

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

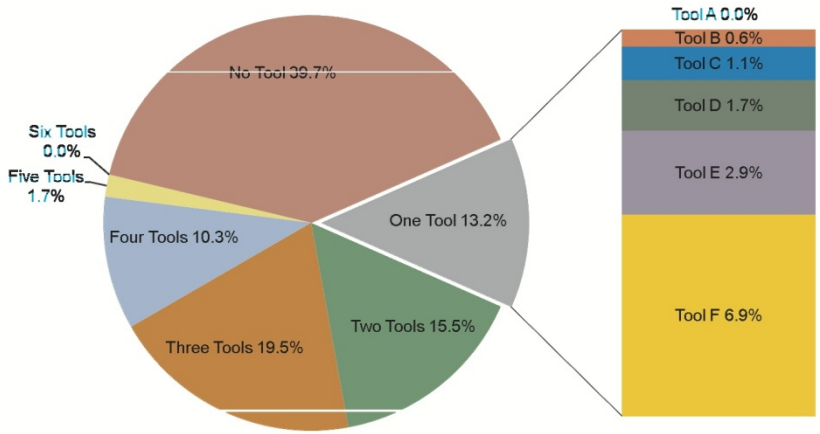
Overview

- 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

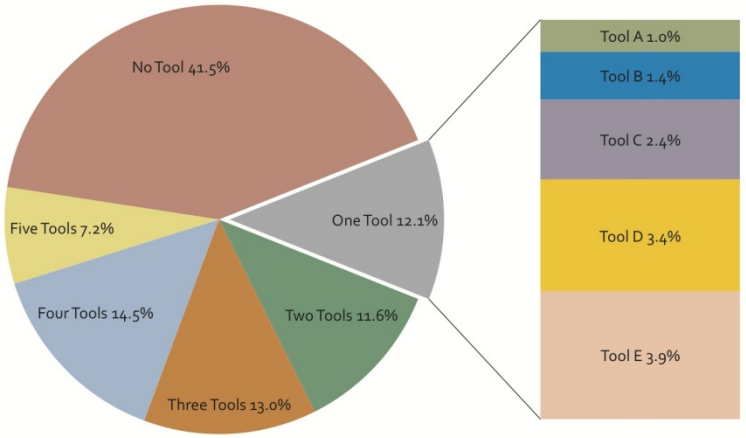
Overview: NSA Source Code Analysis Tool Evaluation

- http://www.iarpa.gov/stonesoup_Merced_DHSAWGbrief.pdf
- 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



