Detecting Targeted Attacks Using Shadow Honeypots

Kostas G. Anagnostakis

P. Akritidis, K. Xinidis, E. Markatos, A. Keromytis
Motivation

- Malicious activity an increasing problem
- Rule-based systems prevent against known attacks
- Honeypots and ADS’s can detect zero-day attacks
  - Tradeoff between accuracy and scope
Contributions

- Hybrid model best of both worlds
  - Tune ADS for low False Negatives (FN)
  - False Positives (FP) weeded out by shadow honeypot
- Defend against targeted attacks
- Protect against client-side attacks
Overview
Tightly & loosely coupled
Tightly coupled

With Server
- Most practical scenario
- Fwd suspicious requests to its shadow
- Mirror functionality and state (same address space)

With Client
- Targets passive attacks
  - Download data containing an attack
- Shadow version of the application is invoked
Tightly coupled architecture
System Workflow

1. Input arrives
   - Known Bad Input? Yes → Drop request
   - Known Bad Input? No → Update AD Model
     - Suspect Input Based on AD? Yes → Use Shadow
       - Attack Detected? Yes → Indicate Hit to AD
         - Update AD Model
         - Update Filtering Component
       - Attack Detected? No → Handle request normally
     - Suspect Input Based on AD? No → Randomly Use Shadow Anyway?
       - Yes → Use Shadow
         - Attack Detected? Yes → Indicate False Negative to AD
           - Update AD Model
         - Attack Detected? No → Handle request normally
       - No → Handle request normally
Prototype Implementation

- **web server farm**
- **shadow server**
- **internal network**
- **switch**
- **router**
- **external network**
- **IXP1200-based firewall+load balancer**
- **loosely-coupled shadow services**
- **PC-based modified snort sensors for anomaly detection**
Shadow Honeypot Creation

- Use DYBOC code transformation tool
  - Could use STEM
  - “Building a Reactive Immune System for Software Services” USENIX 05
- Focus on memory violations (buffer overflows)
- Selectively turn on defense
Shadow generation

- Original code
  ```c
  int func(char *tmp) {
    char buf[10];
    ...
    strcpy(buf, tmp);
    ...
  }
  ```

- Transformed Code
  ```c
  int func(char *tmp) {
    char *buf;
    char buf_txl[10];
    if (shadow_enable())
      buf = pmalloc(10);
    else
      buf = buf_txl;
    ...
    strcpy(buf, tmp);
    ...
    if (shadow_enable())
      pfree(buf);
  }
  ```
State Rollback

- New system call: `transaction()`
- Selectively invoked through `shadow_enable()`
  - Main processing loop, before new request
    - Indicate new transaction has begun
    - Backup memory pages (read only)
    - New page for modified pages
    - Similar to copy-on-write
  - Main processing loop, end of request
    - Indicate transaction successfully completed
    - Discard original memory pages that were modified
- Signal handler
  - Indicate that an attack has been detected
  - Discard modified memory pages, restore original
- Filesystem
Overall system performance

- Workload generated by AD to shadow
  - False positives
  - Used real traces
    - 20-30 FP/sec

- Cost of processing by Shadow Honeypot
  - Server-side: Apache
  - Client-side: Firefox
Shadow Performance - Server

Apache Benchmark

Request per second

- Original Apache
- Apache fully instrumented
Shadow Performance - Client

Mozilla Firefox Benchmark

Normalized Performance

Vanilla Firefox
Instrumented
Vanilla Firefox - Scrolling
Instrumented - Scrolling
Limitations and open issues

- Overload attacks on the system
  - High volume attacks
  - High volume FPs
- Multi-step attacks & root-cause isolation
- Effectiveness of training depends on quality of the anomaly detector
Potential Future Work

- Evaluate different components
  - Use STEM, taintcheck, MINOS
  - Different anomaly detection algorithms
  - Tune AD heuristics using feedback

- Experiment with loosely-coupled shadow honeypots
  - Integration with passive mon infrastructure
  - Focus on “high-risk” applications (file sharing, …)
Summary

- New hybrid approach to dealing with targeted attacks by combining features found today in honeypots and ADS
- Proof of concept implementation
  - Capable of sustaining workload of protected services
## PC-based sensor performance

<table>
<thead>
<tr>
<th>Detection method</th>
<th>Throughput/sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content matching</td>
<td>225 Mbit/s</td>
</tr>
<tr>
<td>APE</td>
<td>190 Mbit/s</td>
</tr>
<tr>
<td>Payload sifting</td>
<td>268 Mbit/s</td>
</tr>
</tbody>
</table>
Filtering & AD performance

- False positive vs detection rate
Filtering - IXP1200 Utilization

![Graph showing utilization of microengines for different packet sizes (64, 512, 1024, 1518 bytes) for FWD, LB, SPITTER, and LB+FWD.]