Intrusion Detection System

Time Machine

Dynamic Application Detection
NIDS: Two generic problems

- Attack identified
  - But what happened in the past???

- Application identification
  - Only by port number!
  - Yet applications use arbitrary ports
    - Benign reasons
      - Lack administrator privileges
      - Circumvention of firewall, e.g., Skype
      - Application tunnels
    - Malicious intend
      - Evasion of security monitoring
      - E.g.: IRC based botnets on ports other than 666x/tcp
      - E.g.: ftp servers on ports other than 21/tcp
Time Machine: Motivation

- Trouble-Shooting
  - What happened before a fault?
- Security monitoring (“Forensics”)
  - Break in 2 days ago:
    - How?
    - What else have they done?
- NIDS: offloading

- Ideas
  - Keep packet traces of past events!
Time Machine: Problem

- Recording
  - High data volume, high load (CPU, disk)
    - E.g., MWN:
      - Gbps link
      - 2004: 2 TB / day
      - 2007: 4-6TB / day
      - 350 / 950 Mbps busy hour load

- Retrieval
  - Using the captured data
  - Find relevant “needle in the haystack”
Time Machine: Concept

- Buffer packets up to connection cutoff
Connection sizes: Heavy-tailed

LBL (2004):
- > 20 KB
  - 12% of connections
  - 96% of all bytes

NERSC (2004):
- > 20 KB
  - 14% of connections
  - 99.86% of bytes
TM: Capabilities (cutoff: 15KB)

- MWN: 10Gb uplink, 3-6 TB per day, 50,000 users
- 600MB memory buffer, 800GB disk buffer

![Graph showing data rate and cutoff ratio over time](image)
TM: Retention time

- Huge potential
TM: CPU utilization

- Head room available for query processing
Coupling NIDS with Time Machine
TM-NIDS: Applications

- **Semi-automatic forensics**
  - NIDS detects standard traffic on non-standard port
  - TM captures connections to file

- **Scanner supervision**
  - NIDS recognizes scanner
  - TM supplies past traffic on successful connection

- **Host supervision**
  - NIDS recognizes weird traffic from some IP
  - Instructs TM to capture / store all traffic
  - Suspends cutoff
TM-NIDS: Applications (2.)

- Offloading NIDS
  - NIDS processes HTTP requests only
    If suspicious request TM supplies response
  - NIDS processes FTP control data only
    If suspicious request TM supplies data connection

- Error correction
  - NIDS recognizes content gap
    TM supplies missing data
Coupling NIDS with Time Machine

- **Requirements**
  - Simple interface
  - NIDS processing of TM data as usual

- **Complications**
  - Two data sources
  - Duplicate data packets from both sources
  - NIDS needs to handle live as well as historic data (violates time sequence assumptions)
Query rate

- In memory queries
- Query rate increases
- Overhead and latency small
HTTP analysis

- Significant CPU gains by offloading HTTP reply
TM-NIDS: Summary

- Stand-alone *time machine*: a very powerful tool
- Coupling a *TM* with an *NIDS*: an even more powerful tool!
NIDS: Two generic problems

☑ Attack identified
  ☑ But what happened in the past???

☑ Application identification
  ☑ Only by port number!
  ☑ Yet applications use arbitrary ports
    • Benign reasons
      – Lack administrator privileges
      – Circumvention of firewall, e.g., Skype
      – Application tunnels
    • Malicious intend
      – Evasion of security monitoring
      – E.g.: IRC based botnets on ports other than 666x/tcp
      – E.g.: ftp servers on ports other than 21/tcp
Applications on non-standard ports

- Data (Oct. 2005):
  - 24 hour full packet trace from Münchner WissenschaftsNetz (MWN)
  - 3.2 TB of data in 6.3 billion pkts, 137M connections

- Application signatures from l7-filter system
- Focus on HTTP, IRC, FTP, SMTP
## Ports accounting > 1% of conns.