C/C++ "Breadth" Test Case Coverage



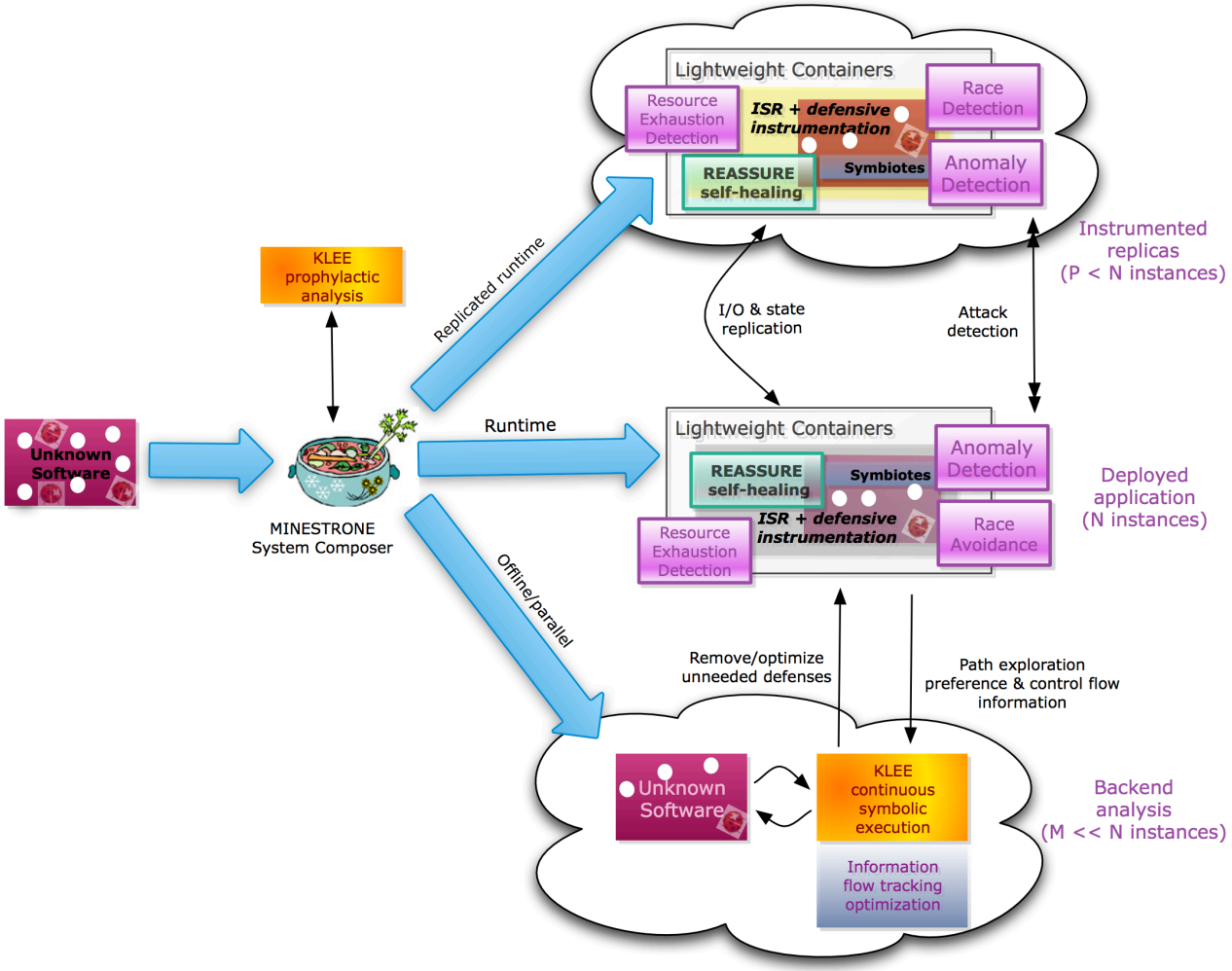
Overview: STONESOUP Program Targets

- 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 Overview: System Architecture



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

```
ssize_t recv(int sockfd, void *buf, size_t len, int flags)
{
  ssize_t ret;
  char *filename;
  ssize_t (*real_recv)(int, void *, size_t, int);
  real_recv = dlsym(RTLD_NEXT, "recv");
  ret = real_recv(sockfd, buf, len, flags);

  if (watch_sock(sockfd) == 1)
    {
#if TEST_SINGLE_FILE
      asprintf(&filename, "interpose.out");
#else
      asprintf(&filename, "%d.out", sockfd);
#endif
      write_to_interpose_file(filename, buf, ret);
      free(filename);
    }
#if VERBOSE
  fprintf (stderr, "Intercepted recv, received %d...\n", ret);
#endif
  return ret;
}
```

```
ssize_t recv (int sockfd, void *buf, size_t len, int flags)
{
  char *filename;
  ssize_t ret;
  ssize_t (*real_recv)(int, void *, size_t, int);

  if (getenv("GETADDRINFO") == NULL)
    {
#if TEST_SINGLE_FILE
      asprintf (&filename, "interpose.out");
#else
      asprintf(&filename, "%d.out", sockfd);
#endif
      ret = read_from_interpose_file (sockfd, filename, buf, len);
      if (verbose == 1)
        fprintf (stderr, "Intercepted recv, returning %d!\n", ret);
    }
  else
    {

      real_recv = dlsym(RTLD_NEXT, "recv");
      ret = real_recv(sockfd, buf, len, flags);
    }
  return ret;
}
```

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.

