Sequencegram: \(n\)-gram Modeling of System Calls for Program Based Anomaly Detection

Neminath Hubballi

Dept of Computer Science and Engineering, IIT Guwahati

January 6, 2011
Introduction

System call monitoring for Host based IDS

Model creation

Sequencegram

Results

Summary

References
Introduction

Definition
Intrusion: Any deliberate or accidental action that compromises security of a computer resource.

Definition
Intrusion Detection System (IDS): A hardware or software component dedicated to detecting intrusions.

Based on detection mechanism used IDSs are mainly of two types
- Signature detection system: detects known attacks
  - Use a signature database
  - Detect protocol violations and known patterns
- Anomaly detection system: detect known and unknown attacks
  - Build a profile of normal system behavior
  - Model network packet or behavior of system calls
Introduction

Based on where it is deployed there are two types of IDSs

- **Host based**: detect attacks against a single host
  - Audit log information
  - System call monitoring
  - User behavior modeling
  - File integrity checking
  - Port monitoring

- **Network based**: detect attacks against a network
  - Normally deployed at network perimeter
  - Analyze raw network packets
  - Normally some kind of traffic filtering is used in the first place

Both Host based and Network based IDSs can be either of the type
Signature detection based or Anomaly detection based
System call monitoring for Host based IDS

Monitor the running process

- Define the normal behavior of a process
- Trace the system calls of a running process to detect suspicious activity

Why system calls?

- System calls mediate between the user program and operating system
- Any compromised program must generate some system calls to cause the damage ex. creating a new process, a write operation etc. generate a system call
Model creation

To create a process execution model training is required. To train the model system calls of a process need to be collected and can be done in two ways:

- **Offline**: Exercise all possible inputs that the program takes and collect the traces.
- **Online**: Collect the system calls generated while the program is running.

**Offline technique**
- It is artificial
- Helpful for replicating experiments
- False and True positive assessment will not be realistic

**Online**:
- All possible execution paths may not appear during training period
- Difficult to judge whether collected trace is indeed normal
Motivation

Handling uncertainty over pure normal data

- Manually analyze and pick
- Every program in the machine need to be modeled: *Manual analysis is not feasible*
- Create models which can tolerate residual/accidental contamination

Sequencegram is a system which can tolerate such contamination

**Key:** Realtime monitoring is essential to understand the true behavior and also model which can tolerate contamination is needed
Sequencegram working

Model creation is also called training phase

**Key idea:** Use frequency information of a sequence to build a tolerant model

- *n*-gram-tree construction: Use short sequences of system calls and construct a tree incrementally
- Frequency binning: Create a set of bins and
- Assign an *anomaly score* to the bin
**n-gram-tree Building**

- Use a sliding window and generate short sequences of system calls between 1 to $N$
- These short sequences are called as $n$-grams
- All nodes between Root and a node at level $i$ represents an $n$-gram of order $i$
- Each node is two tuple $< a_i, F >$
- Since the common prefixes are merged it forms an efficient storage structure

Consider the sequence 1, 2, 5, 2, 3, 4, 1, 4, 2, 1, 2, 5, 2, 3, 4, 1, 4, 2
An example *n*-gram-tree

Figure: *n*-gram-tree of order 3 built with a training trace
**Frequency binning**

- Assign *anomaly score* to each short sequence
- Simple solution: individual *anomaly score* to individual $n$-grams
- Create a group of frequencies and assign a single *anomaly score*

Involves 3 sub-steps

- How many bins?
- Distribute number of $n$-grams to each bin
- Generate bin *anomaly score*
How many bins?

We use wellknown Sturges’s formula to determine appropriate number of bins

\[
\text{numberbins} = \lceil (1 + \log_2(\text{distinct n - grams})) \rceil \tag{1}
\]
How to distribute \( n \)-grams to bins

Simple technique is to divide equally

- Huge frequency variation within a bin
- Defeat the vary objective of frequency information

How we handled it?

- We used \( k \)-means clustering with \textit{numberbins} as \( k \) value
- Determine frequency range of each bin and add a \( n \)-gram to a particular bin if its frequency is within the range of that bin
K-means clustering

- It is an optimization algorithm: minimizes an objective function called squared error function
- Starts with $k$ as input and generate those many clusters at the end
- Randomly selects $k$ cluster centers and assigns elements to these centers
- Recompute the cluster centers with new cluster points
- Reassign the elements to these centers
- Continue reassigning until stabilized
## Frequency clusters generated for UNM lpr

The table below shows the frequency clusters for UNM lpr:

<table>
<thead>
<tr>
<th>Cluster No</th>
<th>Frequency Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2-3</td>
</tr>
<tr>
<td>2</td>
<td>6-10</td>
</tr>
<tr>
<td>3</td>
<td>11-21</td>
</tr>
<tr>
<td>4</td>
<td>27-58</td>
</tr>
<tr>
<td>5</td>
<td>275-766</td>
</tr>
<tr>
<td>6</td>
<td>813-830</td>
</tr>
<tr>
<td>7</td>
<td>1038-4425</td>
</tr>
<tr>
<td>8</td>
<td>20003-37823</td>
</tr>
</tbody>
</table>
Assigning *anomaly score* to bins

