Executable Hardening Measures

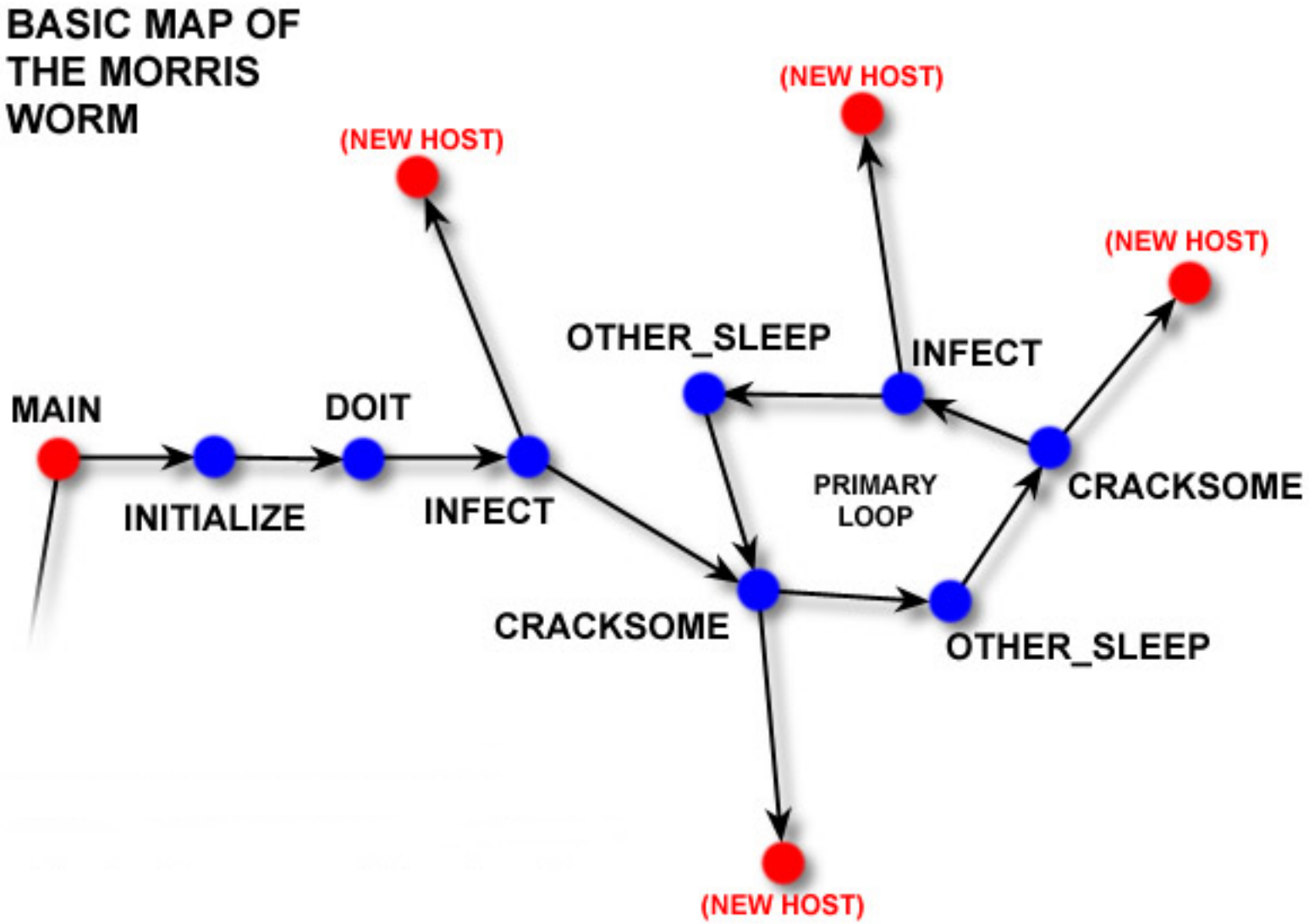
(How they work or don't)

Steve Grubb
Red Hat

Protection Mechanisms.

