Minestrone: Testing the SOUP

Azzedine Benameur, Nathan Evans, Matthew Elder
Agenda

• Overview

• MINESTRONE:
  – Architecture
  – Static and dynamic detection technologies
  – I/O Redirection
  – External Replica Monitoring

• Test and evaluation:
  – Architecture
  – Test suite
  – Results

• Closing Thoughts
Overview

• IARPA STONESOUP Program: “Securely Taking On New Executable Software of Uncertain Provenance”

  – Develop and demonstrate technology that provides comprehensive, automated techniques that allow end users to safely execute new software of uncertain provenance

  – Addressing 8 “weakness” classes across 3 target language classes

• Team: Columbia University (PI: Angelos Keromytis) with Stanford University, George Mason University (GMU), and Symantec

• 4-year project, 3 phases
Overview: NSA Source Code Analysis Tool Evaluation

- Evaluated suite of tools (Coverity Prevent, FindBugs, Fortify SCA, GrammaTech CodeSonar, Klocwork Insight, Ounce Labs Ounce, PMD) against both C/C++ and Java vulnerability test cases in different CWE (Common Weakness Enumeration) classes

### Java “Breadth” Test Case Coverage
- No Tool: 35.7%
- One Tool: 13.2%
- Two Tools: 15.5%
- Three Tools: 19.5%
- Four Tools: 10.3%
- Five Tools: 0.0%
- Six Tools: 1.1%

### C/C++ “Breadth” Test Case Coverage
- No Tool: 41.5%
- One Tool: 12.1%
- Two Tools: 11.6%
- Three Tools: 13.0%
- Four Tools: 14.5%
- Five Tools: 7.3%
- Six Tools: 0.0%

MINESTRONE: Testing the SOUP, CSET 13
Overview: STONESOUP Program Targets

• Target classes of vulnerabilities (including example CWE numbers):
  – Number handling (e.g., integer overflow/underflow, sign conversion: #190, #191)
  – Resource drains (e.g., failure to release memory, structures, devices: #400, #404)
  – Tainted data/input validation errors (#78, #134)
  – Error handling (e.g., unhandled exceptions/error status codes: #248, #252)
  – SQL injection / command injection (#78, #89)
  – Concurrency handling (e.g., race conditions, thread safety: #362, #366)
  – Buffer overflows/underflows/out of bounds accesses/memory safety (#121, #122)
  – Null pointer errors (#476)

• Target language classes:
  – Type-safe languages (Java, C#)
  – Type-unsafe languages (C, C++)
  – Binaries (x86, Windows or Linux)
MINESTRONE

- Architecture
- Static and dynamic detection technologies
- Replica Diversification
- I/O Redirection
- External Replica Monitoring
MINESTRONE

- Architecture

- Static and dynamic detection technologies

- I/O Redirection

- External Replica Monitoring
Detection technologies: Pin-based tools (theRing)

- **REASSURE**
  - Self-contained Mechanism for Healing Software Using Rescue Points
  - Detects program crashes and gracefully recovers

- **ISR: Instruction Set Randomization**
  - Application binary is randomized
  - Shared libraries can be also randomized

- **DFT**
  - Data flow tracking
  - High performance
Detection technologies: continued

- **KLEE**
  - Symbolic Execution
  - Fine-Grained detection

- **Dyboc**
  - Source to source transformation
  - Moving stack buffers to heap
  - Custom version of malloc(): pmalloc()

- **Valgrind (baseline)**
  - State of the art
  - Memcheck
MINESTRONE

- Architecture for confinement
- Static and dynamic detection technologies
- I/O Redirection
- External Replica Monitoring
I/O Redirection: Network/Shared Memory/X11

- Paired-library:
  - Interpose_writer: writes to file from no-sec environment
  - Interpose_reader: read from file in all replicas

```c
#include <stdio.h>

int main() {
    // Code for I/O redirection
    return 0;
}
```

```c
#include <stdio.h>

int main() {
    // Code for I/O redirection
    return 0;
}
```
MINESTRONE

• Architecture for confinement

• Static and dynamic detection technologies

• I/O Redirection

• External Replica Monitoring
External Replica Monitoring

• OpenVZ allows easy replica monitoring
  – CPU from /proc/vz/vestat
  – Memory using bean counters /proc/user_beancounters
  – Network from /vz/root/${replica_id}/sys/class/net/venet0/statistics/tx_bytes

• Overhead comparison:
  – Confinement between containers
  – Fair scheduling
Test and Evaluation

- Architecture
- Test suite
- Results
Test and Evaluation Process, cont’d.