- Anomalous sequences are rare
- The more frequently the sequence appears lesser the *anomaly score* and vise versa

\[
rate[t] = \log_e \left( \frac{\sum_{i=1}^{\text{tracecount}} \sum_{j=1}^{\text{tracelength}} f(<a_1, \ldots, a_n, F>)}{\sum_{i=1}^{\text{tracecount}} \sum_{j=1}^{\text{tracelength}} \bin(t, f(<a_1, \ldots, a_n, F>))} \right)
\]

(2)

where \( f(<a_1, \ldots, a_n, F>) \) is a function which returns the frequency of a *n*-gram and

\( \bin(t, f(<a_1, \ldots, a_n, F>)) \) is a function which returns the frequency of a *n*-gram if it falls in bin \( t \) and 0 otherwise
anomaly scores generated

Table: Example *anomaly score* Generation and Assignment

<table>
<thead>
<tr>
<th>Grand Sum</th>
<th>Bin Sum</th>
<th>Number of <em>n</em>-grams</th>
<th><em>anomaly score</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>2920898</td>
<td>848</td>
<td>72</td>
<td>8.14</td>
</tr>
<tr>
<td></td>
<td>2393</td>
<td>19</td>
<td>6.90</td>
</tr>
<tr>
<td></td>
<td>7645</td>
<td>11</td>
<td>5.94</td>
</tr>
<tr>
<td></td>
<td>20540</td>
<td>21</td>
<td>4.96</td>
</tr>
<tr>
<td></td>
<td>65787</td>
<td>35</td>
<td>3.78</td>
</tr>
<tr>
<td></td>
<td>100632</td>
<td>16</td>
<td>3.36</td>
</tr>
<tr>
<td></td>
<td>173767</td>
<td>64</td>
<td>2.77</td>
</tr>
<tr>
<td></td>
<td>2449417</td>
<td>2</td>
<td>0.17</td>
</tr>
</tbody>
</table>
Testing

- Given test trace $S$ generate $n$-grams
- Search each $n$-gram in $n$-gram-tree
- Find the frequency of each $n$-gram and also the bin to which it falls (it may be NULL if that $n$-gram do not appear before)
- Assign the anomaly score of bin to $n$-gram

$$S_{anomaly} = \sum_{J=1}^{\text{numbergrams}} \frac{\text{anomaly score of } p_J}{\text{numbergrams}}$$  \hspace{1cm} (3)

Note: If a $n$-gram do not appear in the tree it will be assigned twice the maximum anomaly score among all $n$-grams
Complexity

Notations used.

\[ A \quad = \quad \text{alphabet of system calls for a program} \]
\[ \text{size} \quad = \quad \text{cardinality of set } A \]
\[ \text{tracecount} \quad = \quad \text{number of program traces available for training} \]
\[ S \quad = \quad \text{set of training traces } S_0, S_1, \cdots, S_{\text{tracecount}} \]
\[ \text{tracelength} \quad = \quad \text{length of the program trace} \]
\[ N \quad = \quad \text{order of } n\text{-grams to be generated} \]
\[ Y \quad = \quad \text{number of iterations in clustering} \]

Time complexity of training phase is
\[ O(\left(\text{tracecount}\right) \cdot N \cdot \left(\text{tracelength} + (\text{tracelength})^2\right) + \sum_{i=1}^{N} ((\text{tracelength}) \cdot \text{size}^i) \cdot ((\text{numberbins}) \cdot \text{size}^N \cdot Y) \]

Time complexity of testing phase is
\[ O((\text{tracelength}) + ((\text{numbergrams}) \cdot (size \cdot N)) + (\text{numbergram})) \]
Experimental results

- Dataset shared by University of New Mexico (Forrest et al)
- Detection Rate and False Positive Rate as evaluation metric
- Each experiment is a 10 fold cross validation
- What anomaly score is a real anomaly?

Note: Calculate the False Positive Rate when the Detection Rate is 100%
## Results on pure dataset

**Table:** *False Positive Rate* for 10 fold cross validation experiment to achieve 100% *Detection Rate*

<table>
<thead>
<tr>
<th>Program</th>
<th>Number of Normal Traces</th>
<th>Number of Intrusion Traces</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT lpr</td>
<td>2703</td>
<td>1001</td>
<td>1.70%</td>
</tr>
<tr>
<td>UNM lpr</td>
<td>1232</td>
<td>1001</td>
<td>0.00%</td>
</tr>
<tr>
<td>xlock real</td>
<td>220</td>
<td>2</td>
<td>0.00%</td>
</tr>
<tr>
<td>named</td>
<td>462</td>
<td>2</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Results on impure dataset

Table: False Positive Rate for 10 fold cross validation of MIT lpr impure dataset when Detection Rate is 100%