- NX
- ASLR
- PIE
- Stack Protector
- FORTIFY_SOURCE
- RELRO

Morris Worm - 1988



Smashing the Stack for Fun and Profit

Aleph One

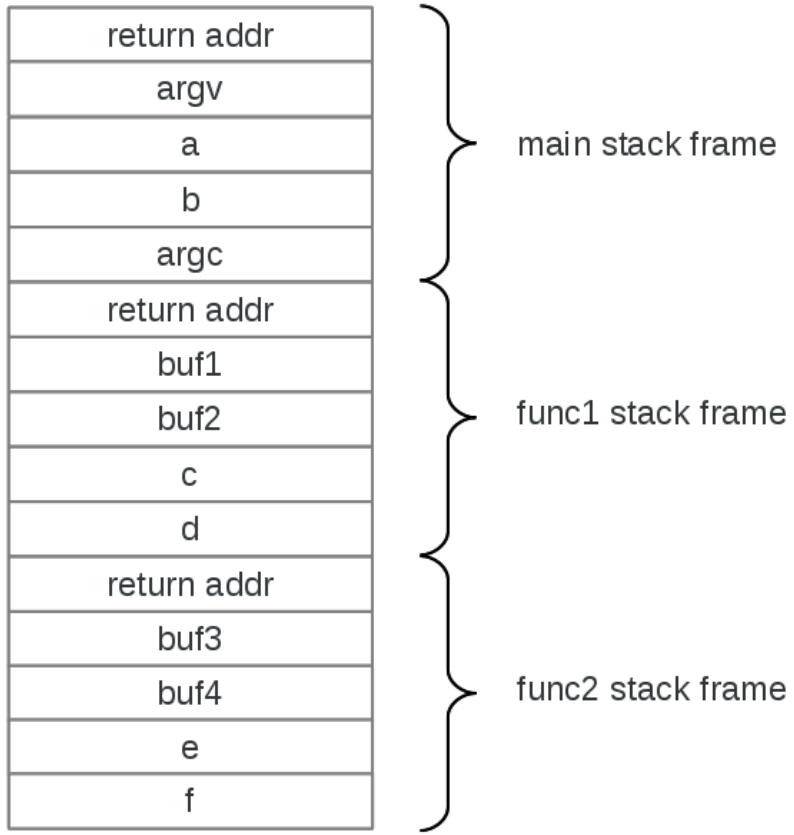
- Phrack #49 August 1996
- Demonstrated how to turn stack overflow into exploit:
  - Copy shell code to stack
