Software Exploit Prevention and Remediation via Software Memory Protection

Clark Coleman, Jack Davidson, David Evans, John Knight, and Anh Nguyen-Tuong

Department of Computer Science, University of Virginia

Presentation Outline

- Introduction and Staffing: Jack Davidson
- Research Overview: Clark Coleman
- Break
- Metrics: Jason Hiser
- Project Schedule, Milestones, and Technology Transfer: Jack Davidson
- Lunch: All
Project Staffing

- **Investigators**
  - Clark Coleman (Research Scientist)
  - Jack Davidson, Dave Evans, John Knight (Faculty)
  - Anh Nguyen-Tuong (Senior Scientist)

- **Staff**
  - Mark Bailey (Visiting Associate Professor)
  - Michele Co (Research Scientist)
  - Jason Hiser (Research Scientist)
  - Dan Williams and Hong Pham (Graduate Students)
  - Nicholas Williams (Undergraduate)

Threat Model

- A program contains memory access vulnerabilities (not all memory operations are guaranteed to stay within proper bounds)
- An attacker can provide malicious input causing a memory overwriting error which compromises the program
The Research Problem

- Protect a running program from memory overwriting exploits, with these requirements:
  - Only a program executable is available (no source code)
  - Provide a general defense, not targeting a subclass of memory overwriting attacks
  - Provide recovery and repair of vulnerable programs
  - Low enough overhead to be usable in real computing environments

- Prior work relaxed one or more of these requirements

Limitations of Prior Work

- Requiring source code (e.g. CCured, Cyclone)
- Requiring custom libraries (e.g. FormatGuard)
- Targeting specific exploits (e.g. StackGuard)
- Defending a program by crashing it rather than allowing recovery and repair (e.g. ASR)
  - Can lead to denial of service (DOS) attack
- Adding so much run time overhead that the solution can only be used while testing (e.g. complete bounds checking or pointer checking)
- Requiring newer hardware (e.g. non-executable data page table bits)
Software Exploit Prevention and Remediation via Software Memory Protection

Software Memory Protection (SMP)

- A general memory overwriting defense, including remediation
- Two components will operate directly on program binaries:
  - A Static Analyzer/Rewriter will analyze memory operations and vulnerabilities off line
  - A Memory Monitor SDT, a virtual execution environment that will watch vulnerable operations at run time and prevent an attacker from compromising the program

Find vulnerable program input locations
Find critical data items that could be overwritten
Record annotations to guide runtime SDT monitoring of input & critical data

Ultra-efficient software dynamic translator analyzes and translates instructions at run time
Static Binary Analyzer annotations guide attack detection and remediation
Static Analyzer/Rewriter

- Disassemble executable into higher level format
- Determine locations of code and data, including function boundaries
- Identify boundaries between data items, including fields within structures, local variables within the stack frame
- Identify pointer writes and pointer arithmetic that are vulnerable to user exploits

Investigating two tool choices: DIABLO and IDA Pro

Comparison: DIABLO and IDA Pro

DIABLO
- Disassembles code and builds control flow graphs
- We have experience using DIABLO to rewrite binaries with encryption to prevent code injection attacks
- Limited targets
- Requires object files
- Open source

IDA Pro
- Disassembler with many robust analyses
- Programmable, plug-in architecture has been extended for security applications by others (e.g. Wisconsin Safety Analyzer)
- 40 targets
- Works on executables
- Commercial product

We will analyze the research, engineering, and technology transfer implications of the choice of tools during July, 2007.
Strata

- Infrastructure designed for building SDTs
- Designed with extensibility in mind
  - Optimization
  - Code compression
  - Performance monitoring
  - Security
- Provides:
  - Platform-independent common services
  - Target interface that abstracts target-specific support functions
  - Target-specific support functions

Software Dynamic Translation

![Diagram of Software Dynamic Translation]
Software Exploit Prevention and Remediation via Software Memory Protection

Performance on Pentium IV (SPEC 2000)

Performance on Pentium IV (Apache)
Performance on Pentium IV (BIND)

General Design

- One basic idea:
  - Via static and dynamic analysis, extract a specification; via run time monitoring, enforce that specification
  - Example: Extract specification of valid memory writes of a program, then have SDT monitor pointer arithmetic, pointer writes, etc. to enforce the specification

- Design is not specific to particular vulnerabilities or targets of attack
Synergy of Static and Dynamic Analysis

- The Static Analyzer/Rewriter runs only when a new executable is selected for protection
  - Analyzer can perform extensive analyses offline
  - The results of these analyses are passed in an annotation file to the Memory Monitor SDT

- Memory Monitor SDT can see actual values at run time that are not visible to a static analyzer
  - Only requires minimal analysis to prevent an exploit from succeeding

