Runtime Defenses

Dawn Song

dawnsong@cs.berkeley.edu

Review

- **Memory-safety vulnerabilities**
  - **Buffer overflow**
  - **Format string vulnerability**
  - **Integer overflow vulnerability**
- **Clarification**

More Memory Safety Vulnerability

- **Double-free**
  - **malloc does not do sufficient checking**
  - **Corrupts free block list**
  - **Write to arbitrary memory location**
  - **"Once upon a free …", Phrack, 11(57), Aug 2001**

Impact of Memory-safety Vulnerabilities

- Modify security-critical data
- Control hijacking: 2 requirements
  - Arrange suitable code to be available in program's address space
    - Code injection
      - Stack
      - Heap
      - Static data area
    - Existing code
      - Libc: E.g. exec(arg)
  - Control flow corruption
    - Return address & base pointer (activation records)
    - Function pointers
    - Longjmp buffers

Defenses & Countermeasures - I

- Type safe languages (Java, ML). DO NOT use C/C++
  - Legacy Code
  - Practical ???
- Secure Coding
  - Avoid risky programming constructs
    - Use fgets instead of gets
    - Use strn* APIs instead of str* APIs
    - Use snprintf instead of sprintf and vsprintf
    - scanf & printf: use format strings
  - Never assume anything about inputs

Defenses & Countermeasures - II

- Mark stack as non-execute.
- Run time checking for memory safety: Purify, array bounds checking (Jones & Kelly).
- Run time overflow detection: Stackguard
- Attack mitigation:
  - Randomization techniques

Marking stack as non-execute

- **Basic stack exploit can be prevented by marking stack segment as non-executable or randomizing stack location.**
  - **Code patches exist for Linux and Solaris.**

- **Problems:**
  - **Does not block more general overflow exploits:**
    - **Overflow on heap: overflow buffer next to func pointer.**
  - **Some apps need executable stack (e.g. LISP interpreters).**

Purify

- **A tool that developers and testers use to find memory leaks and access errors.**
- **Detects the following at the point of occurrence:**
  - **reads or writes to freed memory.**
  - **reads or writes beyond an array boundary.**
  - **reads from uninitialized memory.**

Purify - Catching Array Bounds Violations

- **To catch array bounds violations, Purify allocates a small "red-zone" at the beginning and end of each block returned by malloc.**
- **The bytes in the red-zone → recorded as unallocated.**
- **If a program accesses these bytes, Purify signals an array bounds error.**
- **Problem:**
  - **Does not check things on the stack**
  - **Extremely expensive**

Jones & Kelly: Array Bounds Checking for C

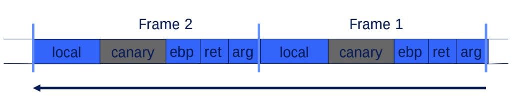
- **A gcc patch that does full array bounds checking**
- **Do not change representation of pointers**
  - **Compiled programs compatible with other gcc modules**
- **Derive a base pointer for each pointer expression, check attributes of that pointer to determine whether the expression is within bounds**
- **High performance overhead**

Administravia

- **Office hour**
  - **If you have any questions or any feedback, pls come by**
- **Background**
  - **Lectures try to be self-contained**
- **Group partner**
- **Project 1**

Run time detection: StackGuard

- **Solution: StackGuard**
  - **Run time tests for stack integrity.**
  - **Embed "canaries" in stack frames and verify their integrity prior to function return.**



Canary Types

- **Random canary:**
  - Choose random string at program startup.
  - Insert canary string into every stack frame.
  - Verify canary before returning from function.
  - To corrupt random canary, attacker must learn the random string.

13

StackGuard (Cont.)

- StackGuard implemented as a GCC patch.
  - Program must be recompiled.
- Low performance effects: 8% for Apache.
- Problem
  - Only protect stack activation record (return address, saved ebp value)

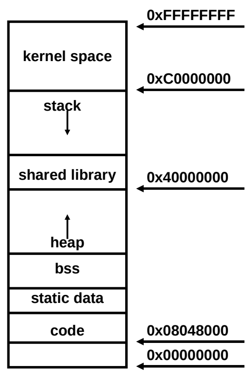
14

Randomization Techniques

- For successful exploit, the attacker needs to know where to jump to, i.e.,
  - Stack layout for stack smashing attacks
  - Heap layout for code injection in heap
  - Shared library entry points for exploits using shared library
- *R*andom*iz*ation Techniques for Software *S*ecurity
  - Randomize system internal details
    - » Memory layout
    - » Internal interfaces
  - Improve software system security
    - » Reduce attacker knowledge of system detail to thwart exploit
    - » Level of indirection as access control

15

Predictable Memory Layout for Linux Application Process



Randomize Memory Layout (I)

- **Randomize stack starting point**
  - **Modify execve() system call in Linux kernel**
  - **Similar techniques apply to randomize heap starting point**
- **Randomize heap starting point**
- **Randomize variable layout**

Randomize Memory Layout (II)

- **Handle a variety of memory safety vulnerabilities**
  - **Buffer overruns**
  - **Format string vulnerabilities**
  - **Integer overflow**
  - **Double free**
- **Simple & Efficient**
  - **Extremely low performance overhead**
- **Problems**
  - **Attacks can still happen**
    - **Overwrite data**
    - **May crash the program**
  - **Attacks may learn the randomization secret**
    - **Format string attacks**