  - Modify return address
  - Finish function & return
  - Captured!

Stacks

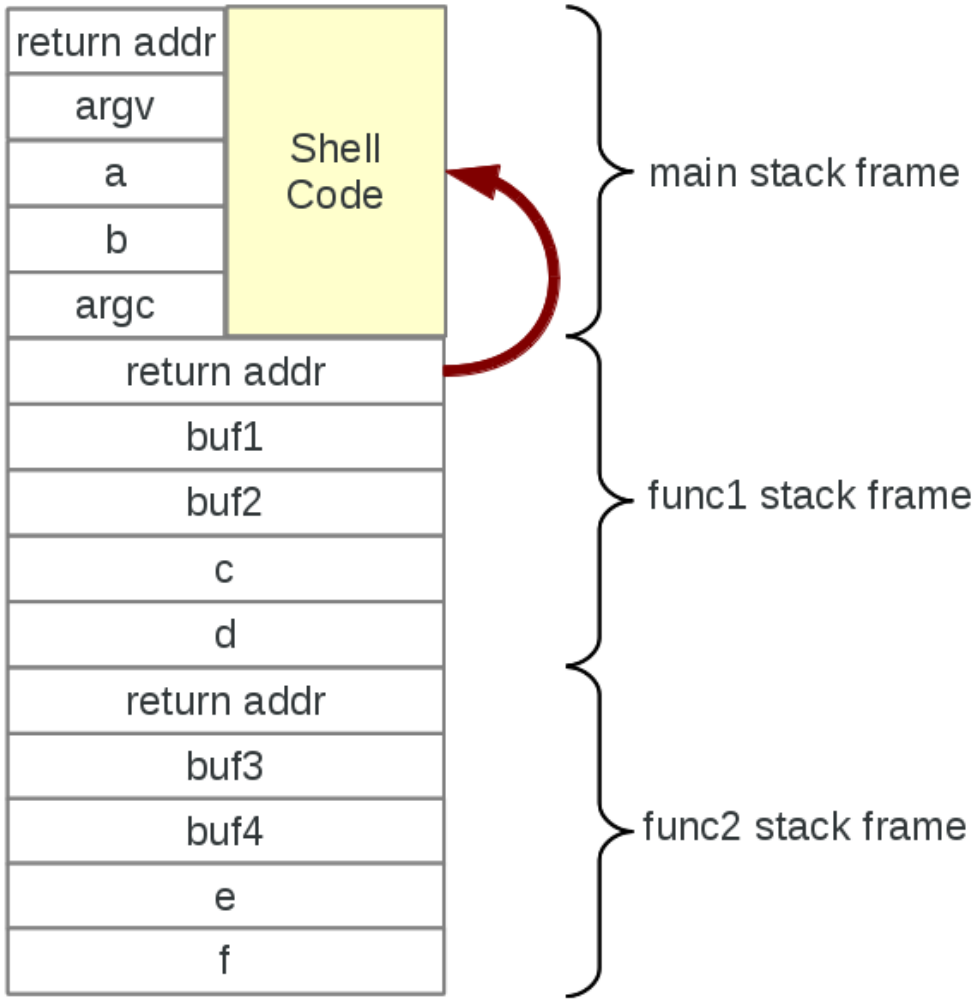
- Computer Science Abstraction
  - Used for temporary storage
  - LIFO – Last in, first off
  - PUSH, POP
- On Intel Processors it grows down
  - Conflicts with the way we think of memory

