Preventing Buffer Overflow Attack by Enforcing Control Flow Integrity

Bryan Ta

Project’s Goal

- Goal: implement a tool enforcing CFI to prevent adversaries from taking control of program execution through buffer overflows
- Tool: DynamoRIO
Defense Techniques

- Compile Time Defenses
  - StackGuard and ProPolice: stack canaries
  - Requires source code
- Instruction Set Randomization (ISR)
  - incurs significant overheads unless supported by hardware
- Address Space Layout Randomization (ASLR)
  - No defense against attackers with ability to control the software execution
  - Limited utility on 32-bit architectures
  => Reliance on secret values represents a vulnerability
- In practice, attacks e.g. Shacham et al. (ASLR), Sovarel et al. (ISR - FEEB)

Control Flow Integrity (CFI) policy

- A program’s execution must follow a path of a Control-Flow Graph (CFG) determined ahead of time
  - Effective defense against any attack which attempts to change a program’s control flow
  - Ignore the complexity of various vulnerabilities
CFI Paper’s Implementation

- Martin Abadi, et al. implementation
  - CFG using static binary analysis
  - Dynamic check implemented by static binary rewriting -> modify the binary
  - Requires a binary compiled w/ debug information
  - Overhead : 16% on SPEC2000 benchmarks.

Implementation

- Profiling phase: Builds CFG
  profile normal executions of a program -> determine legitimate control flow transitions ("expected" requests, and no attack is taking place)
  - Static profiling
  - Dynamic profiling
Implementation

- Profiling phase
  - only need to consider “dynamic” control flow, indirect branch instructions
    - no way to change the target addresses of direct branch instructions.
    - e.g. `main() { f(); g();}
  - only consider the returns from `f()` and `g()` to `main()`
  - Do not check the calls to `f()` and `g()`

Implementation

- Enforcement phase
  - Check whether all indirect control flow transferring at runtime match the recorded CFG
    - Each address `X` has a list of legal targets `H(X)={Y1,...,Yn}`
    - at each control transfer at `X`, look up `H[X]` to see whether the jump will be to any of `Y1 ... Yn`.
  - Static binary rewriting:
    - Associate each target address with a tag
    - Addresses in the same `H(x)` have the same tag
    - Insert tags and tag checks into the program
  - False positive (forgot to exercise some part of programs during the profiling phase)
DynamoRIO
Dynamic Binary Instrumentation Tool Platform

- A collaboration between MIT and HP Labs 2001
- supports code transformations while the program executes.
- exports interfaces for building dynamic tools
  - e.g. program analysis and understanding, profiling, instrumentation, optimization, translation, etc.

DynamoRIO

- interpose between the application and the operating system.
- copies the application code one dynamic basic block at a time into its basic block code cache
- runs in the same process and address space with application code.
- Frequently executed sequences of basic blocks are combined into traces for convenient access to hot application code streams.
Future Work

- Implement tagging approach
- More experiments and evaluation