A Sense of Self for Unix Processes

Stepannie Forrest, Steven A. Hofmeyr, Anil Somayaji, Thomas A. Longstaff

Presenter: Ge Ruan
Overview

- This paper presents an intrusion detection algorithm which is learned from mechanisms of natural immune systems.
- In natural immune system, pattern recognition is used to check whether a cell is normal or abnormal?
- So, how to define the pattern of normal or “Self” is the main focus of this paper.
Definition of Self

- What we mean self in computer system is more dynamic than in the case of natural immune systems.
  - Load updated software
  - Edit files
  - Run new programs
  - Change work habits

In these cases, the normal behavior of the system is changed, sometimes dramatically.
Definition of Self (Cont’d)

- Requirement of a successful definition
  - It must accommodate legitimate activities
  - It must be sensitive to dangerous foreign activities.
Definition of Self (Cont’d)

- The experiments show that short sequences of system calls in running processes generate a stable signature for normal behavior
  - This signature has low variance over a wide range of normal operating condition, and is specific to each different processes.
  - The signature has a high probability of being perturbed when abnormal activities.
Classification of Intrusion Detection

- There are two basic approaches to intrusion detection based on different definition
  - Based on the definition of normal behavior (anomaly intrusion detection)
  - Based on the prior knowledge about the specific form of intrusion (misused intrusion detection)
- In this paper, the author only concern with anomaly detection.
Related Work

- IDES: combining both approaches
- IDES, TIM: Slowly adaptive approach, changing profiles gradually to accommodate changing user behavior.
- Approach of Fink, Levitt and Ko: Instead of trying to build up normal user profiles, they focus on determining normal behavior for privileged processes.
The Author’s Approach

- Their approach is similar to Fink’s approach. They only focus on root processes.
- However it differs in that a much simpler representation of normal behavior is used. They rely on examples of normal runs rather than formal specification of a program’s expected behavior. So they do not have to determine a behavioral specification from the program code but simply accumulate it by tracing normal runs of the program.
The Author’s Approach (Cont’d)

- They define behavior in terms of short sequences of system calls in running process.
- Once a stable database is constructed for a given process, the database can be used to monitor the process’ ongoing behavior.
- So the sequence of system calls form the set of normal patterns for database, and abnormal sequences indicate anomalies in running process.
Details of the Algorithm

The algorithm have two stages

- Scan traces of normal behavior and build up a database of characteristic normal patterns, i.e. observed sequences of system calls.

- Scan new traces that might contain abnormal behavior, looking for patterns not present in the normal database.
Suppose $k=3$, and are given the following sequence of system class to define normal behavior: `open`, `read`, `mmap`, `mmap`, `open`, `getrlimit`, `mmap`, `close`.

After sliding the window across the complete sequence, they produce this expanded database.
Details of the Algorithm (Cont’d)

- If we were given a new trace of calls, differing at one location from normal:
  
  \textit{open, read, mmap, open, open, getrlimit, mmap, close}
  
- Also sliding the window across this sequence, then this trace would generate 4 mismatches out of maximum database size of 18, i.e. the miss rate is 22%. If this exceed the threshold, it is regarded as an abnormal activity.
Experiment Details

There are some questions we are cared about in the experiment

- Dataset size?
  - If it is small, then it defines a compact signature, that would be practical to check in real-time.
  - Conversely, if the database is large then the approach will be too expensive to use for on-line monitoring. On the other hand, too much variability in normal would preclude detecting anomalies.
Experiment Details (Cont’d)

- Threshold?
  - Too small: result in false alarm
  - Too big: result in undetect of some abnormal activity.
Does this definition of normal behavior distinguish between different kinds of programs?

It is obvious that the processes except *sendmail* have a significant number of abnormal sequences.
Experiment Details (Cont’d)

- Does this definition detect anomalous behavior?

In the table, three sources of abnormal and detection result are shown

- Traces of successful sendmail intrusion: sunsendmailcp, syslog, decode, lprecp
- Traces of sendmail intrusion attempts that failed: sm565a, sm5x
- Traces of error conditions: forward loop
Experiment Details (Cont’d)

- Most of the successful intrusions are detected except decode attack.
- The percentage of abnormal sequences of unsuccessful attack is on the low end of the range for successful attacks.
- Error condition differs from normal by a small but clear percentage.

<table>
<thead>
<tr>
<th>Anomaly</th>
<th>5</th>
<th>6</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>%</td>
<td>#</td>
<td>%</td>
</tr>
<tr>
<td>sunsendmailcp</td>
<td>3.8</td>
<td>72</td>
<td>4.1</td>
</tr>
<tr>
<td>syslog:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remote 1</td>
<td>3.9</td>
<td>361</td>
<td>4.2</td>
</tr>
<tr>
<td>remote 2</td>
<td>1.4</td>
<td>111</td>
<td>1.5</td>
</tr>
<tr>
<td>local 1</td>
<td>3.0</td>
<td>235</td>
<td>4.2</td>
</tr>
<tr>
<td>local 2</td>
<td>4.1</td>
<td>307</td>
<td>3.4</td>
</tr>
<tr>
<td>decode</td>
<td>0.3</td>
<td>21</td>
<td>0.3</td>
</tr>
<tr>
<td>lprcp</td>
<td>1.1</td>
<td>7</td>
<td>1.4</td>
</tr>
<tr>
<td>sm565a</td>
<td>0.4</td>
<td>27</td>
<td>0.4</td>
</tr>
<tr>
<td>sm5x</td>
<td>1.4</td>
<td>110</td>
<td>1.7</td>
</tr>
<tr>
<td>forward loop</td>
<td>1.6</td>
<td>43</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Discussion

- The approach is predicated on two important properties:
  - The sequence of system calls executed by a program is locally consistent during normal operation.
    Because code of most programs is static, and system calls occur at fixed places within code.
  - Some unusual short sequences of system calls will be executed when a security hole in a program is exploited.
    Even the attack is successful without being detected, it would likely execute a sequence of system calls not in the database. Finally, it is highly likely a successful intruder will need to fork a new process in order to take advantage of the system. And these should be detectable.
Discussion (Cont’d)

- However, there are some intrusion does not fit into either of these two categories, the presented algorithm will certainly miss it.
  - For example, the race condition attacks. These types of attack steal a resource created by root program before the program restrict access to the resource.
  - An intruder using another user’s account.
Conclusion

■ Deficiency:

The algorithm will not provide a cryptographically strong or completely reliable discriminator between normal and abnormal behavior.

■ Advantage (promising):

  ➢ Simple, practical and real-time
  ➢ The method used to collect normal traces allows for a unique database at each site. Thus, a successful intrusion at one site would not necessarily be successful at all sites running the same software.
Future Work

- Using partial or approximate matching.
- Using on-line learning
- Incorporating different mechanisms to provide more comprehensive security.
Questions?
Thank You!