Sample Stack Layout

```
void func2(int e, int f)
{
        char buf3[8];
        char buf4[8];
}

void func1(int c, int d)
{
        char buf1[8];
        char buf2[8];

        func2(c, d);
}

int main(int argc, char *argv[])
{
        int a=1, b=2;

        func1(a, b);
        return 0;
}
```



Stack Exploit



NX.

- Prevent execution on stack addresses
- Available per page rather than whole segment
- Can be emulated in the kernel
- Gcc support
  - Compile time creates gnu-stack.notes
  - Linker sets ELF flags
  - Kernel interprets flags and marks stack R/W
- eu-readelf ./test | grep STACK

```
GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RW  0x8
```

Still get executable stack when...

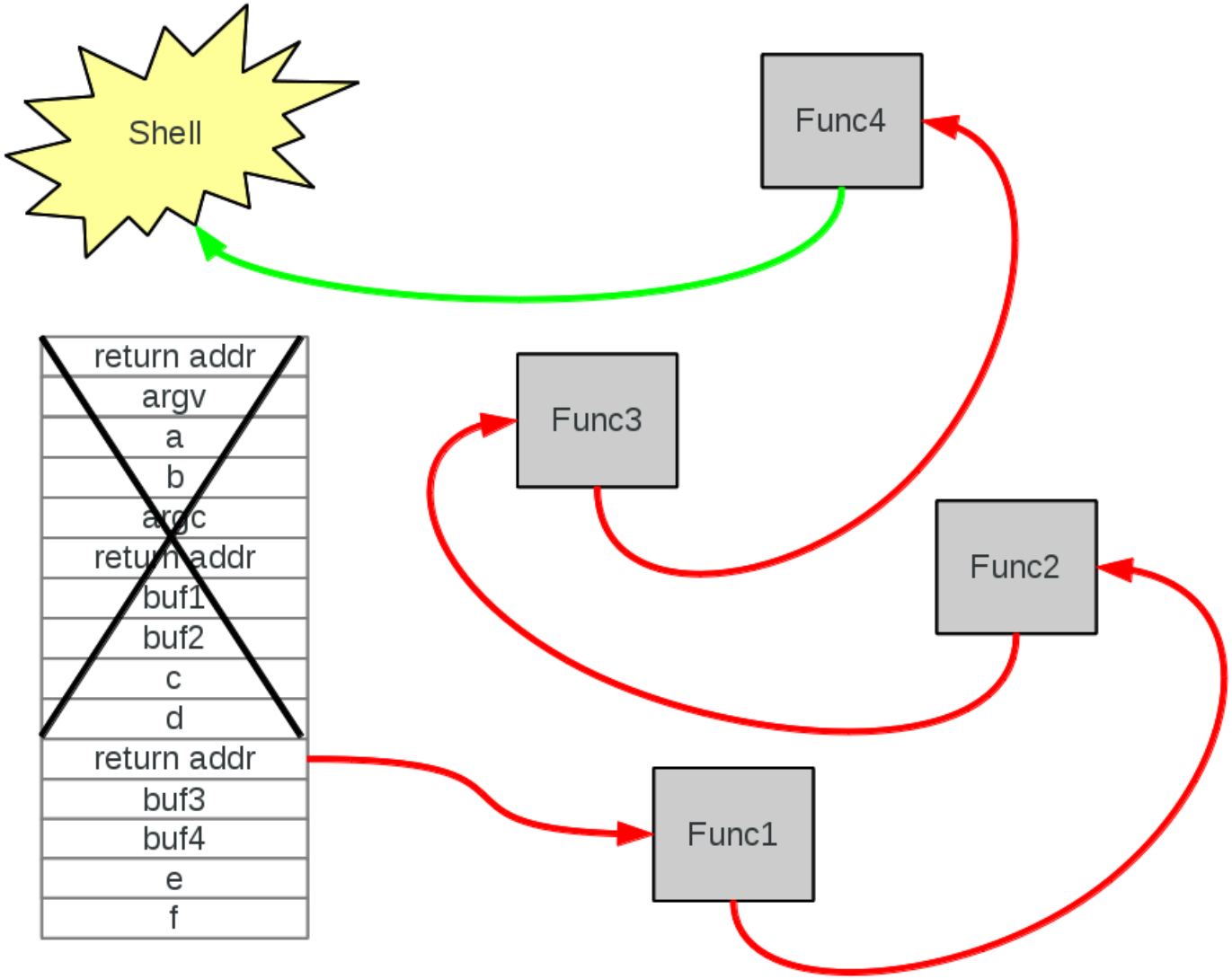
```
int main(int ac, char **av)
{
     int localfn(int a) {
         return a+ac;
     }
     int (*fptr)(int) = localfn;

     printf("%d\n", fptr(-1));
     return 0;
}
```

```
$ eu-readelf -l ./test | grep STACK
  GNU_STACK      0x000000 0x0000000000000000 0x0000000000000000 0x000000 0x000000 RWE 0x8
```

ROP

- With NX, shell code has to go somewhere else
  - Heap
  - Return into libc
  - Return Oriented Programming
    - Stack is loosely coupled with function using it
    - Each function does some work before returning
    - Stack can be arranged to return to other functions to string together shell code or exploit

ROP


Shell
Func4
Func3
Func2
Func1
return addr
argv
a
b
argc
return addr
buf1
buf2
c
d
return addr
buf3
buf4
e
f

