UNOBSERVABLE INTRUSION DETECTION BASED ON CALL TRACES IN PARAVIRTUALIZED SYSTEMS

Marino Miculan
University of Udine

(Work in collaboration with Carlo Maiero)

Intrusion Detection Systems

- IDS gathers data from the management system (via "sensors") and using a KB decides if to raise an alert
- Crucial design questions:
  - What to observe?
  - How to observe?

What To Observe?

- "Syntactic" IDS look for discrepancies in code, data… (virus signatures, digests of programs, patterns…)
  - Quite limited
    - Patterns change often
    - (Antivirus detect ~50% viruses)
    - Difficult to look into process memory (e.g. to detect buffer overflows)

- "Semantic" IDS: look for discrepancies in the run-time behavior with respect to the expected one (the "model")
  - More robust to changes, non intrusive, …
  - Behavior = interactions with environment

  Slogan: A process behavior is fully determined by its system call traces (with parameters)
  - Black box approach: no need to look "inside" the application
First (naive) Architecture

- Process to monitor
- Model Learning
- dtrace int 0x80
  - Kernel space
  - User space
  - Syscall kernel code
  - Entry probe
  - Return probe

Anomaly Detection Engine
- Process to monitor
- dtrace int 0x80
  - Kernel space
  - User space
  - Syscall kernel code
  - Entry probe
  - Return probe

Model

Training phase (in safe context)

Monitoring phase (online, unsafe context)

But The Enemy Is Smart…

- First architecture requires changes in Operating System kernel in order to place probes on system calls
- Attacker can
  - notice the presence of probe, and change attack accordingly
  - attack the IDS itself, by removing probes
- How to observe system call traces WITHOUT changing OS?

Solution: Paravirtualized Systems

How To Intercept Syscall In VM

- In paravirtualized system system calls are trapped in a different way
- What and where to intercept?
New Architecture: XenIDS

Stealth Interception

Advantages Of XenINII/XenIDS Architecture

Model Construction And Anomaly Detection

- Secure: does not change any guest kernel structure, thus cannot be tampered
- Isolated and unobservable: the attacker cannot tell whether is monitored or not
- Flexible and independent from virtual machine
- Independent from memory: no introspection in guest memory or disk
- Simple: only one point of deployment
Algorithms For Anomaly Detection: Stide

- Stide looks for suspect subsequences of syscalls

- **Model**: All subsequences of length \( k \) of normal execution (patterns) of all programs running on a machine (usually \( k = 5 \) or 6)

- **Learning**: All pattern generated by a machine during normal execution are stored in database
  - This can lead to more false negatives in a server running many programs, but not more false positives. (Not observed in our tests)

   ![Dictionary of normal sequences](image)

   - P1
     - 5 2 1 2 2
   - P2
     - 3 5 5 5 3
   - P3
     - 4 3 2 4 3
   - P4
     - 3 4 3 1 6

Algorithms For Anomaly Detection: Stide (cont.)

- **Detection**: an intrusion is recognized only if the number of anomalies on the last \( n \) syscalls is > threshold.

   ![Detection Diagram](image)

About The Threshold

- Choosing the threshold \( Th \) is crucial
  - Low \( Th \) => too many false positives
  - High \( Th \) => attacks with less anomalies than \( Th \) are not detected (false negative)

  - For our test, after two weeks of training period we identified \( Th \) as 15%
  - No false positives
  - Behaviors differing less than 15% from stored sequences are considered “safe”

Stide: Evaluation Of Detection Capability

- **Offline test** on M.I.T. interception traces: all attacks have been recognized, no false positives
- **Online test**: observation of a modified (i.e. “hacked”) FTP server

<table>
<thead>
<tr>
<th>Change to FTP server</th>
<th>Mismatch</th>
<th>Anomaly?</th>
</tr>
</thead>
<tbody>
<tr>
<td>local copy string</td>
<td>20%</td>
<td>Yes</td>
</tr>
<tr>
<td>open a system shell</td>
<td>50%</td>
<td>Yes</td>
</tr>
<tr>
<td>remote copy string</td>
<td>30%</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use</th>
<th>Mismatch</th>
<th>Anomaly?</th>
</tr>
</thead>
<tbody>
<tr>
<td>strings of 25 chars</td>
<td>&lt; 15%</td>
<td>No</td>
</tr>
<tr>
<td>strings of 100 chars</td>
<td>&lt; 15%</td>
<td>No</td>
</tr>
<tr>
<td>closing using kill</td>
<td>&lt; 15%</td>
<td>No</td>
</tr>
</tbody>
</table>
Stide: Performance Evaluation

Overall overhead: 7–8% (in asynchronous mode)

Conclusions

• We have shown how to detect host intrusions by observing only system calls, without being observed by the intruder
• The overhead of XenIDS is acceptable for real time detection
• Threshold is delicate: it depends on various aspects
  • the training period
  • the desired “aggressiveness” of the IDS
• To circumvent these issues, we are working on new models based on Execution Graphs extended with Data Flow constraints

Thanks For Attention

Questions?

marino.miculan@uniud.it