MINESTRONE: Combining Static and Dynamic Analysis for Software Protection

Angelos D. Keromytis, Junfeng Yang, Sal Stolfo (Columbia)
Angelos Stavrou, Anup Ghosh (GMU)
Dawson Engler (Stanford)
Marc Dacier, Matthew Elder, Darrell Kienzle (Symantec)
MINESTRONE

• Address the problem of security of software of unknown provenance
  – open source or COTS software is brought within an organization
    • might contain intentional vulnerabilities (backdoor or active information leakage)
    • certainly contains unintentional vulnerabilities
  – how do we establish some measure of trust and/or assurance?

http://nsl.cs.columbia.edu/projects/minestrone
Project Focus

• Continuous feedback between dynamic confinement and static analysis techniques to improve vulnerability detection and reduce performance impact of security

• Multi-thrust approach, focusing on legacy applications written in unsafe languages for which source code may be available
  – however, not always desirable/feasible to operate on source code

• Looking at current and future vulnerability classes
  – e.g., problems introduced by increased use of multicore CPUs and parallelism
Runtime Confinement

• Exploring different approaches for introducing an adaptive inline reference monitor
  – binary instrumentation
  – source-code rewriting
  – binary injection
  – lightweight virtualization containers
• Experimental evaluation along performance and effectiveness axis
• Related capabilities: self-healing, leakage detection, multi-core and GPU exploitation
Concurrent Analysis

• Continuous symbolic execution
  – Use dynamic instrumentation to prune/direct state-space exploration

• Static analysis often generates a lot of “noise”
  – Conservatively follow leads by applying dynamic instrumentation (through IRM)
    • Do not rollout unnecessary instrumentation
  – Over time, remove instrumentation deemed “unnecessary”
  – Optimize instrumentation for common case

• Expose information gleaned from source code (e.g., types, information flow) to dynamic confinement component
Software Diversification

• Mitigation mechanism for certain classes of vulnerabilities
  – Code injection (SQL, binary, ROP, ...)
  – Sensor for a posteriori detection of attacks/vulnerabilities

• Key starting technology: Instruction Set Randomization (ISR)
  – On-the-fly creation of diversified runtimes (x86, SQL)
Backup Material
Limitations in state of the art

- Dynamic confinement techniques impose performance and functionality limitations
- Static analysis techniques do not scale much beyond 10,000 LoC
  - improvements basically track Moore’s law
MINESTRONE Architecture

- **Lightweight Containers**
  - Resource Exhaustion Detection
  - ISR + defensive instrumentation
  - REASSURE self-healing

- **ISR + defensive instrumentation**
- **Anomaly Detection**
- **Race Detection**
- **Symbiotes**

- **KLEE prophylactic analysis**

- **Unknown Software**

- **MINESTRONE System Composer**

- **Runtime**
  - Replicated runtime

- **Offline/Parallel**

- **Deployed application (N instances)**

- **Path exploration preference & control flow information**

- **Remove/optimize unneeded defenses**

- **Back end analysis (M << N instances)**

- **KLEE continuous symbolic execution**

- **Information flow tracking optimization**

- **Instrumented replicas (P < N instances)**
Evaluation

• Test against a variety of attacks
  – synthetic
  – hand-crafted
  – real exploits

• Eval scope is unprecedented
  – contribution by itself
Outcomes to date

• Publications (7) and prototypes (3)
  
  • “Practical, low-effort verification of real code using under-constrained execution”
  
  • “Retrofitting Security in COTS Software with Binary Rewriting”
  
  • “Global ISR: Toward a Comprehensive Defense Against Unauthorized Code Execution”
  
  • “Stable Deterministic Multithreading through Schedule Memoization”
  
  • “Bypassing Races in Live Applications with Execution Filters”
  
  • "Fast and Practical Instruction-Set Randomization for Commodity Systems"