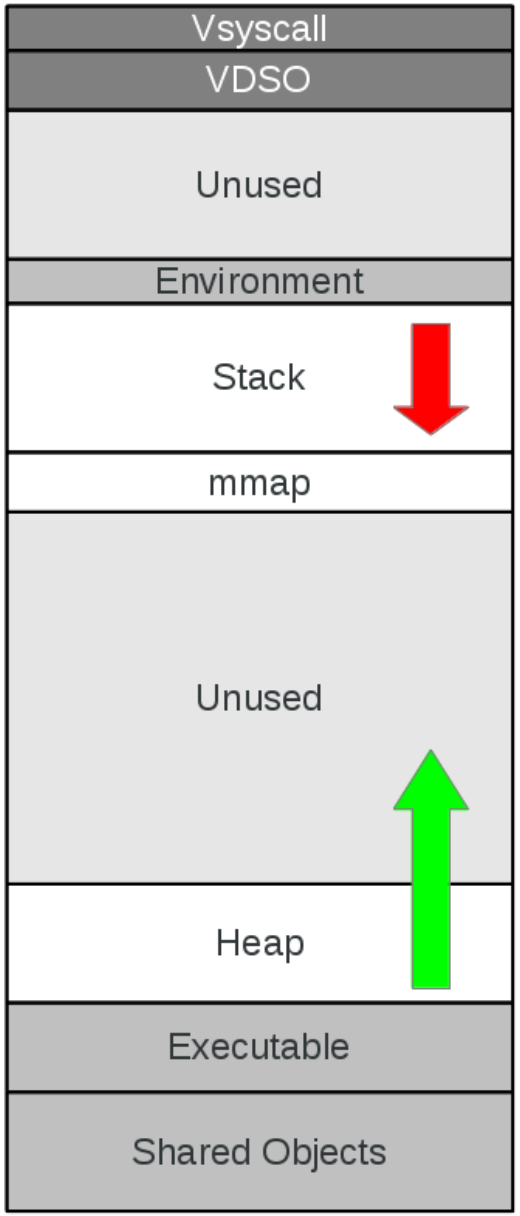
ASLR

- Vary memory locations on each run
  - Stack
  - Heap
  - Mmap
  - Executable
  - Shared Object entry points
    - Prelink affects this

ASLR – default layout

```
int main(int argc, char *argv[], char *envp[])
{
        int pid = getpid();
        char *cmd = malloc(50);
        char *ptr = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
                    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0);

        printf("%018p  stack\n", &pid);
        printf("%018p  env\n", getenv("SHELL"));
        printf("%018p  exec\n", main);
        printf("%018p  heap\n", cmd);
        printf("%018p  mmap\n", ptr);
        printf("%018p  so\n", snprintf);

        return 0;
}
```

```
$ ./layout | sort -r
0x00007fff1700c489  env
0x00007fff1700bd6c  stack
0x00007fdb10824000  mmap
0x0000000002520010  heap
0x00000000004006ec  exec
0x00000000004005b0  so
```


Vsyscall
VDSO
Unused
Environment
Stack
mmap
Unused
Heap
Executable
Shared Objects

ASLR – randomness

```
$ ./all-bits
heap    14 bits
exec    No randomization
mmap    29 bits
so      No randomization
stack   28 bits


$ ./all-mask
heap     0x0000000003FFF000
exec     0x0000000000000000
mmap     0x000001FFFFFFF000
so       0x0000000000000000
stack    0x00000000FFFFFFF0
```

ASLR – Bias?

```
$ cat heap.log | ./summary | head
0x0000000000bb7010                      20
0x00000000161c010                       19
0x0000000000dac010                      19
0x0000000001a28010                      18
0x00000000023f8010                      18
0x0000000000f3c010                      18
0x0000000001f1b010                      18
0x0000000000f5a010                      18
0x00000000011bc010                      18
0x0000000000976010                      17

$ cat heap.log | ./summary | tail
0x0000000000cc2010                      1
0x00000000016ae010                      1
0x00000000017b2010                      1
0x00000000018be010                      1
0x00000000015c6010                      1
0x0000000001e60010                      1
0x0000000000e7f010                      1
0x00000000001148010                     1
0x0000000001d6e010                      1
0x0000000001bfe010                      1
```

ASLR – Bias?

