Intrusion Detection and Forensics Using Series of Function Calls

Sean Peisert
UCSD

February 21, 2006
Talk for UCI Graduate “Language-Based Security” Class
How Do We Do “Good” Forensics & ID?

- Forensic tools help to analyze the data
- Data helps understand what went wrong
- Data is descriptive
- Real-time ID tools help to find the anomaly
- Data is useful for automated “search”
- Why can’t we have both?
Possible System Abstractions?

- Assembly Code
- Syslog Messages
- Login/Logouts
- Resource Usage Metrics
- System Calls
Forensic Principles

- Consider the entire system
- Consider actions and their effects.
- Runtime data is the only authoritative record of what happened.
- Actions and results must be processed and presented in a way that is understandable by humans.
Intrusion Detection

- Anomaly Detection
- Anomaly Detection w/Automated Rule Generation
- Signature (Misuse) Detection
Anomaly Detection

- D. Denning in 1986
- Immunological Approach (Forrest, et al.)
- Data Mining
Forensics

- syslog
- “Analyzing Computer Intrusions” (Andrew Gross)
- BackTracker (King & Chen)
Research Questions

Can we improve post-hoc anomaly detection accuracy by using function calls as data, as opposed to system calls alone?

Can we enable forensic analysis of “intrusions” not otherwise possible or easy?
Methods

- Capture all calls, their arguments, and their return values
- Compare series of calls between “safe” set and “test” set
- Future: Compare arguments and return values between “safe” set and “test” set
Hamming Distance

Example:

- “Safe”: a b c d e f
- “Test”: a b c d e g

Hamming Distance (d) = ?

Min sequence length to find anomaly = ?
Minimum Hamming Distance

“Safe” Corpus:
- Size 2: e f, f c, f a, f b,
- Size 3: e f a, e f b, f f c

“Test” Sequence: e f c

What is the minimum sequence length required to detect this as an anomaly?
Immunological Approach

- Sliding window of size $k$
- "Safe" sequences $j$
- "Test" sequences $i$

$d_{\text{min}} = \min\{d(i,j) \text{ for all safe sequences } j\}$

$\hat{S}_A = \max\{d_{\text{min}}(i) \text{ for all new sequences } i\}/k$
Analyzing Function Arguments & Return Values

- Can’t use the same techniques — need more advanced data mining
- Clustering: k-nearest-neighbor, k-means
Forensic Methods

Prefer to have source code to search for captured calls
Gathering Data

Variety of methods:

- Virtual Machine (a la “Introvirt”)
- Binary Rewriter/Dynamic Instrumentation
- Compiler
- Intel’s Pin (Luk & Cohn, et al., PLDI 2005)
## su Experiment #1

- Removed call to `pam_authenticate()`. What changed?

<table>
<thead>
<tr>
<th>k=4</th>
<th>total calls</th>
<th>total seq</th>
<th>unique seq</th>
</tr>
</thead>
<tbody>
<tr>
<td>su-orig</td>
<td>88208</td>
<td>51085</td>
<td>2170</td>
</tr>
<tr>
<td>su-mod</td>
<td>49453</td>
<td>30669</td>
<td>1891</td>
</tr>
<tr>
<td></td>
<td>total different seqs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>k=4</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>only in su-orig</td>
<td>18453 (315 unique)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>only in su-mod</td>
<td>36 (all 36 unique)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**su-orig vs. su-mod**
Total Function Call Seqs in su

- su-orig
- su-mod

Number of sequences vs. sequence length graph.
Difference in Total Function Call Seqs in su

- in su-orig, not su-mod
<table>
<thead>
<tr>
<th>sequence</th>
<th># total occurrences</th>
<th>% of total program</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5Update</td>
<td>5538</td>
<td>10.85%</td>
</tr>
<tr>
<td>MD5Final</td>
<td>2005</td>
<td>3.92%</td>
</tr>
<tr>
<td>MD5Init</td>
<td>1002</td>
<td>1.96%</td>
</tr>
<tr>
<td>MD5Pad</td>
<td>1002</td>
<td>1.96%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9547</strong></td>
<td><strong>18.69%</strong></td>
</tr>
</tbody>
</table>
su Experiment #2

- Ignored result of `pam_authenticate()` call

<table>
<thead>
<tr>
<th>k=2</th>
<th>total different seqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>only in su-orig</td>
<td>2594</td>
</tr>
<tr>
<td>(2379 unique)</td>
<td></td>
</tr>
<tr>
<td>only in su-mod</td>
<td>2</td>
</tr>
<tr>
<td>(both unique)</td>
<td></td>
</tr>
</tbody>
</table>

One of 2 seqs: `strcmp`, `pam_authenticate`
ssh Experiment #1

- Edited ssh to echo the password to the terminal

<table>
<thead>
<tr>
<th></th>
<th>k=4</th>
<th>total different seqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>only in ssh-orig</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>only in ssh-mod</td>
<td></td>
<td>47</td>
</tr>
</tbody>
</table>

vfprintf, vfprintf, fprintf, read_passphrase
ssh Experiment #2

Edited ssh to send the password through a network socket

<table>
<thead>
<tr>
<th>k=4</th>
<th>total different seqs</th>
</tr>
</thead>
<tbody>
<tr>
<td>only in ssh-orig</td>
<td>14 (14 unique)</td>
</tr>
<tr>
<td>only in ssh-mod</td>
<td>45 (45 unique)</td>
</tr>
</tbody>
</table>

inet_aton, inet_addr, rtld_free_tls, rtlf_free_tls
lpr Experiment

- Recreated UNM experiment that exploits lpr bug.
- Exploits counter, “creat” syscall, and symlink to rewrite /etc/passwd.
## lpr Results

<table>
<thead>
<tr>
<th>only in lpr-orig</th>
<th>only in lpr-mod</th>
</tr>
</thead>
<tbody>
<tr>
<td>seteuid, error_unthreaded</td>
<td>sys_write, close</td>
</tr>
<tr>
<td>sbrk, sys_umask</td>
<td>lseek, sys_write</td>
</tr>
<tr>
<td>open, sys_umask</td>
<td>copy, close</td>
</tr>
<tr>
<td></td>
<td>nfile, sys_read</td>
</tr>
<tr>
<td></td>
<td>creat, sys_umask</td>
</tr>
<tr>
<td></td>
<td>sys_read, sys_write</td>
</tr>
<tr>
<td></td>
<td>sys_read, sys_syscall</td>
</tr>
<tr>
<td></td>
<td>open, creat</td>
</tr>
<tr>
<td></td>
<td>sys_unlink, error_unthreaded</td>
</tr>
<tr>
<td></td>
<td>close, copy</td>
</tr>
<tr>
<td></td>
<td>close, close</td>
</tr>
<tr>
<td></td>
<td>close, seteuid</td>
</tr>
<tr>
<td></td>
<td>sys_umask, fchown</td>
</tr>
</tbody>
</table>
Conclusions

These initial experiments seem to help highlight anomalies and then help understand them.
(Immediate)

Future Work

- More experiments (including blind and/or double-blind ones)
- Arguments & return values
- Machine learning applied to function calls
- Tuning parameters
Key References

Acknowledgements

This work was partially funded by a Lockheed-Martin Information Assurance Technology Focus Group 2005 University Grant, and NSF.