Improving Host Based Anomaly Detection

Shariyar, Afroza, Prasanna and Abdelwahab

Software Behaviour Analysis Research Lab
abdelw@ece.concordia.ca

Montreal, QC
Dec. 9th, 2011
Reporting on Four Studies that We Conducted: Outline

• Our objective is to investigate advanced anomaly detection techniques
  – Study 1 - Comparing kernel space and user space tracing mechanisms for anomaly detection
  – Study 2 - Reducing false positive rate using generalization of system calls
  – Study 3 - Enhanced hidden Markov model using the concept of n-gram
  – Study 4 - Linux-based attack taxonomy
Comparison of user space and kernel space traces in discovering anomalous software behaviour

Duration: May to July and October, 2011
Why Identify Anomalous Behaviour?

• Identification of normal and anomalous software behaviour is important in:
  – Software debugging, such as fault localization (Murtaza et al. 2010, Jones et al. 2005)
  – Autonomic computing, such as self managing applications (Jiang et al. 2005)
  – Software intrusion, such as anomaly detection systems (Warrender et al. 1999, Wang et al. 2004)
Trace Examples

Function call trace (User space)

23  fooPrevious exit
24  foo1 entry
25
26
27
28  || foo3 exit
30  || foo2 exit
31  || foo1 exit
32  || fooLater entry

No comparison of user space and kernel space tracing exists in the literature: can we substitute one with another or which one is the best?

System events trace (Kernel space)

channel:kernel; event:syscall_entry process:/gzip.exe; state: SYSCALL; markers:ip = 0x22cbad, syscall_id = 6 [sys_close+0x0/0x100]; pid:2842

channel:fs; event:close process:/gzip.exe; state:SYSCALL;

channel:page fault; event:page_fault_entry process:/gzip.exe; state:TRAP; markers:ip = 0x8049aa9, address = 0x805d000, trap_id = 14, write_access = 1; pid:2842
Research Questions

• (Q1) Can kernel space tracing be used to classify pass fail traces of a program with the same accuracy as user space tracing?

• To find the answer we employed six classification algorithms (i.e., NB, C4.5, ANN, SVM, BBN, and HMM) and in the process identified a novel secondary research question.

• (Q2) Can we substitute one classification algorithm with another without affecting the accuracy of classification of normal and abnormal traces?
Approach

• Step 1: Collect user space and kernel space traces.
• Step 2: Extract events (e.g., function or system calls).
• Step 3: For each type of tracing evaluate all the classifiers from two perspectives:
  (a) Training and testing on both normal and anomalous traces.
  (b) Training on only normal traces and testing on both types of traces.

NB, C4.5, ANN, SVM, BBN, HMM
## Dataset

**Releases used:** Flex 2.5.1; Grep 2.4; Gzip 1.1.2; Sed 4.0.7.

<table>
<thead>
<tr>
<th>Prog.</th>
<th>LOC</th>
<th># Functions</th>
<th># Faults</th>
<th># Passing Traces</th>
<th># Failed Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flex</td>
<td>9724</td>
<td>167</td>
<td>20</td>
<td>566</td>
<td>545</td>
</tr>
<tr>
<td>Grep</td>
<td>9041</td>
<td>149</td>
<td>18</td>
<td>799</td>
<td>710</td>
</tr>
<tr>
<td>Gzip</td>
<td>4032</td>
<td>88</td>
<td>16</td>
<td>214</td>
<td>204</td>
</tr>
<tr>
<td>Sed</td>
<td>4735</td>
<td>115</td>
<td>6</td>
<td>366</td>
<td>166</td>
</tr>
</tbody>
</table>
### Results for two-class classification

#### Results on user space traces

<table>
<thead>
<tr>
<th>Algo.</th>
<th>TP</th>
<th>FP</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5</td>
<td>0.924</td>
<td>0.099</td>
<td>0.925</td>
</tr>
<tr>
<td>NB</td>
<td>0.159</td>
<td>0.053</td>
<td>0.609</td>
</tr>
<tr>
<td>BBN</td>
<td>0.371</td>
<td>0.145</td>
<td>0.675</td>
</tr>
<tr>
<td>ANN</td>
<td>0.981</td>
<td>0.804</td>
<td>0.646</td>
</tr>
<tr>
<td>SVM</td>
<td>0.721</td>
<td>0.323</td>
<td>0.699</td>
</tr>
<tr>
<td>HMM</td>
<td>0.706</td>
<td>0.0</td>
<td>0.416</td>
</tr>
</tbody>
</table>

