Profiler of Kernel Rootkits

- Real-time switch to kernel rootkit profiling mode
- Reverse VMI: given a memory location, infer the kernel object
- Association of rootkit code execution with system calls

3. Challenges and Solutions

- Adapting instantaneous rootkit detection system (e.g., NICKLE)
- Combat tracking: following the reads/writes of kernel rootkits
- System call introspection and process-based association

4. Results

- 10 real-world kernel rootkits profiled
- High accuracy compared with manual analysis
- Telling researcher what a rootkit did and helping her determine why

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Kernel Objects Modified</th>
<th>Syscalls Affected</th>
<th>Attack Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adore 0.42</td>
<td>770</td>
<td><code>sys_call_table[2, 4, 5, 6, 18, 37, 39, 84, 106] sys_call_table[107, 120, 141, 195, 196, 220]</code></td>
<td>2 - fork, 4 - write, 5 - open, 6 - close, 195 - stat64, 196 - stat64, 220 - getdents64</td>
<td>syscall hook</td>
</tr>
<tr>
<td>Adore-ng 0.53</td>
<td>733</td>
<td><code>sys_call_table[2, 4, 5, 6, 18, 37, 39, 120, 141, 220] proc_net=netdir-&gt;next-&gt;(...)-&gt;next-&gt;get_info proc_root_inode_operations-&gt;lookup</code></td>
<td>1 - exit, 2 - fork, 3 - read, 5 - open, 6 - close, 85 - readlink, 195 - stat64, 220 - getdents64</td>
<td>syscall hook, data hook</td>
</tr>
<tr>
<td>Adore-ng 0.56</td>
<td>785</td>
<td><code>proc_net=netdir-&gt;next-&gt;(...)-&gt;next-&gt;get_info proc_root_inode_operations-&gt;lookup proc_root_operations-&gt;readdir ext3_dir_operations-&gt;readdir ext3_file_operations-&gt;write unix_dgram_ops-&gt;recvmsg</code></td>
<td>3 - read, 5 - open, 85 - readlink, 195 - stat64, 220 - getdents64</td>
<td>data hook</td>
</tr>
</tbody>
</table>