Dynamic Repair of Applications with Runtime Snap-ins

J. Peter Brady
Dartmouth College, Hanover NH
jpbr@cs.dartmouth.edu
Advisors: Dr. Sean Smith, Dr. Sergey Bratus
The Challenge of Getting Things Right

“Politics is the art of looking for trouble, finding it everywhere, diagnosing it incorrectly and applying the wrong remedies.” – Groucho Marx

(You can say the same about software!)
The Challenges

• Current “best” practice:
  • Ignore, AKA “The Ostrich”
  • Corrective maintenance, AKA “The Patch”

• Traditional patching doesn’t work in the real world
  • Systems that can’t be reached
    • Satellites
    • Mars Rover
    • EDS OT
  • Systems that can’t be down
    • EDS OT systems in a 24x7 operation – what time is good?
  • Obsolete equipment
    • Vendor is out of business
    • So old that no one knows how to maintain it
  • Patching must not change mission-critical behavior
Runtime Snap-ins – Our Approach

• Deliver application changes in real-time
  • Fault repair
  • Functionality updates
  • New features

• Use system features and engineering standards
  • Keep our code base small
  • Make maintenance/enhancements simpler

• **Our secret weapon**: automated ways to locate and modify code only at safe times during execution
Repairing Applications with Runtime Snap-ins

• Repaired code is stored in shared libraries
  • Code objects that can be shared among multiple executables
  • Bound to executables at run time

• Run the *Replacement Constructor Program* (RCP)
  • Creates mapping data of repaired interfaces to apply to existing programs
  • Runs only when a new library is added to system

• The *Snap-in Controller*
  • A separate daemon
  • Searches out target applications
  • Applies the RCP mapping to running programs
Runtime Snap-in Example

• Simple model of the user space memory in a Linux application
  • Text – read-only executable code
  • Data – initialized data (read-only and read-write)
  • BSS – uninitialized data
  • Heap – user allocated memory
  • Shared Libraries
  • Stack – allocated memory for local variables and function parameters
Runtime Snap-in Example

• Application has an external function call *mtxAdd()* which calls a shared library

• *mtxAdd()* has a memory leak we want to fix
Runtime Snap-in Example – Install Library

- The *Snap-in Controller* runs
- The target application is paused by the Controller with *ptrace*
- A small set of dynamic loader instructions are loaded into the heap
- The Controller sets the targets execution pointer to the new instructions and tells the target to run.
- The snap-in library loads into the shared library area. Any number of libraries could be loaded at this point.
- The last instruction on the heap returns to the Controller
Runtime Snap-in Example – Use New Library

• The Controller removes the install code from the heap
• It then rewrites the library pointer to point to the new instance of `mtxAdd()` in the new library by changing offsets in the Global Offset Table
Runtime Snap-in Example – Normal Execution Continues

• The Controller restores the execution pointer of the target, then releases control
• Normal execution is restored on the target application
• Future calls to `mtxAdd()` go to the new version in the snap-in library
Conclusions and Next Steps

• Runtime Snap-in
  • Simple concept allows for powerful updates to running applications
  • Repairs or enhancements are delivered in real-time
  • Future versions will allow more automated search and repair operations

• Operational Timeline
  • Proof-of-Concept
    • Specific set of libraries installed into a specific target application
    • Completed June 2017
  • In-process
    • Search selected applications for routines to be replaced and create mapping to repaired libraries
    • Those system applications can have selected libraries installed automatically
    • Complete in March 2018
  • Future
    • Search for potential failing code in system applications
      • Find known failure signatures in an application and auto-create mapping
    • Rollback one or more repairs
    • Repair code that’s part of the application (i.e.: not in a shared library)
    • Multiple architectures