Disruption Tolerance in a Distributed E-Commerce System

Saurabh Bagchi
Dependable Computing Systems Lab
School of Electrical and Computer Engineering
Purdue University
sbagchi@purdue.edu

http://shay.ecn.purdue.edu/~dcsl

Outline

- What is disruption tolerance & Motivation
- Adaptative Disruption Tolerant System
- Disruption Detection
- Our Approach
- System Design
- Results
- Disruption Containment and Response
- Conclusions
What is Disruption Tolerance?

- Causes of system downtime
  - Naturally occurring failures: hardware, software, interfaces
  - Malicious intrusions: internal, external
- Disruption = Failure + Intrusion
- Similarities in approach to tolerate the two causes
  - Both cause system to be unavailable or degraded in functionality
  - Sometimes root cause cannot be distinguished
  - Sometimes response is identical (e.g., take component offline and bring in a diverse spare)
- Dissimilarities in approach to tolerate the two causes
  - Number of coincident events
  - Counter-response

Motivation

- Handling failures and intrusions under same framework gives the following advantages
  - Reduce overhead: Example – A separate detection routine for each subsystem is not required
  - Leverage synergy between two actions: Example – A component that is compromised due to an intrusion need not be recovered from a natural fault
- What is tolerating disruption?
  - Not enough to simply detect: Large volume of intrusion detection systems, error detection protocols
  - Need to address the other phases of the process: Diagnosis, Containment, Response
  - The phases are closely coupled in their cost metrics: A pinpointed diagnosis reduces the cost of recovery
**Our Approach: Adaptive Disruption Tolerant System (ADTS)**

- Application Layer
- Information Systems Layer
- Network Layer

**Disruption Detection**

- Initial phase of disruption tolerance process
- Based on previous approach to intrusion detection systems (IDS)
- IDSs are based on two alternative choices
  - Anomaly based: Specify the normal behavior
  - Misuse based: Specify the patterns of attacks
- Metrics for evaluating IDSs
  - False positives, or False alarms
  - False negatives, or Missing alarms
- Our approach: Collaborative Disruption Detection Systems (CODDS)
  - Multiple detectors specialized for different parts of system
  - Manager infrastructure for combining alarms from multiple detectors
  - Rulebase at manager to decide on appropriate response
CODDS Approach

- Motivation
  - Single IDS can have false positives (false alarms) or false negatives (missed alarms)
  - Single IDS is specialized for certain kinds of attacks
  - Timing based correlation from multiple detectors may indicate useful characteristics of attack such as propagation speed

CODDS Components

- Elementary Detectors (EDs): Specialized detectors distributed through the system
  - The EDs may be off-the-shelf and minimal change is required for integration into CODDS
  - Different hosts may have different configurations of EDs
- Message Queue (MQ): Communication layer for multiple CODDS components
  - Secure through a shared key and hash digest
- Connection Tracker (CT): Kernel level entity to track which process has active connection on which port
- Manager: Workhorse of CODDS responsible for collating alerts from EDs and generating a combined alert which is expected to be more accurate
  - Can take into account local alerts from individual hosts to make a global determination
**Manager Architecture**

- Manager communicates with other entities through MQ and has shared key with each ED
- Manager components are
  - Translation engine: Translates native alert formats into CODDS format
  - Event dispatcher: Dispatches the event to the appropriate host’s Inference Engine instance
  - Inference Engine: Matches the received events against the Rule Objects to come up with a determination of disruption.
    - A separate instance of the Local Inference Engine for each host
    - A Global Inference Engine for correlating the results from the local engines
    - Rule Objects store the rules, one for each class of disruption
  - Combining Engine: If multiple types of inference engine, this combines the detection decision from each
Graph-Based Inference Engine

- Rules are represented as graphs
- Nodes are events and Edges represent sequencing of events
- Edge weights represent assurance values indicating likelihood of sequence

- Assurance Value (AV) for a disruption given by sum of edge weights
- An event is matched with a rule object if it is fusionable, i.e., belongs to the same disruption instance
- Discounted Assurance Value (DAV) for partial matches
  \[ DAV = AV \times (\frac{\text{Partial path length}}{\text{Complete path length}}) \]

Bayesian Network Based Inference Engine

- In a Bayesian Network, the nodes represent random variables and edges the direct influence of one variable on another
- Three step process for creating rule object
  - Nodes to represent events
  - Edges to represent conditional probability relations among the events
  - Creation of table with conditional probability values