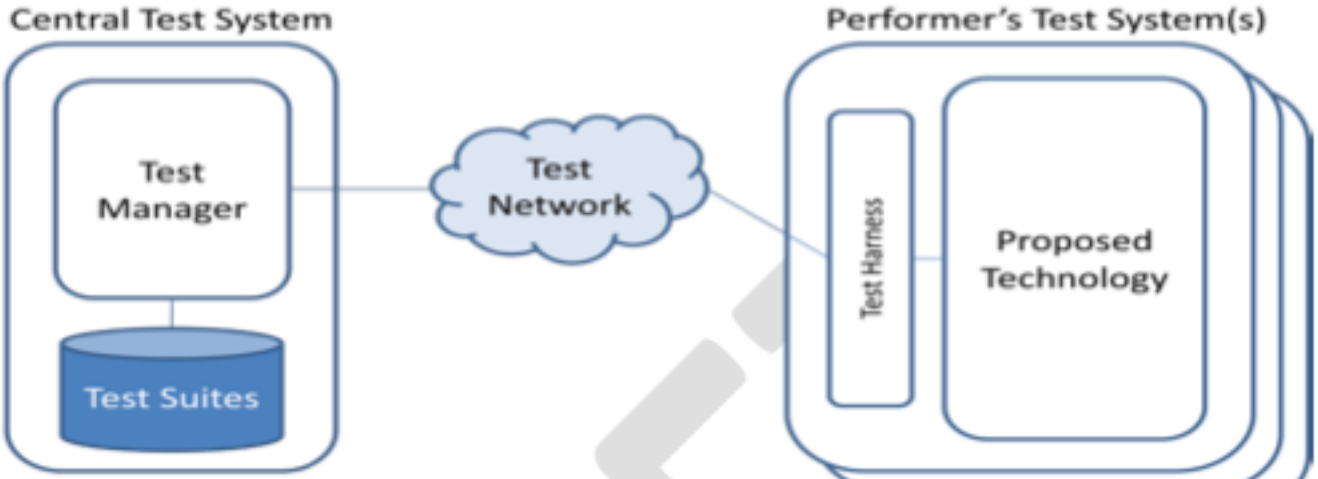


Figure 6-1 Test Environment

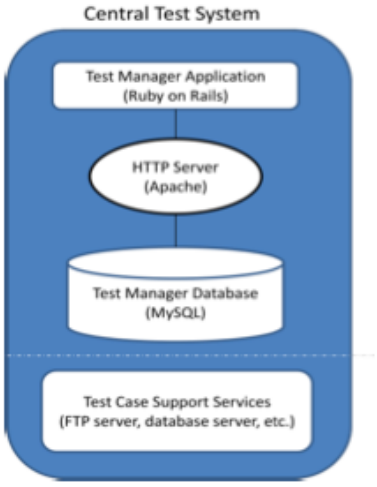


Figure 6-2 Central Test System

- 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
    http://samate.nist.gov/SRD/testsuites/stonesoup/stonesoup-c-np.zip
  - Buffer Over/underflow, 231
    http://samate.nist.gov/SRD/testsuites/stonesoup/stonesoup-c-mc.zip
- Input source:
  - Environment variable
  - Command line arguments
  - File
  - Network
  - Shared Memory
  - Clipboard

Test and Evaluation

- Architecture
- Test suite
- Results

Test and Evaluation: Results

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

TABLE IV. SUMMARY OF RESULTS FOR DYBOCS.

Test and Evaluation: Results

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

TABLE II. SUMMARY OF SCORING RESULTS ON THE 114 NULL POINTER TEST CASES.

Test and Evaluation: Results

	KLEE	DFT	ISR	REASSURE	DYBOC	theRing	Combined	Valgrind
# Processed	97.69% (221/226)	98.23% (222/226)	98.67% (223/226)	100% (226/226)	100% (226/226)	100% (226/226)	100% (226/226)	100% (226/226)
# Unaltered	23.98% (53/221)	90.0% (200/222)	86.55% (193/223)	90.27% (204/226)	56.64% (128/226)	84.96% (192/226)	90.27% (204/226)	88.50% (200/226)
Raw Score	22.12% (50/226)	4.87% (11/226)	37.61% (85/226)	4.87% (11/226)	45.58% (103/226)	45.58% (103/226)	63.72% (144/226)	47.35% (107/226)
Processed Score	22.62% (50/221)	4.95% (11/222)	38.12% (85/223)	4.87% (11/226)	45.58% (103/226)	45.58% (103/226)	63.72% (144/226)	47.35% (107/226)
Unaltered Score	94.34% (50/53)	5.5% (11/200)	44.04% (85/193)	5.50% (11/200)	80.47% (103/128)	53.65% (103/192)	70.59% (144/204)	53.50% (107/200)