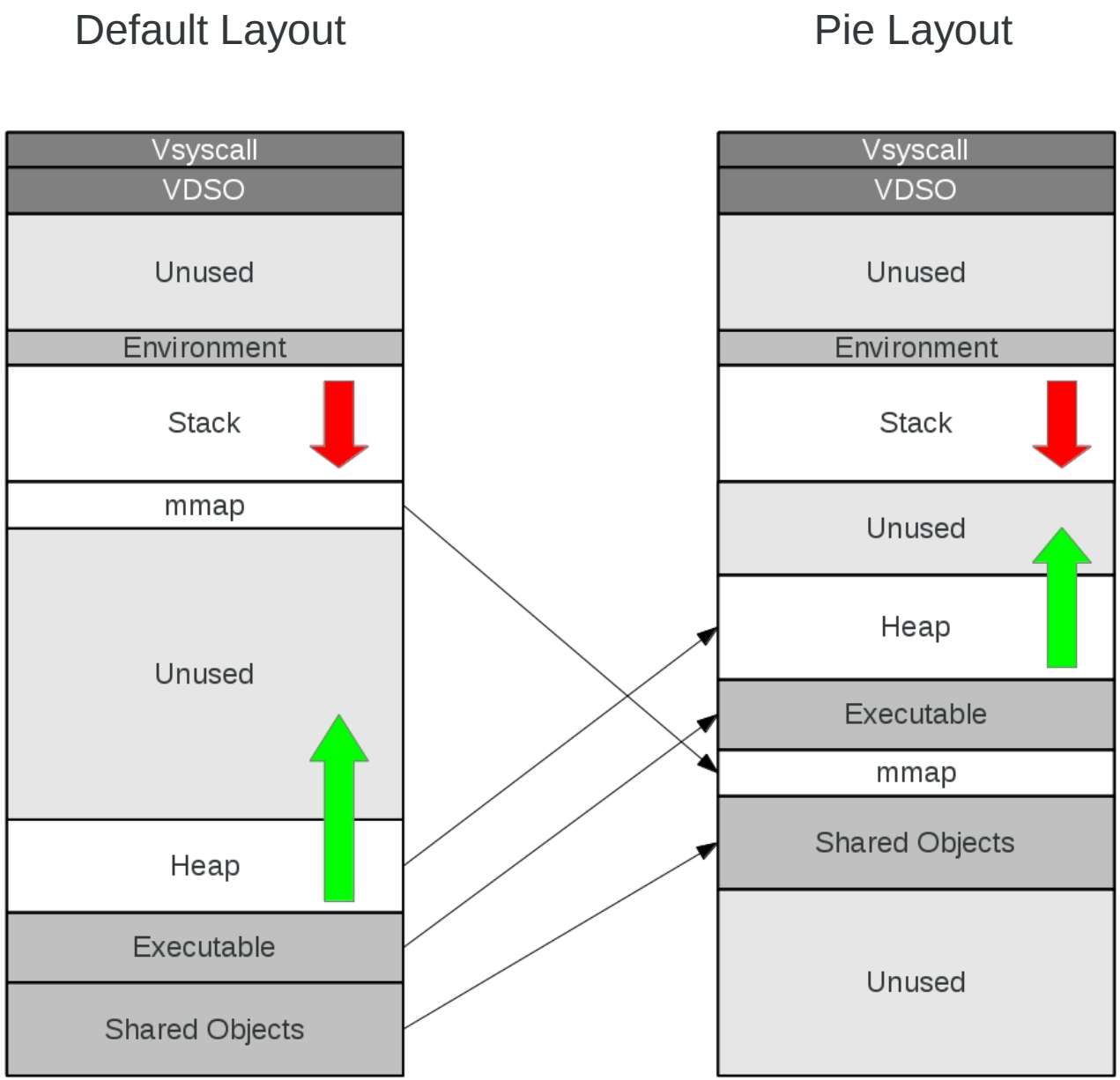
Distribution	Total
1	21
2	87
3	229
4	442
5	760
6	1009
7	1155
8	1160
9	992
10	820
11	635
12	379
13	245
14	133
15	63
16	36
17	16
18	6
19	2
20	1

PIE

- Position Independent Execution
  - CFLAGS+=”-DPIE -fPIE”
  - LDFLAGS+=”-pie”
  - Slower startup
  - Introduces a new writable memory segment
    - Must be fixed by using RELRO
- Suggested Use
  - Daemons, Setuid, Network Facing, Parsing untrusted media

PIE – ASLR Layout

```
./layout-pie | sort -r
0x00007ffff73f9481  env
0x00007ffff73f8a2c  stack
0x00007f91a20a2010  heap
0x00007f91a14c3a00  exec
0x00007f91a14be000  mmap
0x00007f91a0f38e40  so
```


Default Layout
Pie Layout
Vsyscall
VDSO
Unused
Environment
Stack
mmap
Unused
Heap
Executable
Shared Objects
Vsyscall
VDSO
Unused
Environment
Stack
Unused
Heap
Executable
mmap
Shared Objects
Unused

PIE – ASLR randomness

```
$ ./all-bits
heap        29 bits
exec        29 bits
mmap        29 bits
So          29 bits
stack       28 bits
```

```
$ ./all-mask
heap        0x000001FFFFFFF000
exec        0x000001FFFFFFF000
mmap        0x000001FFFFFFF000
so          0x000001FFFFFFF000
stack       0x00000000FFFFFFF0
```

PIE – ASLR Bias?