<table>
<thead>
<tr>
<th>Port</th>
<th>% Conns</th>
<th>% Success</th>
<th>% Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>80</td>
<td>70.82%</td>
<td>68.13%</td>
</tr>
<tr>
<td></td>
<td>445</td>
<td>3.53%</td>
<td>0.01%</td>
</tr>
<tr>
<td>Web</td>
<td>443</td>
<td>2.34%</td>
<td>2.08%</td>
</tr>
<tr>
<td>SSH</td>
<td>22</td>
<td>2.12%</td>
<td>1.75%</td>
</tr>
<tr>
<td>Mail</td>
<td>25</td>
<td>1.85%</td>
<td>1.05%</td>
</tr>
<tr>
<td></td>
<td>1042</td>
<td>1.66%</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>1433</td>
<td>1.06%</td>
<td>0.00%</td>
</tr>
<tr>
<td></td>
<td>135</td>
<td>1.04%</td>
<td>0.00%</td>
</tr>
<tr>
<td>&lt; 1024</td>
<td></td>
<td>83.68%</td>
<td>73.73%</td>
</tr>
<tr>
<td>&gt; 1024</td>
<td></td>
<td>16.32%</td>
<td>4.08%</td>
</tr>
</tbody>
</table>
Signature-based app. detection

- Port information offers no information for ports > 1024
- l7-filter system application signatures
- HTTP highly attractive for hiding other applications
- Most successful conns. trigger expected signature
- FTP higher percentage of false negatives

<table>
<thead>
<tr>
<th>Method</th>
<th>HTTP</th>
<th>IRC</th>
<th>FTP</th>
<th>SMTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port (succ.)</td>
<td>93,429K</td>
<td>75,876</td>
<td>151,700</td>
<td>1,447K</td>
</tr>
<tr>
<td>Signature</td>
<td>94,326K</td>
<td>73,962</td>
<td>125,296</td>
<td>1,416K</td>
</tr>
<tr>
<td>expected port</td>
<td>92,228K</td>
<td>71,467</td>
<td>98,017</td>
<td>1,415K</td>
</tr>
<tr>
<td>other port</td>
<td>2,126K</td>
<td>2,495</td>
<td>27,279</td>
<td>265</td>
</tr>
</tbody>
</table>
Signature detection: Well known ports

- Some connections trigger more than one signature
- Not yet wide-spread abuse
- But some misappropriate use of well known ports

<table>
<thead>
<tr>
<th>Port</th>
<th>HTTP</th>
<th>IRC</th>
<th>SMTP</th>
<th>Other</th>
<th>No match</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>92,228,291</td>
<td>59</td>
<td>0</td>
<td>41,086</td>
<td>1,158,977</td>
</tr>
<tr>
<td>666x</td>
<td>1,217</td>
<td>71,650</td>
<td>0</td>
<td>4,238</td>
<td>524</td>
</tr>
<tr>
<td>25</td>
<td>459</td>
<td>2</td>
<td>1,415,428</td>
<td>195</td>
<td>31,889</td>
</tr>
</tbody>
</table>
Architecture for dynamic analysis

- Goals
  - Detection scheme independence
  - Dynamic analysis
  - Modularity
  - Efficiency
  - Customizability

- Design (USENIX Security’06)
  - Dynamic processing path
  - Per connection dynamic analyzer trees
Reliable detection of non-standard ports

- **UCB:** 1 day
  - internal: 6
  - remote: 17
  - FTP servers: 6
  - HTTP servers: 568 (54,830)
  - IRC servers: 2
  - SMTP servers: 8

- **MWN** similar

- **Non-standard port connection**
  - **UCB:** 99% HTTP (28% Gnutella, 22% Apache)
  - **MWN:** 92% HTTP (21% BitTorrent, 20% Gnutella), 7% FTP
  - Two open HTTP proxy detected: now closed
  - SMTP server that allowed relay: now closed
Detecting IRC-based Botnets

Idea

- Botnets like IRC protocol (remote control features)
- Botnet detector on top of IRC analyser
  - Checks client nickname for typical patterns
  - Checks channel topics for typical botnet commands
  - Checks if new clients connect with IRC to identified bot-servers

Results

- MWN:
  - > 100 distinct IPs with Botnet clients
  - Now part of an automatic prevention system
- UCB:
  - 15 distinct IPs
Summary: Dynamic app. analysis

- Ideas:
  - Dynamic processing path
  - Per connection dynamic analyzer trees

- Operational at three large-scale networks
- Detected significant number of security incidents
- Bot-detection now automatically blocks IP