Dynamic Binary Instrumentation

August 2016

Francis Giraldeau
francis.giraldeau@polymtl.ca

Prof. Michel Dagenais
michel.dagenais@polymtl.ca

Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL

ERICSSON

Principle of operation

- Library overloading
- Modify assembly code to insert instrumentation
- Offline: rewrite the executable file
- Online: self modifying code
  - Paused: i.e. ptrace()
  - Non-stop (kernel)
    - Code update while executing!
    - tracepoints (static_key)
    - FTrace

Library overloading

- Compile a library with the symbols to overload
- Preload the library with LD_PRELOAD
- The overloaded function will be called instead of the original library function. You may call the real library function.
- Does not work with static executables

Exemple: overloading malloc()

Function pointer for the real malloc

```
void *(*malloc_real)(size_t size);

void *malloc(size_t size)
{
    void *ptr;
    if (malloc_real == NULL) {
        malloc_real = dlsym(RTLD_NEXT, "malloc");
    }
    ptr = malloc_real(size);
    const char *msg = "malloc() called";
    write(1, msg, strlen(msg));
    return ptr;
}
```

Redefine malloc()

Load the real malloc if not already loaded

Custom instrumentation. Beware: you must not call malloc() here, otherwise an infinite recursion will occur! Call malloc_real() instead.

Dyninst

- API to inspect and modify assembly code
- Online with ptrace() or offline mode
- Can inject generated code
- Web site: https://github.com/dyninst/dyninst
- Example: dyninst-demo

Linux tracepoint

- When not enabled: nop
- Enabling changes the code to a jump
- Based on gcc asm goto
  - Requires a constant key address
  - False branch code is not optimized out
- Meta-info about instrumented sites in section __jump_table
- See example 31-asm-goto

Linux FTrace

- Compilation with -pg (man gcc)
- All function prologue contains call to mcount
  - The space is 5 bytes
- Call to mcount is patched to nop
- The nop is patched to jump to instrumentation

Runtime patching on x86

- Processor must see valid instructions at all time!
- Patch the first instruction to 0xCC
  - Call to Interrupt Procedure
  - Interrupt handler jumps to the next instruction
- Synchronize cores
- Change the last 4 bytes
- Synchronize cores
- Change the first byte
- Synchronize cores
- New code is effective!

Useful resources

- http://ref.x86asm.net/coder64.html
- http://www.felixcloutier.com/x86/
- Tools: objdump, nm, readelf
- See linux/arch/x86/kernel/jump_label.c