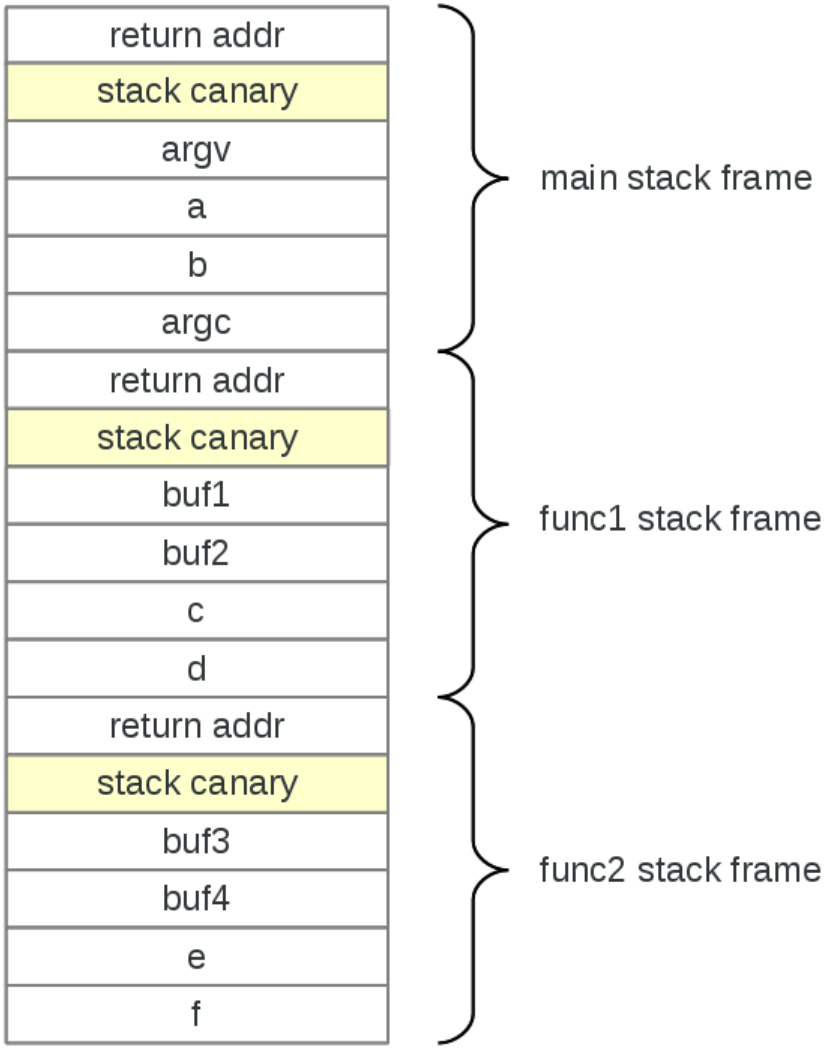
```
$ cat pie-heap.log | ./summary | head
0x00007f6ee3e18010                          2
0x00007f142e372010                          2
0x00007f39702f3010                          1
0x00007f3bbfa77010                          1
0x00007fed8e7be010                          1
0x00007fedd3ca6010                          1
0x00007fe3c812f010                          1
0x00007fc1d4a1f010                          1
0x00007f3847caf010                          1
0x00007fa6a7cf5010                          1
```

Conclusion: Fix non-PIE heap to be the same!