<table>
<thead>
<tr>
<th>Number of Normal Traces</th>
<th>Number of Intrusion Traces Added</th>
<th>Number of Test Traces</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2703</td>
<td>25</td>
<td>1001</td>
<td>1.70%</td>
</tr>
<tr>
<td>2703</td>
<td>50</td>
<td>1001</td>
<td>1.70%</td>
</tr>
<tr>
<td>2703</td>
<td>100</td>
<td>1001</td>
<td>1.70%</td>
</tr>
<tr>
<td>2703</td>
<td>200</td>
<td>1001</td>
<td>1.82%</td>
</tr>
<tr>
<td>2703</td>
<td>300</td>
<td>1001</td>
<td>1.82%</td>
</tr>
<tr>
<td>2703</td>
<td>500</td>
<td>1001</td>
<td>3.41%</td>
</tr>
<tr>
<td>2703</td>
<td>1001</td>
<td>1001</td>
<td>99.80%</td>
</tr>
</tbody>
</table>
## Results on impure dataset

Table: *False Positive Rate* for 10 fold cross validation of UNM lpr impure dataset when *Detection Rate* is 100%

<table>
<thead>
<tr>
<th>Number of Normal Traces</th>
<th>Number of Intrusion Traces Added</th>
<th>Number of Intrusion Traces Tested</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1232</td>
<td>25</td>
<td>1001</td>
<td>0.00%</td>
</tr>
<tr>
<td>1232</td>
<td>50</td>
<td>1001</td>
<td>0.00%</td>
</tr>
<tr>
<td>1232</td>
<td>100</td>
<td>1001</td>
<td>1.45%</td>
</tr>
<tr>
<td>1232</td>
<td>200</td>
<td>1001</td>
<td>1.79%</td>
</tr>
<tr>
<td>1232</td>
<td>300</td>
<td>1001</td>
<td>4.64%</td>
</tr>
<tr>
<td>1232</td>
<td>500</td>
<td>1001</td>
<td>14.79%</td>
</tr>
<tr>
<td>1232</td>
<td>1001</td>
<td>1001</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
Results on impure dataset

Table: False Positive Rate for 10 fold cross validation of named impure dataset when Detection Rate is 100%

<table>
<thead>
<tr>
<th>Number of Normal Traces</th>
<th>Number of Intrusion Traces Added</th>
<th>Number of Intrusion Traces Tested</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>462</td>
<td>1</td>
<td>2</td>
<td>0.00%</td>
</tr>
<tr>
<td>462</td>
<td>2</td>
<td>2</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Results on impure dataset

Table: False Positive Rate for 10 fold cross validation of xlock real impure dataset when Detection Rate is 100%

<table>
<thead>
<tr>
<th>Number of Normal Traces</th>
<th>Number of Intrusion Traces Added</th>
<th>Number of Intrusion Traces Tested</th>
<th>False Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>220</td>
<td>1</td>
<td>2</td>
<td>0.00%</td>
</tr>
<tr>
<td>220</td>
<td>2</td>
<td>2</td>
<td>0.00%</td>
</tr>
</tbody>
</table>
Practical time and space complexity

Table: Short sequence growth

<table>
<thead>
<tr>
<th>Trace</th>
<th>$n$-gram 1st order</th>
<th>$n$-gram 2nd order</th>
<th>$n$-gram 3rd order</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIT lpr</td>
<td>41</td>
<td>141</td>
<td>250</td>
</tr>
<tr>
<td>UNM lpr</td>
<td>15</td>
<td>71</td>
<td>118</td>
</tr>
<tr>
<td>named real</td>
<td>41</td>
<td>212</td>
<td>433</td>
</tr>
<tr>
<td>xlock real</td>
<td>36</td>
<td>111</td>
<td>188</td>
</tr>
</tbody>
</table>

Our proof of concept unoptimized version of Sequencegram process 1234 UNM /lpr training traces in 60ms (build the $n$-gram-tree, does frequency binning and calculating the anomaly score of each bin).
Software

Implementation: A single C program (Sequencegram.c) released under GNU public licence
Can be found at my webpage ("www.iitg.ernet.in/stud/neminath/")
Summary

- Presented **Sequencegram** a program based anomaly detection technique
- It models the short sequences of system calls with their occurrence frequency
- Results show **Sequencegram** work with impure training data
- Analyzed time and space complexity of algorithm

Further plans

- Collecting our own traces in production systems
- Experiments on varied and complex programs such as WEB Server, FTP Server
Modeling system calls for intrusion detection with dynamic window sizes.  

S. Forrest, S. A. Hofmeyr, A. Somayaji, and T.A. Longstaff.  
A sense of self for unix processes.  

S. A. Hofmeyr, S. Forrest, and A. Somayaji.  
Intrusion detection using sequences of system calls.  

C. C. Michael and A. Ghosh.  
Simple, state-based approaches to program-based anomaly detection.

A. Somayaji and S. Forrest.
Automated response using system-call delays.