- MITRE developed testing framework and API
- We/Symantec developed the interface to interact with the test harness
Test and Evaluation

- Architecture
- Test suite
- Results
Test and Evaluation: Test suite

- Vulnerability Classes:
  - Null pointer, 113
  - Buffer Over/underflow, 231

- Input source:
  - Environment variable
  - Command line arguments
  - File
  - Network
  - Shared Memory
  - Clipboard
Test and Evaluation

- Architecture
- Test suite
- Results
## Test and Evaluation: Results

<table>
<thead>
<tr>
<th></th>
<th>DYBOC (Original)</th>
<th>DYBOC (Coccinelle)</th>
<th>DYBOC w/CIL (Coccinelle)</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td># Altered</td>
<td>43.36% (98/226)</td>
<td>9.29% (21/226)</td>
<td>31.42% (71/226)</td>
<td>8.41% (19/226)</td>
</tr>
<tr>
<td>Processed Score</td>
<td>45.58% (103/226)</td>
<td>47.79% (108/226)</td>
<td>37.61% (85/226)</td>
<td>57.08% (129/226)</td>
</tr>
<tr>
<td>Unaltered Score</td>
<td>80.47% (103/128)</td>
<td>52.68% (108/205)</td>
<td>54.84% (85/155)</td>
<td>62.32% (129/207)</td>
</tr>
</tbody>
</table>

**TABLE IV. Summary of results for DYBOCs.**
## Test and Evaluation: Results

<table>
<thead>
<tr>
<th></th>
<th>KLEE</th>
<th>DFT</th>
<th>ISR</th>
<th>REASSURE</th>
<th>dyboc</th>
<th>theRing</th>
<th>Combined</th>
<th>Valgrind</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong># Processed</strong></td>
<td>100% (114/114)</td>
<td>100% (114/114)</td>
<td>100% (114/114)</td>
<td>100% (114/114)</td>
<td>100% (114/114)</td>
<td>100% (114/114)</td>
<td>100% (114/114)</td>
<td>99.12% (113/114)</td>
</tr>
<tr>
<td><strong># Altered</strong></td>
<td>100% (114/114)</td>
<td>1.75% (2/114)</td>
<td>1.75% (2/114)</td>
<td>3.51% (4/114)</td>
<td>10.53% (12/114)</td>
<td>2.63% (3/114)</td>
<td>0.88% (1/114)</td>
<td>1.77% (2/113)</td>
</tr>
<tr>
<td><strong>Raw Score</strong></td>
<td>0.0% (0/114)</td>
<td>0.0% (0/114)</td>
<td>97.37% (111/114)</td>
<td>96.49% (110/114)</td>
<td>89.47% (102/114)</td>
<td>97.37% (111/114)</td>
<td>98.25% (112/114)</td>
<td>97.37% (111/114)</td>
</tr>
<tr>
<td><strong>Processed Score</strong></td>
<td>0.0% (0/114)</td>
<td>0.0% (0/114)</td>
<td>97.37% (111/114)</td>
<td>96.49% (110/114)</td>
<td>89.47% (102/114)</td>
<td>97.37% (111/114)</td>
<td>98.25% (112/114)</td>
<td>98.23% (111/113)</td>
</tr>
<tr>
<td><strong>Unaltered Score</strong></td>
<td>0.0% (0/112)</td>
<td>0.0% (0/112)</td>
<td>99.11% (111/112)</td>
<td>100% (110/110)</td>
<td>100% (102/102)</td>
<td>100% (111/111)</td>
<td>99.12% (112/113)</td>
<td>100% (111/111)</td>
</tr>
</tbody>
</table>