#### Results on kernel space traces

<table>
<thead>
<tr>
<th>Algo.</th>
<th>TP</th>
<th>FP</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5</td>
<td>1.00</td>
<td>0.002</td>
<td>0.998</td>
</tr>
<tr>
<td>NB</td>
<td>1.00</td>
<td>0.002</td>
<td>0.999</td>
</tr>
<tr>
<td>BBN</td>
<td>0.993</td>
<td>0.004</td>
<td>0.999</td>
</tr>
<tr>
<td>ANN</td>
<td>0.998</td>
<td>0.002</td>
<td>1.00</td>
</tr>
<tr>
<td>SVM</td>
<td>0.998</td>
<td>0.002</td>
<td>0.998</td>
</tr>
<tr>
<td>HMM</td>
<td>1.00</td>
<td>0.002</td>
<td>0.996</td>
</tr>
</tbody>
</table>
(1) Kernel space tracing identifies software anomalies better than the function call traces at user space level
   – Time to train classifiers on kernel space traces was 20-60% less than user space traces

(2) No significant difference exist among classification algorithms in detection of software anomalies using execution traces
   – However, the C4.5 decision tree yields higher accuracy in two-class classification and neural network yields higher accuracy in one-class classification.
Reduction of false positive rate in anomaly detection through generalization of system calls.

False positives: A major problem in anomaly detection system

- A major problem is the generation of number of incorrect alarms on normal software behaviour—i.e., false positives.
- A large number of false positives in anomaly detection systems have made the misuse (signature based) detection systems first choice in the industry.

Is the problem in the application of algorithms on different datasets or is it in the properties of underlying data?
### Motivating Example

<table>
<thead>
<tr>
<th>Sequence 1</th>
<th>Sequence 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>fork</td>
<td>fork</td>
</tr>
<tr>
<td>read</td>
<td>read</td>
</tr>
<tr>
<td>fork</td>
<td>fork</td>
</tr>
<tr>
<td>read</td>
<td>read</td>
</tr>
<tr>
<td>fork</td>
<td>fork</td>
</tr>
<tr>
<td>read</td>
<td>read</td>
</tr>
</tbody>
</table>

Different contiguous repetitions of system calls but the task performed is exactly the same: **creation of a process** (fork) and **reading from an I/O device** (read).

There will be a mismatch (false positive) for “Sequence 2” if an algorithm is trained on “Sequence 1”, even though the task is the same.

These observations warrant an empirical investigation.
Research Hypothesis

On generalizing system calls, we can reduce false positive rate of an anomaly detection algorithm without affecting the true positive rate.
Approach

Training phase:
- System with normal behaviour
  - Collect traces
  - Remove Contiguous Repetition
  - Extract by sliding a window and store

Testing phase:
- System with unknown behaviour
  - Collect a trace
  - Remove Contiguous Repetition
  - Compare by sliding a window
  - Normal or abnormal trace