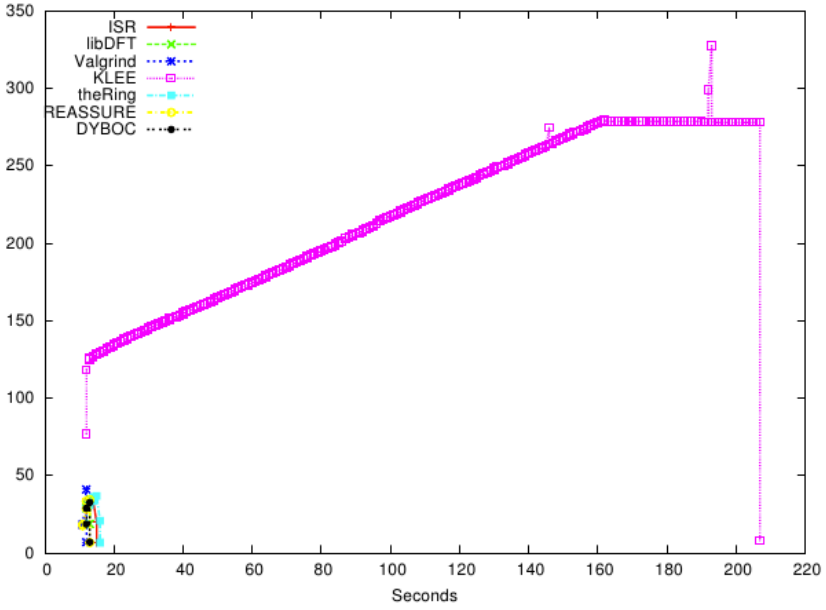
TABLE I. SUMMARY OF SCORING RESULTS ON THE 226 MEMORY CORRUPTION TEST CASES.

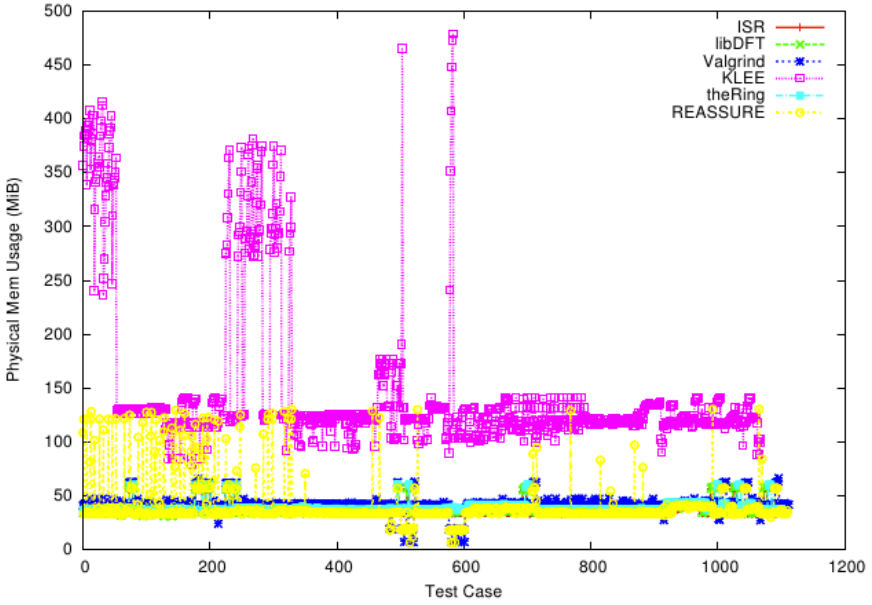
Test and Evaluation: Results

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

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

Copyright © 2011 Symantec Corporation. All rights reserved. Symantec and the Symantec Logo are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.

This document is provided for informational purposes only and is not intended as advertising. All warranties relating to the information in this document, either express or implied, are disclaimed to the maximum extent allowed by law. The information in this document is subject to change without notice.