Adaptive (Learning) Algorithm

- Run time overhead can be kept minimal during normal execution by not implementing the complete recovery, diagnosis, and repair mechanisms

- Graceful termination after the first attack can be followed by escalating the protection level on subsequent executions
  - Only pay the heavier price during attacks, not all the time
SMP Will Succeed Because:

1. General defense design is not easily circumvented by new exploits
2. Synergy of static and dynamic analysis escapes the limitations of prior efforts
3. Adaptive (learning) algorithm is novel and permits reasonable overhead while still providing extensive remediation capabilities

Impact of Success

- Practical protection for large class of vulnerabilities
  - General techniques for software memory protection
  - Efficient
- Source code not required
  - Legacy apps, COTS, GOTS, 3rd party libraries, outsourced/offshored SW
- Migration path to “safe” software
- Reduces urgency of installing vendor patches
  - Enables administrators to thoroughly test and vet patches
  - No Zero-Day attacks
- Provides NIC with analysis capabilities

Research Impact

- Novel blend of static and dynamic analysis techniques
- Novel application of virtual machine technology
Metrics

- How will we know when success is achieved?
  - Exploit testing
  - False positives/negatives
  - Coverage
    - Executable-only compared to source level information
  - Performance
  - Recovery and remediation success rate

Exploit Testing

- Gather a variety of exploits
  - Problem: Exploits are precariously constructed
    - Simply using SDT or changing one line of source makes the exploit unsuccessful.
  - Solution: Test directly when feasible, otherwise condense to a workable epitome.

- Test that exploits are defeated
  - Expect 100% of memory overwriting exploits to be defeated
  - Expect to defeat many program errors coincidentally
False Positives and False Negatives

- **False Positive**: Detecting an exploit that does not exist
  - **Expected Results**
    - Many as the system starts up and is performing in depth static analysis and adaptive learning
    - Few or none as system progresses

- **False Negative**: Not detecting an exploit
  - **Expected Results**
    - Few in early stages when coarse-grained approach is refined.
    - Very few or none as type inference system refines the granularity of the protection

Coverage

- **How well are we doing compared to source code analysis?**
  - Percent of variables static analysis detects
  - Percent of undetected variables that are vulnerable
  - Percent of variables that are undetected and vulnerable

- **Expected results**
  - State of the art is 88% of variables detected via static analysis. Dynamic+Static Analysis likely to exceed that.
  - We expect exploitable vulnerabilities errors to be in detectable variables.
Performance

- How fast can we provide protection?
  - Compared to native execution
  - Compared to SDT execution without protection

- Expected results
  - Initial version: very high overhead, 100x slower than native execution.
  - Final version: some slowdown, but practical for large class of applications, especially I/O intensive applications such as MS Word.

Recovery and Remediation Success Rate

- Continue execution or repair the application
  - Preliminary Metrics
    - # Attacks until execution can continue
    - # Attacks until repairs can be made
    - Static analysis time to allow remediation
    - Dynamic overhead of system when trying to determine where to make repairs
  - More metrics as recovery and remediation plans solidify
## Milestones and Progress

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prot. 1: Coarse grained Static Data</td>
<td>5/7/2007</td>
<td>9/15/2007</td>
<td>17w</td>
</tr>
<tr>
<td>Prot. 2: Coarse grained Stack &amp; Static Data</td>
<td>9/15/2007</td>
<td>1/18/2008</td>
<td>13w</td>
</tr>
<tr>
<td>Prot. 4: Fine grained Heap</td>
<td>3/31/2008</td>
<td>5/29/2008</td>
<td>5w</td>
</tr>
<tr>
<td>Adaptive learning scheme</td>
<td>5/22/2008</td>
<td>7/4/2008</td>
<td>9w</td>
</tr>
<tr>
<td>Repair and recovery</td>
<td>7/17/2008</td>
<td>9/15/2008</td>
<td>9w</td>
</tr>
<tr>
<td>Evaluation and documentation</td>
<td>9/16/2008</td>
<td>1/17/2008</td>
<td>9w</td>
</tr>
<tr>
<td>Optimization</td>
<td>9/30/2007</td>
<td>5/29/2008</td>
<td>35w</td>
</tr>
<tr>
<td>Improving Granularity</td>
<td>9/30/2007</td>
<td>5/29/2008</td>
<td>35w</td>
</tr>
</tbody>
</table>

### Technology Transfer & Activities

- **UVa Patent Foundation**
  - Licensing of related technology (anti-tampering, execution of static/dynamic encrypted binaries) to financial consulting company

- **Academic**
  - Strata virtual machine in use at other research institutions

- **Industrial**
  - Raytheon (Integrated Defense Systems): experimental evaluation on internal applications

- **National Intelligence Community**
  - Would like to work with NICIAR in identifying potential customers and applications