Honeynet Data Analysis:
A technique for correlating sebek and network data

Edward G. Balas
Indiana University
Advanced Network Management Lab
6/15/2004
About the Author

• Edward G. Balas
  – Security Researcher at Indiana University’s Advanced Network Management Lab.
  – Honeynet Project Member
    • Sebek project lead
    • Honeywall User Interface project lead

• Research Sponsorship
  • This materials based on research sponsored by the Air Force Research Laboratory under agreement number F30602-02-2-0221. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.
Abstract

• Honeynets contain 2 data sources of interest to this talk.
  – Packet Captures
  – Sebek Logs
• Currently there is no way to
  – relate a process to a given network flow
  – identify if a given process is a descendant of another.
• We describe our efforts to solve this problem by tracking additional system calls in Sebek and how this can lead to new ways of visualizing honeynet data.
Introduction
The Problem

- Packet captures historically represented the fundamental data source for honeynets.
- To thwart observation, many blackhats began to use session encryption.
- Sebek was developed to circumvent this encryption.
- This new data source increased amount of data for the analyst.
- Each data source had its own analysis tools.
- This created a “needle in the needle stack” problem.
Trying to solve the problem

• Within Sebek, we monitored additional system calls to enable the relation of network data to sebek data. This allowed us to:
  – map a flow to a process
  – create directed graph of process execution

• Combining these capabilities meant we were able to relate 2 connections that share a common process tree.
  – useful for identifying an intrusion for which Snort has no signature but where we observed an outbound connection
Remainder of the Talk

– Related Works
– Details on our enhancements to Sebek
– How these enhancements improve Data Analysis.
– Current status and Future work.
Related work
Sebek

• **Sebek Data Capture tool**
  – kernel space tool that monitors `sys_read` call
  – covertly exports data to server.
  – used to monitor keystrokes, recover files, and other related activities even when session encryption used.
Sebek Illustrations

- top left shows general architecture
- bottom left provides illustration of how Sebek gains access to sys_read data.
CoVirt

• CoVirt and the BackTracker system
  – Enhanced UML system allows host to monitor guests system call activity.
  – “Automatically identifies potential sequences of steps that occurred in an intrusion.”
BackTracker output
Enhancements to Sebek
sys_socket monitoring

• To correlate network connections to process / file number we added the ability to monitor the sys_socket call.
  – in Linux, all socket calls are multiplexed through one generic socket call.
  – gained access using the same technique as used with sys_read.
  – this provided a mapping of:
    • src/dst ip endpoints for a connection
    • src/dst ports and protocol
    • state of connection.
    • Related Process, File No, etc.
Socket Tracking: TCP

- pretty straight forward.
- We have the advantage of knowing the state
- `sys_connect` for outbound connections
- `sys_accept` for inbound connections
Socket Tracking: UDP

- a bit messy
- For a given socket each UDP message sent can have different endpoint info depending on the socket calls used.
- For every call below we need to send a record to the server
  - sendto
  - recvfrom
  - sendmsg
  - resvmsg
- Need to watch this as a potential bottleneck
Parent PID tracking

- To record the process inheritance tree we modified the `sys_read` monitoring such that it also recorded the parent process ID.
  - Quick and Dirty
  - Not so robust when a series of forks occur without the process doing any `read()`s.
  - Next step will be to send dummy records on `fork()`.
How our approach differs from BackTracker

- Exists entirely in kernel space works on a single physical host.
- Focus on sockets not files, will revisit files in future.
- Capable of relating network data to process data, which will enable relation of IDS events to a specific process.
- Full content of sys_read recorded.
How our approach is similar to BackTracker

- Host level process tree recreation is key in building a fault tree in both approaches.
- Both use syscall monitoring
  - we do it from within kernel
  - CoVirt does it outside the kernel
- we will eventually want to add file system centric capabilities of BackTracker.
Attacking Sebek

As with BackTracker, there are three types of attacks to be concerned with.