**TABLE II.**  SUMMARY OF SCORING RESULTS ON THE 114 NULL POINTER TEST CASES.
# Test and Evaluation: Results

<table>
<thead>
<tr>
<th></th>
<th>KLEE</th>
<th>DFT</th>
<th>ISR</th>
<th>REASSURE</th>
<th>DYBOC</th>
<th>theRing</th>
<th>Combined</th>
<th>Valgrind</th>
</tr>
</thead>
<tbody>
<tr>
<td># Processed</td>
<td>97.69%</td>
<td>98.23%</td>
<td>98.67%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td># Unaltered</td>
<td>23.98%</td>
<td>90.0%</td>
<td>86.55%</td>
<td>90.27%</td>
<td>56.64%</td>
<td>84.96%</td>
<td>90.27%</td>
<td>88.50%</td>
</tr>
<tr>
<td>Raw Score</td>
<td>22.12%</td>
<td>4.87%</td>
<td>37.61%</td>
<td>4.87%</td>
<td>45.58%</td>
<td>45.58%</td>
<td>63.72%</td>
<td>47.35%</td>
</tr>
<tr>
<td>Processed Score</td>
<td>22.62%</td>
<td>4.95%</td>
<td>38.12%</td>
<td>4.87%</td>
<td>45.58%</td>
<td>45.58%</td>
<td>63.72%</td>
<td>47.35%</td>
</tr>
<tr>
<td>Unaltered Score</td>
<td>94.34%</td>
<td>5.5%</td>
<td>44.04%</td>
<td>5.50%</td>
<td>80.47%</td>
<td>53.65%</td>
<td>70.59%</td>
<td>53.50%</td>
</tr>
</tbody>
</table>

**TABLE I.** Summary of scoring results on the 226 memory corruption test cases.
## Test and Evaluation: Results

<table>
<thead>
<tr>
<th></th>
<th>KLEE</th>
<th>DFT</th>
<th>ISR</th>
<th>REASSURE</th>
<th>DYBOC</th>
<th>theRing</th>
<th>Valgrind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. User CPU Jiffies</td>
<td>25086.07</td>
<td>2461.68</td>
<td>3198.04</td>
<td>1946.90</td>
<td>978.61</td>
<td>4032.69</td>
<td>1092.83</td>
</tr>
<tr>
<td></td>
<td>σ 62181.82</td>
<td>σ 899.76</td>
<td>σ 868.80</td>
<td>σ 1164.51</td>
<td>σ 420.42</td>
<td>σ 1420.39</td>
<td>σ 1034.03</td>
</tr>
<tr>
<td>Avg. System CPU Jiffies</td>
<td>414.55</td>
<td>364.51</td>
<td>667.76</td>
<td>403.65</td>
<td>201.93</td>
<td>723.12</td>
<td>86.46</td>
</tr>
<tr>
<td></td>
<td>σ 604.78</td>
<td>σ 129.08</td>
<td>σ 190.06</td>
<td>σ 315.23</td>
<td>σ 92.66</td>
<td>σ 244.34</td>
<td>σ 28.18</td>
</tr>
<tr>
<td>Avg. Phys. Mem.</td>
<td>146.99 MiB</td>
<td>36.90 MiB</td>
<td>35.70 MiB</td>
<td>39.87 MiB</td>
<td>30.31 MiB</td>
<td>39.38 MiB</td>
<td>41.65 MiB</td>
</tr>
<tr>
<td></td>
<td>σ 71.52</td>
<td>σ 5.32</td>
<td>σ 5.36</td>
<td>σ 19.70</td>
<td>σ 8.21</td>
<td>σ 5.42</td>
<td>σ 7.52</td>
</tr>
<tr>
<td>Average Runtime</td>
<td>25.59s</td>
<td>4.59s</td>
<td>6.72s</td>
<td>5.49s</td>
<td>3.87s</td>
<td>6.39s</td>
<td>3.7</td>
</tr>
<tr>
<td></td>
<td>σ 62.36</td>
<td>σ 5.47</td>
<td>σ 13.52</td>
<td>σ 13.82</td>
<td>σ 13.52</td>
<td>σ 7.41</td>
<td>σ 8.14</td>
</tr>
</tbody>
</table>

**TABLE III.** SUMMARY OF CPU AND MEMORY USAGE FOR MEMORY CORRUPTION TEST CASES.
Test and Evaluation: Results
Closing Thoughts
Lessons learned

• Symbolic execution limitations:
  – Limited model
  – Very slow when it works (observed a 2700X overhead)

• Writing test suite from scratch is tricky:
  – Stack not always initialized to 0
  – Provide the vulnerability location to establish the ground truth

• I/O Redirection/replay is not a solved problem:
  – Many implementation available, ioapps, Jockey
  – Can you build it? Do they work?

• Enterprise products are not the silver bullet:
  – Single multi-purpose tools don’t outperform single purpose tailed tools
Thank you!

Azzedine Benameur  azzedine_benameur@symantec.com
Nathan Evans       nathan_evans@symantec.com
Matthew Elder      matthew_elder@symantec.com