Software Security V: Error Detectors

Chengyu Song

Slides modified from Kostya Serebryan
Administrivia

- Lab1
  - Due: Monday Oct 23 11:59pm
- Talk
  - NOT mandatory
  - Security R&D in a Security Company: Career and Challenges
  - Claud Xiao, Palo Alto Networks
  - Monday, October 23rd, 12:00-1:00pm in Bourns A265
Finding vulnerabilities

- Vulnerabilities are very valuable
- Static analysis vs. dynamic analysis
- Manual vs. automated
- Fuzzing (automated, dynamic)
  - Input generation
  - Feedback
  - Error detection
Monitors

• How to we know if there is bug?
  • Crash?
  • Manually inserted assertions?
  • Error detectors
    • AddressSanitizer (OOB, UAF), ThreadSanitizer (data races),
      MemorySanitizer (UBI), UndefinedBehaviorSanitizer (integer,
      casting), DataFlowSanitizer (information flow), LeakSanitizer
      (memory leak)
Memory error detection

- Memory errors: accessing **undefined** memory
  - Spatial errors: out-of-bound memory
  - Temporal errors: freed memory, uninitialized memory
- Detecting memory errors -> detecting access to undefined memory
- How?
AddressSanitizer (ASAN)

- Source code based instrumentation
  - Inserting redzones around live memory objects
  - Mark freed objects as redzones
  - Inserting checks for accessing redzones
- How to check if an access is valid or not?
  - "magic" number (like canary)
    - shadow memory
Shadow bytes

- Idea: use another memory to store the accessibility of a memory address
- Observation: every aligned 8-byte word of memory have only 9 states
  - First $k$ ([0,8]) bytes are accessible, the rest are not
  - State of every 8-byte word can be encoded in 1 shadow byte
    - 0: all 8 bytes are accessible
    - < 0: inaccessible (different redzone has different negative value)
    - heap redzones, stack redzones, global redzones, freed memory
Locating shadow bytes

\[ \text{Shadow} = (\text{Addr} \gg 3) + \text{Offset} \]
Inserted checks

• An 8-byte memory access

    ShadowAddr = (Addr >> 3) + Offset;
    if (*ShadowAddr != 0)
        ReportAndCrash(Addr);

• 1-, 2-, 4-byte memory access

    ShadowAddr = (Addr >> 3) + Offset;
    k = *ShadowAddr;
    if (k != 0 && ((Addr & 7) + AccessSize > k))
        ReportAndCrash(Addr);
How to insert redzones?

• Heap: replacing \texttt{malloc/free} like functions

• Global objects: insert redzones at compile time
How to insert redzones? (cont.)

- Stack: dynamically at prologue

```c
void foo() {
    char a[10];
    <function body> } ->

void foo() {
    char rz1[32]; // 32-byte aligned
    char arr[10];
    char rz2[32-10+32];
    unsigned *shadow =
        (unsigned*)((long)rz1>>8)+Offset);
    // poison the redzones around arr.
    shadow[0] = 0xffffffff; // rz1
    shadow[1] = 0xffffff0200; // arr and rz2
    shadow[2] = 0xffffffff; // rz2
    <function body> }
```
What about UAF?

1. Mark the freed object as redzone
2. Insert the object into a quarantine queue (FIFO) to prevent reuse
3. When quarantine zone is full, release the memory to the heap allocator
How to re-use memory?

- Heap: mark the allocated part as accessible before return
- Stack
  - Version one (faster): unpoison at epilogue
    - Tricky when exceptions or longjmp are present
  - Version two (slower): unpoison locals at function entry
    - Friendly to exceptions and longjmp
    - Better for use-after-return
ASAN availability

- Both Clang (LLVM) and GCC have support
- OS: Linux, OS X, iOS simulator, FreeBSD, Android
- ISA: x86, x86_64, ARM, ARM64, MIPS, MIPS64, PowerPC64
Example

% cat tests/use-after-free.c
#include <stdlib.h>
int main() {
    char *x = (char*)malloc(10 * sizeof(char*));
    free(x);
    return x[5];
}
% ../clang_build_Linux/Release+Asserts/bin/clang -fsanitize=address -O1 -fno-omit-frame-pointer -g tests/use-after-free.c
Example report

% ./a.out
==9901==ERROR: AddressSanitizer: heap-use-after-free on address 0x60700000dfb5 at pc 0x45
READ of size 1 at 0x60700000dfb5 thread T0
  #0 0x45917a in main use-after-free.c:5
  #1 0x7f8f25e76c in __libc_start_main /build/buildd/eglibc-2.15/csu/libc-start.c:226
  #2 0x459074 in _start (a.out+0x459074)
0x60700000dfb5 is located 5 bytes inside of 80-byte region [0x60700000dfb0,0x60700000e000]
freed by thread T0 here:
  #0 0x4441ee in __interceptor_free projects/compiler-rt/lib/asan/asan_malloc_linux.cc:
  #1 0x45914a in main use-after-free.c:4
  #2 0x7f8f25e76c in __libc_start_main /build/buildd/eglibc-2.15/csu/libc-start.c:226
previously allocated by thread T0 here:
  #0 0x44436e in __interceptor_malloc projects/compiler-rt/lib/asan/asan_malloc_linux.cc:
  #1 0x45913f in main use-after-free.c:3
  #2 0x7f8f25e76c in __libc_start_main /build/buildd/eglibc-2.15/csu/libc-start.c:226
SUMMARY: AddressSanitizer: heap-use-after-free use-after-free.c:5 main
Effectiveness

• Found hundreds bugs since May 2011
  • Chromium (WebKit, ffmpeg)
  • Server-side apps
  • Even one in clang

• AFL + ASAN -> default setup for many people
Weaknesses?

- To detect all spatial errors, redzone based memory error detector requires the redzone size to be infinite
  - ASAN: 32 bytes
- To detect all UAF errors, redzone based memory error detector should never reuse freed memory
  - ASAN: quarantine zone
- Trade-off: memory overhead vs. false negatives
- Sub-fields
Other memory error detector?

- Windows: PageHeap
- KASAN
- MEDS, OSCAR
For next class ...

• Software Security VI: symbolic execution