Sequences of length n

Foo3
Foo1
Foo3
Foo4
Foo2
Foo2
Foo2
## Datasets

<table>
<thead>
<tr>
<th>Program</th>
<th>Intrusion traces</th>
<th>Normal traces</th>
<th>Normal traces used for training</th>
<th>Normal traces used for testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sendmail</td>
<td>25</td>
<td>346</td>
<td>135</td>
<td>211</td>
</tr>
<tr>
<td>Stide</td>
<td>105</td>
<td>13726</td>
<td>600</td>
<td>13126</td>
</tr>
<tr>
<td>MIT live lpr</td>
<td>1001</td>
<td>2703</td>
<td>415</td>
<td>2288</td>
</tr>
<tr>
<td>UNM live lpr</td>
<td>1001</td>
<td>4298</td>
<td>390</td>
<td>3908</td>
</tr>
<tr>
<td>Xlock</td>
<td>2</td>
<td>1731</td>
<td>121</td>
<td>1610</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th></th>
<th>Sendmail</th>
<th>Stide</th>
<th>MIT live lpr</th>
<th>UNM live lpr</th>
<th>Xlock</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP</td>
<td>TP</td>
<td>FP</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td><strong>Win(w) = 6</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stide</td>
<td>24</td>
<td>16</td>
<td>69</td>
<td>104</td>
<td>196</td>
</tr>
<tr>
<td>CRA</td>
<td>23</td>
<td>16</td>
<td>66</td>
<td>104</td>
<td>181</td>
</tr>
<tr>
<td><strong>Win(w) = 10</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stide</td>
<td>27</td>
<td>16</td>
<td>12746</td>
<td>104</td>
<td>350</td>
</tr>
<tr>
<td>CRA</td>
<td>25</td>
<td>16</td>
<td>137</td>
<td>104</td>
<td>183</td>
</tr>
<tr>
<td><strong>Win(w) = 15</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stide</td>
<td>30</td>
<td>16</td>
<td>12760</td>
<td>104</td>
<td>458</td>
</tr>
<tr>
<td>CRA</td>
<td>27</td>
<td>16</td>
<td>187</td>
<td>105</td>
<td>183</td>
</tr>
<tr>
<td><strong>Win(w) = 20</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stide</td>
<td>33</td>
<td>18</td>
<td>12770</td>
<td>104</td>
<td>537</td>
</tr>
<tr>
<td>CRA</td>
<td>33</td>
<td>18</td>
<td>188</td>
<td>105</td>
<td>212</td>
</tr>
</tbody>
</table>

Significant difference exists in false positives, according to Wilcoxon signed rank test; but no significant increase in TP at higher win width.
Results

• At window width 6, the effect size is 0.5482 between our approach and Stide:
  – The results are interpreted as:
    • The average false positive rate of Stide will be 0.5482 standard deviations above than the average false positive rate of CRA.

• Thus, our hypothesis has been validated: False positives can be reduced significantly by removing contiguous repetitions of system calls.
Enhanced hidden Markov model using the concept of n-gram.

Duration: May to Oct., 2011.
Hidden Markov Model (HMM)

**Advantage:**
- Very Accurate

**Disadvantage:**
- Very Slow

\[ \text{The Time Complexity} = O(N(1+T(M+N))) \] [Langford 07]
- Number of Observables (M)
- Number of Hidden States (N)
- Length of Training Sequences (T)
Proposed Algorithm: I-HMM

Reduce the training time
- Reduce the number of observables
- Reduce the length of training sequences

Use frequent pattern (N-Gram) extracting technique
N-Gram Extraction: Example

Training Traces: ECDB, CDBA and EACDB

1-gram: A (2), B (3), C (3), D (3), E (2)

2-gram: EC (1), CD (3), DB (3), BA (1), EA (1), AC (1)

3-gram: CDB (3)

frequency(p_{k+1}) > \alpha \times \min(frequency(q_k), frequency(r_k))

[\alpha = 0.6]
Replacement of N-Grams: Example

Training Traces: ECDB, CDBA and EACDB

Assign unique ID: \( A(2) = 1, B(3) = 2, C(3) = 3, D(3) = 4, E(2) = 5, CD(3) = 6, \\
DB(3) = 7, CDB(3) = 8 \)

- ECDB, CDBA, EACDB
  - Replace CDB
  - E8, 8A, EA8
    - Replace A
    - E8, 81, E18
      - Replace E
      - 58, 51, 518
Model Construction

HMM
- Set of Observables = \{A, B, C, D, E\}
- Set of Training Sequences = \{ECDB, CDBA, EACDB\}
- Set of Hidden States = \{X_1, X_2, \ldots, X_m\}

I-HMM
- Set of Observables = \{1, 5, 8\}
- Set of Training Sequences = \{58, 51, 518\}
- Set of Hidden States = \{X_1, X_2, \ldots, X_m\}

