A Tamper-resistant Framework for Unambiguous Detection of Attacks in User Space Using Process Monitors

R. Chinchani
S. Upadhyaya
Kevin Kwiat

Computer Science & Eng.
University at Buffalo

Buffalo, New York, 14260

March 24, 2003
Upadhyaya – IWIA 2003
Outline

- Basic Terminology
- What is Tamper-resistant Monitoring?
- Analysis
- Simple Problem Transformation
- A New Framework
- Experimental Results
- Conclusions and Future Work
Basic Terminologies

- **A Host**
  - A computer system

- **A Service**
  - A user space program that performs some useful task

- **A Vulnerability**
  - A flaw in a program with security implications
  - An Attacker targets vulnerabilities in programs to gain elevated privileges
  - An *intrusion detection system* (IDS) attempts to detect and prevent attacks
Tamper-resistant Monitoring

- Protecting user space components
  - To have secure enclaves, every component of IDS must be watched and there are no open ends
- Attacker’s perspective
The Attack Model

- An attacker is successful, if:
- He compromises the service running on the host
- He disables or compromises the IDS, if one is deployed
- **Note:**
  - As coverage of IDS improves, the attacker’s focus will be on the IDS itself
  - Who watches the watcher?
Criteria for Analysis

- Failure due to faults and failure due to attacks are not one and the same.
- Each software component can potentially fail due to attacks; we just don't know how yet!
- Security of a system is only as strong as the weakest link.
- Hence, even if a service is monitored by a separate detection mechanism, is the entire setup really secure?
- An IDS is a program and it too can be compromised.
- So, implement it inside the kernel for tamper-resistance.
- But, poor kernel implementations affect the whole system.
Probabilistic Analysis

- An attacker has little knowledge about how the entire system works and the attacks are by trial and error

- Factors that determine the overall security are:
  - For each component, individual probability of failure
  - For a $n$ component system - the probability of total failure
  - An attack can proceed in stages - probability of per-stage failure
  - Search space for a successful attack
  - False sense of security, since an attacker will eventually learn of these weaknesses
Deterministic Analysis

- An attacker has complete knowledge of the system and its vulnerabilities
- Factors that determine the overall security are:
  - A successful attack strategy
- The only line of defense is the structure of the security mechanism
Theoretical Model

- Each process is a node
- If a process monitors another process, then there is a directed edge between the corresponding nodes
- Useful in representing and analyzing the structure of interaction
- Reduces to a graph problem
- No mutual trust among processes
If a detection mechanism is deployed, then how much overhead will it incur?

- $\delta$ - overhead due to isolated execution
- $\theta$ - overhead due to monitoring
Simple Replication

- Chameleon project at UIUC
- Can be easily compromised
Layered Hierarchy

- An example is the AAFID project at Purdue
- Provides only a marginal increase in security
New Proposed Architecture

- Circulant Digraph

- Very difficult to subvert
- Provides an infinite hierarchy of monitoring using finite number of monitors
## Comparison

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Simple Replication</th>
<th>Layered Hierarchy</th>
<th>Circulant Digraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subversion by sequential attack?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Total Probability of Subversion</td>
<td>$p^n$</td>
<td>$p^n$</td>
<td>$p^n$</td>
</tr>
<tr>
<td>Per-stage Probability of Subversion</td>
<td>$p$</td>
<td>$p$</td>
<td>$p^n$</td>
</tr>
<tr>
<td>Degree of Incidence</td>
<td>0</td>
<td>1</td>
<td>$d$</td>
</tr>
<tr>
<td>Overhead due to isolated execution</td>
<td>$n.\delta$</td>
<td>$n.\delta$</td>
<td>$n.\delta$</td>
</tr>
<tr>
<td>Overhead due to monitoring</td>
<td>$n.\theta$</td>
<td>$n.\theta$</td>
<td>$n.d.\theta$</td>
</tr>
<tr>
<td>Total overhead</td>
<td>$n.\delta + n.\theta$</td>
<td>$n.\delta + n.\theta$</td>
<td>$n.\delta + n.d.\theta$</td>
</tr>
</tbody>
</table>
Implementation

- Requires a sense-decide-act loop
- Uses FreeBSD's *kqueue* subsystem
- Each node in the graph is a process monitor
- Multiple outgoing edges from a node are threads
kqueue subsystem

- kqueue is a replacement for select() and poll() in FreeBSD 4.5 and above
- Guaranteed event delivery
- Non-repudiation of events

(1) Register listener

(2) System call

(3) Report event

KQUEUE
Attack Scenarios

- Kqueue currently supports only exit(), fork() and exec() family of system calls.
- Currently, our experiments are limited to crash attacks.
- All processes can be crashed using the kill() system call.
- An attacker is given all information about vulnerabilities.
- Are TOCTOU style attacks possible?
  - Experiments performed under heavy system load.
Results
Interpretation

- Even with all the information, it is extremely difficult to launch a successful attack
- As degree of incidence increases, more attack alerts are raised
- So, it is enough to deploy a small number of process monitors but with a high degree
Conclusion

- Mechanisms based on no-knowledge of vulnerabilities gives a weaker sense of security
- We have presented a methodology to analyze security in terms of structure
- We show that a circulant digraph configuration is a better solution
Future Work

- The short term goal is to supplement the kqueue subsystem to support more events
  - To increase coverage
  - To detect other attacks
- The long term goal is to devise a way to provide a generic tamper-resistant wrapper around user space services
  - Extend it to network level monitoring
- Our website:
  - http://www.cse.buffalo.edu/caeiae/