1. attacking the sebek infrastructure directly
2. circumvent monitoring by using system calls not monitored by sebek
3. generate large event graphs to obfuscate activities on a system
Attacking Sebek Directly

- DoS attack on client by doing very large number of small reads
  - sebek client will drop packets as network becomes saturated on a bps or pps basis.
- Current Linux Sebek is kernel module based, a number of articles have outlined how to disable the sebek client by rewriting the system call table.
Circumventing Monitoring

- Dorseif, Holz, and Klien outline a technique for avoiding `sys_read` through the use of `mmap`.
  - does not work for pipes or devices but for special built tools would allow intruder to access files, currently.
- Current client only tracks UDP and TCP sockets, raw sockets are not tracked.
  - if intruder uses `libnet/pcap` based applications to communicate we will not be able to track them currently.
Creating noise to hide the signal

• It is possible that an intruder would be able to generate a sufficiently large process / event sequence without causing a DoS.
  – This would be an attack on the ability of the user interface to adequately render the data for the analyst.
References to attack techniques:


Data Analysis
Three steps in analysis

- Collect/Screen
  - Identify raw data of interest

- Coalesce
  - Combine data from different data sources, identifying cross data source relations and providing some type of normalized access to the data.

- Report
  - Identify central themes, screen out superfluous data.
Iteration in Analysis

• Though I show the screening as happening before coalescing, ideally its more of an act of convergence.
  1. screen data
  2. coalesce data
  3. look for interesting “thread”
  4. Goto 1, screen on “thread” properties
How it is done today

- Each data type has its own analysis tool
  - causing a bit of a stovepipe effect.
  - each data set goes through the 3 steps in isolation.
- Switching between data sources requires a wetware context switch.
- Relations between data manually discovered and expressed to each tool for screening by analyst.
- No automatic way to track interesting sequences across data sources.
Why this is no good

- Labor intensive
  - I am lazy
- Error Prone
  - I am sloppy
- Lots of menial work being done by a human
  - I paid a lot for this computer
Who’s putting out the effort?

<table>
<thead>
<tr>
<th>Task</th>
<th>Human</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Screening</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Data Coalescing</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Reporting</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>
Where we want to be

- We want to shift the Screening and Coalescing burden away from the human and onto the computer.
- Focus human effort on tasks best suited to the human.
- Provide an interface that supports the analyst’s workflow.
  - primary goal is to support cross data source event sequence tracking.
Improving Data Analysis

• The new data coming from sebek allows us to automatically relate network and sebek data.
• To automate coalescing we developed a backend daemon called Hflow.
• To demonstrate the impact of these capabilities on reporting, we developed a web based user interface named Walleye.
The challenge facing HFlow

Honeynet Datastore

- IDS
- Netflow
- Remote OS ID
- Sebek IDS

Raw Data
- Packet Captures
- Sebek
- Firewall Logs
- Syslog

Derived Data
Hflow Overview

- Multithreaded flow daemon
- Automates the process of Data Coalescing.
- Inputs:
  - pcap data
  - Snort IDS events.
  - Sebek socket records.
  - p0f OS fingerprints.
- Outputs:
  - normalized honeynet network data uploaded into relational database.
Hflow Illustration
Artistic partial rendition of schema
What this gives us.

- **Automatic identification**
  - Type of OS initiating a flow
  - IDS events related to a flow
  - Honeypot processes and File Numbers related to a given flow.

- **Flow data acts as an index to the pcap data**
  - Central theme of an event sequence can be identified without having to examining packet traces.
  - When packet traces needed, flow info helps facilitate retrieval.
Potential Attack Vector?