Reduces the size of the set of the observables

Reduces the length of the training sequences.
Experiments and Results

Training Time for HMM and I-HMM Algorithms

<table>
<thead>
<tr>
<th>Numer of Traces</th>
<th>HMM</th>
<th>I-HMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>15.83</td>
<td>10.77</td>
</tr>
<tr>
<td>75</td>
<td>23.14</td>
<td>15.65</td>
</tr>
<tr>
<td>100</td>
<td>33.48</td>
<td>22.33</td>
</tr>
<tr>
<td>125</td>
<td>41.74</td>
<td>26.36</td>
</tr>
<tr>
<td>150</td>
<td>50.09</td>
<td>31.16</td>
</tr>
<tr>
<td>175</td>
<td>69.99</td>
<td>36.16</td>
</tr>
<tr>
<td>200</td>
<td>79.75</td>
<td>41.12</td>
</tr>
</tbody>
</table>
Experiments and Results

Training Time Reduction in I-HMM Algorithm

<table>
<thead>
<tr>
<th>Numer of Traces</th>
<th>Training Time Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>31.96</td>
</tr>
<tr>
<td>75</td>
<td>32.37</td>
</tr>
<tr>
<td>100</td>
<td>33.30</td>
</tr>
<tr>
<td>125</td>
<td>36.85</td>
</tr>
<tr>
<td>150</td>
<td>37.79</td>
</tr>
<tr>
<td>175</td>
<td>48.34</td>
</tr>
<tr>
<td>200</td>
<td>48.44</td>
</tr>
</tbody>
</table>
Experiments and Results

Accuracy for HMM and I-HMM Algorithms

Accuracy (%) | HMM | I-HMM
--- | --- | ---
60.00 | | |
65.00 | | |
70.00 | | |
75.00 | | |
80.00 | | |
85.00 | | |
90.00 | | |
95.00 | | |
100.00 | | |
105.00 | | |
110.00 | | |
115.00 | | |
120.00 | | |
125.00 | | |
130.00 | | |
135.00 | | |
140.00 | | |
145.00 | | |
150.00 | | |
155.00 | | |
160.00 | | |
165.00 | | |
170.00 | | |
175.00 | | |
180.00 | | |
185.00 | | |
190.00 | | |
195.00 | | |
200.00 | | |

Accuracy for HMM and I-HMM Algorithms

Accuracy (%) | HMM | I-HMM
--- | --- | ---
50.00 | 72.00 | 88.00
75.00 | 78.67 | 93.33
100.00 | 85.00 | 95.00
125.00 | 88.00 | 96.80
150.00 | 88.00 | 97.33
175.00 | 89.71 | 97.71
200.00 | 93.00 | 98.00
Linux Kernel-based attack taxonomy.
Duration: Sep. to Nov., 2011
Objective

• Build a taxonomy of known attacks and vulnerabilities for the Linux kernel
  – That can lead to techniques for mitigating these attacks

• There exist many attack taxonomies
  – They vary in coverage and target platforms
  – None focuses explicitly on the Linux kernel
  – Refer to: “AVOIDIT: A Cyber Attack Taxonomy” by C. Simmons et al. from the University of Memphis
## Proposed Attack Taxonomy Framework

<table>
<thead>
<tr>
<th><strong>Affected component</strong></th>
<th>The component of the Linux kernel that is vulnerable: Net, fs, etc..</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Effect of the attack</strong></td>
<td>What effect the attack has on the system: DoS, privilege escalation, information disclosure</td>
</tr>
<tr>
<td><strong>Origin of attack</strong></td>
<td>Locally exploitable, local area network exploitable, and remotely exploitable</td>
</tr>
<tr>
<td><strong>Complexity of access</strong></td>
<td>The need of privileges, special conditions, presence of other vulnerabilities…</td>
</tr>
<tr>
<td><strong>Impact</strong></td>
<td>Classified into confidentiality, Integrity, and Availability</td>
</tr>
</tbody>
</table>
Analysis of the 2011 Linux Kernel Vulnerabilities

