Software Exploit Prevention and Remediation via Software Memory Protection

Clark Coleman, Michele Co, Jack Davidson, David Evans, Jason Hiser, John Knight, and Anh Nguyen-Tuong

Department of Computer Science, University of Virginia
Goals

- Provide *push-button* protection of *binaries* from *memory overwriting exploits*
  - Memory overwriting exploit
    - Vulnerability may be unintentional (i.e. a bug) or planted (i.e., insider threat)
  - Binary *only*: No source code; no special hardware
  - Push button: No user intervention
    
    ```
    % protect untrusted_binary
    ```
Goals

- Enable automatic *recovery* and *repair* of an attacked program
  - Recovery: Continued operation in the face of attacks
  - Repair: Diagnose and remove vulnerability (i.e., automatic patching)
Software Memory Protection (SMP)

- Program Code and Data
- SDT-based Profiler
- Static Binary Analyzer
- Offline
- Runtime
- Memory Monitor SDT (mmStrata)
  - Annotations
  - Code and Data

Vulnerable Program Binary
Progress

- **Fine-grained prototype complete**
  - Protects all variables (stack, global, and heap)
  - No false negatives/positives (SPEC, Wilander, SAMATE, etc.)

- **Recovery prototype complete**
  - Successfully recovers from test exploits
  - Adaptive feedback selects recovery policy

- **Performance optimization**
  - Profiler aggressively optimizes instrumentation
  - Adaptive feedback corrects over optimization
Lessons learned

- Static/dynamic approach provides necessary precision

- Adaptive feedback provides flexibility
  - E.g., speculative optimizations
  - E.g., select recovery policy

- Coarse-grained defenses are effective
  - Prevents most known attacks
  - Effort required to create fine-grained attacks is very high
Impact of Success

- NIC can protect binaries including those of untrusted provenance (e.g., insider threat)
- Continuous, autonomous protection
- Provides foundation for additional security monitoring

Come see demo at poster session!