- Hflow’s in memory flow cache could be the target of resource exhaustion attack.
  - uses a single stage hash, there is configured limit on size of table, when table exceeds a threshold, Hflow aggressively purges single packet flows.
  - In future Hflow may use multistage with different hash functions or provide some other adaptive mechanism

- Wonder how robust Argus or Netflow collectors are to this type of attack?
Reporting with Walleye

• perl based web interface
• provides unified view
  – connection oriented flow pairs
  – IDS events
  – OS Fingerprints
• Allows user to jump from network to host data.
• Visualizes multiple data types together.
# Walleye Alpha

**Events related to .26 For the 6/2/2004 18:00**

<table>
<thead>
<tr>
<th>Event Time</th>
<th>Event Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>18:06:54</td>
<td>TCP 0.225 KB 4 pps</td>
<td>33171</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.225 KB 4 pps</td>
<td>33172</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33173</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33174</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33175</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33176</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33177</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33178</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33179</td>
</tr>
<tr>
<td>18:06:54</td>
<td>TCP 0.128 KB 4 pps</td>
<td>33180</td>
</tr>
</tbody>
</table>

**June 2004**

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Time</th>
<th>Event Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Looking closely

- host x.x.x.31 attacked x.x.x.25 on its https port.
- x.x.x.31 was a linux host.
- The attack matched the OpenSSL worm signature and triggered 2 additional alerts that indicate the attacker gained www and then root access.
- If we click on Proc View, we jump to a high level view of related process activity.
What you are seeing

• Display shows process trees and ids events which are “related” to a single IDS event.
  – Yellow Boxes are root processes
  – Cyan Boxes are non-root processes
  – Red Boxes are IDS events
  – Red Arrow represents direction of flow associated with event
    • Only displaying IDS related flows.

• Graph automatically generated from DB with graphviz tool from ATT.

• Notice anything odd about the graph?
Walleye tracked intrusion across 2 honeypots

• Both the .25 and .26 honeypots were running the enhanced version of Sebek.
• We are able to provide a sense of attribution in situations where all stepping stones are running Sebek.
• Based on fault tree we could then click on a yellow box and then jump into the sebek interface.
Old questions made easy

- I see an outbound connection but didn’t see an IDS alert, what was the cause?
  - walleye is able to identify all flows related via common process tree, allowing us to quickly identify potential causes of the intrusion, basically allowing us to climb the process tree.

- What happened after the intrusion?
  - This is really good for constraining the amount of sebek data the analyst needs to examine by ignoring data that is unrelated, in this case we simply traverse down the process tree.
Features

- Identify descendant flows or sebek events related to a given event.
- Identify ancestral flows or sebek events related to a given event.
- Effectively, the combination of the two allow us to filter all data which cannot be related to an event of interest.
Wrapping it up
Current Status

• Sebek
  – socket code in linux client rather stable
  – parent PID tracking currently missing some data for processes that fork and don’t read (easy to fix)

• Hflow
  – few bugs and it’s not syslog friendly

• Walleye interface
  – a few bugs, look and feel not 100% happy with
  – not yet integrated with conventional analysis tools.
  – doesn’t provide way to access raw packets
Future work

- **Sebek**
  - track fork call so that we always get a view of the process tree
  - look at various anti-anti-sebek options.
- **Hflow**
  - testing, lots of testing.
  - evaluate attack resistance
- **Walleye**
  - get UI to better support workflow
  - provide alerting
  - provide some summary reports
  - clean, debug, document
  - integrate with existing tools where sensible.
- Get everything to work on the Honeywall CDROM!
Timeline

• The goal is to have the interface and sebek enhancements released within next 6 months.
  – Walleye may become the basis for a new Honeynet Data Analysis Interface to be distributed on the Honeywall.
  – Sebek client enhancements for linux will be available in the August timeframe.
Taking this out of the Honeynet context

- Sebek is a good tool for post intrusion intelligence gathering on an intruder
- On a production box it generates great amounts of data, making it difficult to use.
- With previously mentioned enhancements, Sebek may be a more viable tool, due to its improved coalescing and screening.
- Next major release will also allow you to define what types of activity you want to record or not record.