• We studied 77 vulnerabilities in the Linux Kernel discovered and reported in the year 2011
  – Based on the vulnerabilities discovered and reported in 2011
  – We used www.cvedetails.com to filter Linux Kernel based vulnerabilities from the CVE database
Analysis of 2011 Kernel Vulnerabilities Based on Affected Component
Analysis of 2011 Kernel Vulnerabilities Based on the Effect of Attack

<table>
<thead>
<tr>
<th>Vulnerability Description</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain Sensitive Information</td>
<td>15</td>
</tr>
<tr>
<td>Denial of Service/Overflow</td>
<td>8</td>
</tr>
<tr>
<td>Denial of Service/Overflow/Memory Corruption</td>
<td>4</td>
</tr>
<tr>
<td>Denial of Service/Memory Corruption</td>
<td>2</td>
</tr>
<tr>
<td>Denial of Service/Overflow/Gain Privileges/Memory Corruption</td>
<td>2</td>
</tr>
<tr>
<td>Overflow/Gain Privileges/Obtain Sensitive Information</td>
<td>2</td>
</tr>
<tr>
<td>Unspecified</td>
<td>2</td>
</tr>
<tr>
<td>Denial of Service/Bypass</td>
<td>1</td>
</tr>
<tr>
<td>Denial of Service/Gain Privileges</td>
<td>1</td>
</tr>
<tr>
<td>Denial of Service/Gain Privileges/Memory Corruption</td>
<td>1</td>
</tr>
<tr>
<td>Denial of Service/Gain Privileges/Memory Corruption/Obtain Sensitive Information</td>
<td>1</td>
</tr>
<tr>
<td>Denial of Service/Obtain Sensitive Information</td>
<td>1</td>
</tr>
<tr>
<td>Denial of Service/Overflow/Gain Privileges</td>
<td>1</td>
</tr>
<tr>
<td>Denial of Service/Overflow/Obtain Sensitive Information</td>
<td>1</td>
</tr>
<tr>
<td>Denial of Service/ Overflow/ Gain Privileges</td>
<td>1</td>
</tr>
<tr>
<td>Overflow/Obtain Sensitive Information</td>
<td>1</td>
</tr>
</tbody>
</table>
Analysis of 2011 Kernel Vulnerabilities Based on the Origin of the Attack

- Local: 79%
- Local Network: 11%
- Remote: 10%
Analysis of 2011 Kernel Vulnerabilities Based on Access Complexity

- High: 9%
- Low: 56%
- Medium: 35%
Analysis of the 2011 Data Based on the Vulnerability Impact

- Availability
  - Partial: 1
  - Complete: 58
- Integrity
  - Partial: 19
  - Complete: 7
- Confidentiality
  - Partial: 7
  - Complete: 58
Conclusions

- Kernel space tracing is better than user space tracing in detecting normal and anomalous behaviour
- Classification algorithms when classifying normal and abnormal software behaviour yield similar results
- Generalization of system calls can reduce false positive rates significantly
- Using n-gram representation of function calls reduce the training time of HMM by 31.96% to 48.44%
- Most of the Linux vulnerabilities are exploited through host based attacks
Future Work

• Experiment with trace abstraction techniques to further reduce the trace size and training time
• Study other anomaly detection mechanisms based on continuous monitoring of system usage
• Incremental analysis of host-based systems to multiple system processes
• Investigate additional generalization methods in detection of anomalies
• Investigate feedback-directed and self-adaptive anomaly detection techniques
Thank you!
Dr. Wahab Hamou-Lhadj
Associate Professor

Mailing Address:
Department of ECE
Concordia University
1455 de Maisonneuve West
Montreal, Quebec H3G 1M8 Canada

Tel: +1 514 848 2424 x.7949
Fax: +1 514 848 2802
Email: abdelw@ece.concordia.ca

Civic Address:
Department of ECE
Concordia University
1515 St. Catherine, West
Montreal, Quebec H3G 2W1 Canada


References (3)


Gzip Official Website http://www.gzip.org/


LTTng Official Website http://lttng.org

Weka Official Website http://www.cs.waikato.ac.nz/ml/weka/