- Bayesian Network toolbox used for solving
- Input is fusionable event stream
- Output is conditional probability of root (the disruption)
CODDS System: Current Implementation

Apache web server
Libsafe
Snort
Sysmonitor
Countermeasure agent
Manager
Netfilter
Linux Kernel

External Network
Internal Network

CODDS Elementary Detectors

- Application level: *Libsafe*. Middleware to intercept “unsafe” C function calls and prevent stack overflow attacks.
- Network level: *Snort*. Sniffs on incoming network packets and matches against rulebase to perform misuse based detection.
  - Intercepts system calls for file accesses and executions.
  - Takes a set of rules for disallowed accesses or executions
    - Can be specified using wildcards or directory tree
  - Intercepts signals of interest that can flag illegal operations.
    - SIG_SEGV to indicate segmentation violation that may be caused by buffer overflow
Simulated Disruptions

- Disruptions which are of type intrusions are simulated for our experiments.
- Three classes of disruptions, multiple types within each class, and multiple variants within each type
  - Buffer overflow: Can be used to overwrite parts of stack and write and execute malicious code
    - Apache chunk attack
    - Open SSL attack
  - Flooding: Overwhelm the network with redundant or malicious packets causing a denial of service
    - Ping flood
    - Smurf
  - Script based: Exploit poorly written scripts which do not do input validation to execute arbitrary commands
    - Used unchecked `open()` and `system()` calls

Results: Performance – Without Disruptions

- Measured without and with disruptions
- 30 web clients running concurrently
- Transactions per second of workload transaction measured
- When multiple EDs present, manager with both Inference Engines is deployed

No Disruption

- Degradation overall: 3.95% with Snort rules modified, 5.60% without
- Degradation due to Sysmon alone: 3.46%
Results: Performance – With Disruptions

- OpenSSL Attack performance degradation is 6.33%
- Chunk Attack performance improves!!!
  - Having Libsafe prevents core dumping
- Highest performance degradation due to Matlab Bayesian Network toolbox

Results: Accuracy of Detection

<table>
<thead>
<tr>
<th></th>
<th>Snort</th>
<th>Libsafe</th>
<th>Sysmon (Signal)</th>
<th>Sysmon (File)</th>
<th>CIDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No attacks</td>
<td>Yes (1807,1933)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No attack</td>
</tr>
<tr>
<td>Open SSL</td>
<td>Yes (1881,1897)</td>
<td>No</td>
<td>Yes</td>
<td>R1</td>
<td>Yes</td>
</tr>
<tr>
<td>Open SSL variant</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>R1</td>
<td>Yes</td>
</tr>
<tr>
<td>Apache Chunk</td>
<td>Yes (1807, 1808, 1809)</td>
<td>Yes</td>
<td>Yes</td>
<td>R1</td>
<td>Yes</td>
</tr>
<tr>
<td>Smurf 1000</td>
<td>Yes (499)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Smurf 500</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Ping Flooding</td>
<td>Yes (523, 1322)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Script</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Yes: Detected. Figures in parentheses are the rule numbers within Snort. Sysmon(File) is the file access detection part, Sysmon(Signal) is the illegal signal detection part; R1: The attack was not successful in creating a file.
**Disruption Containment & Response**

- Representation model used is Disruption DAG
- Algo #1: Compute the Compromised Confidence Index (CCI) of each node and classify it as candidate for response
- Algo #2: Decide on response based on CCI of node, Disruptivity Index (DI) of response and Effectiveness Index (EI) of response action

**Response Table**

<table>
<thead>
<tr>
<th>Node Class</th>
<th>Response</th>
<th>Disruptivity Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>R1</td>
<td>1</td>
</tr>
<tr>
<td>WC</td>
<td>R2</td>
<td>2</td>
</tr>
<tr>
<td>YWC</td>
<td>R3</td>
<td>3</td>
</tr>
<tr>
<td>SC</td>
<td>R4</td>
<td>4</td>
</tr>
</tbody>
</table>

**Conclusion**

- Goal is to provide tolerance for causes of system downtime, be it natural failures or malicious intrusions
- Detection is the first phase and is best done by using multiple specialized detectors and combining their alerts into a system wide alert
- The combined alert is shown to be more accurate and efficient
- A CODDS Manager designed for the system
- A distributed e-commerce based platform used for demonstration
- Simply detection is not enough, containment and response are subsequent phases that are also important