Stack Protector

- ASLR alone is not enough – exec & DSO
- Gcc has built in stack protector
  - Add random number into call frame
  - Check on return
  - Abort program if its altered
  - Re-arrange the layout so less things get damaged on overflow
- Using it
  - -fstack-protector or -fstack-protector-all
  - --param=ssp-buffer-size=4

Stack Protector



Stack Protector.

```
.LVL2:
        .loc 1 8 0
        movl    $7959911, 32(%rsp)
        .loc 1 10 0
        call    printf
.LVL3:
        .loc 1 11 0
        movq    56(%rsp), %rax
        xorq    %fs:40, %rax
        jne     .L5
        addq    $72, %rsp
        .cfi_remember_state
        .cfi_def_cfa_offset 8
        ret
.L5:
        .cfi_restore_state
        .p2align 4,,6
        call    __stack_chk_fail
.LVL4:
        .cfi_endproc
```

Stack Protector Gotcha's

- alloca(3) not covered unless optimizations are used
- Many cases are NOT covered
  - Wide characters
  - Char arrays in structures / unions
  - Integer arrays
  - Access via pointer arithmetic

Wide characters

```
#include <stdio.h>
#include <stdlib.h>

void *memset(void *dest, int c, size_t n);

int main(void)
{
    wchar_t buf1[8];

    memset(buf1, 'a', 0x60);
    printf("buf1:%p, %s.\n", &buf1[0], buf1);

    return 0;
}
```

Passing address of array

```
#include <stdio.h>
#include <string.h>

// This is on-purpose to isolate the problem to
//the stack protector
void *memset(void *dest, int c, size_t n);

void test2(char **buf1)
{ // Now pretend something corrupts the stack
  memset(buf1, 'a', 40);
}

void test1(void)
{
  char *buf = strdup("test");;
  test2(&buf);
}

int main(void)
{
  test1();
  return 0;
}
```

CVE-2013-0288 nss-pam-ldapd

```
#include <stdio.h>
#include <sys/select.h>

// Simulates CVE-2013-0288 nss-pam-ldapd: FD_SET array index
// error, leading to stack-based buffer overflow

void my_select(int fd)
{
        struct timeval tv;
        fd_set fdset;

        FD_ZERO(&fdset);
        FD_SET(fd,&fdset);
        if (!fd)
                select(FD_SETSIZE,&fdset,NULL,NULL,&tv);
}

int main(void)
{
        int i = 0;
        while(i<1500) {
                i++;
                my_select(i);
        }
        return 0;
}
```

Measuring Coverage

- Eu-readelf can see if any calls to __stack_chk_fail()
- Disassembler?
- Build logs?
  - Add -fdump-rtl-expand compile flag
  - Produces *.expand files
  - Call tree can be extracted
  - Analysis and visualization can be done

Coverage – dir listing

```
unbound-checkconf.ltrans14.150r.expand  unbound.ltrans21.150r.expand
unbound-checkconf.ltrans15.150r.expand  unbound.ltrans2.150r.expand
unbound-checkconf.ltrans16.150r.expand  unbound.ltrans22.150r.expand
unbound-checkconf.ltrans17.150r.expand  unbound.ltrans23.150r.expand
unbound-checkconf.ltrans18.150r.expand  unbound.ltrans24.150r.expand
unbound-checkconf.ltrans19.150r.expand  unbound.ltrans25.150r.expand
unbound-checkconf.ltrans20.150r.expand  unbound.ltrans26.150r.expand
unbound-checkconf.ltrans21.150r.expand  unbound.ltrans27.150r.expand
unbound-checkconf.ltrans2.150r.expand   unbound.ltrans28.150r.expand
unbound-checkconf.ltrans22.150r.expand  unbound.ltrans29.150r.expand
```

Coverage

```
cat $f | awk '  /;; Function/ { printf "Function: %s\n", $3 }
                $6 == "(call" { printf " calls: %s\n", $9}
                $1 =="(call"  { printf " calls: %s\n", $4}'
```

```
Function: setup_if
 calls: ("memdup")
 calls: ("ipstrtoaddr")
 calls: ("calloc")
Function: delegpt_log
 calls: ("dname_str")
 calls: ("log_info")
 calls: ("delegpt_count_ns")
 calls: ("delegpt_count_addr")
 calls: ("log_info")
 calls: ("dname_str")
 calls: ("log_info")
 calls: ("log_addr")
 calls: ("__stack_chk_fail")
```

Sample results

- Openssh-6.0p1
  903 functions, 134 protected, 14.8%
  10005 function calls

- Unbound-1.4.13
  5817 functions, 225 protected, 3.87%
  45441 function calls

-fstack-protector-strong

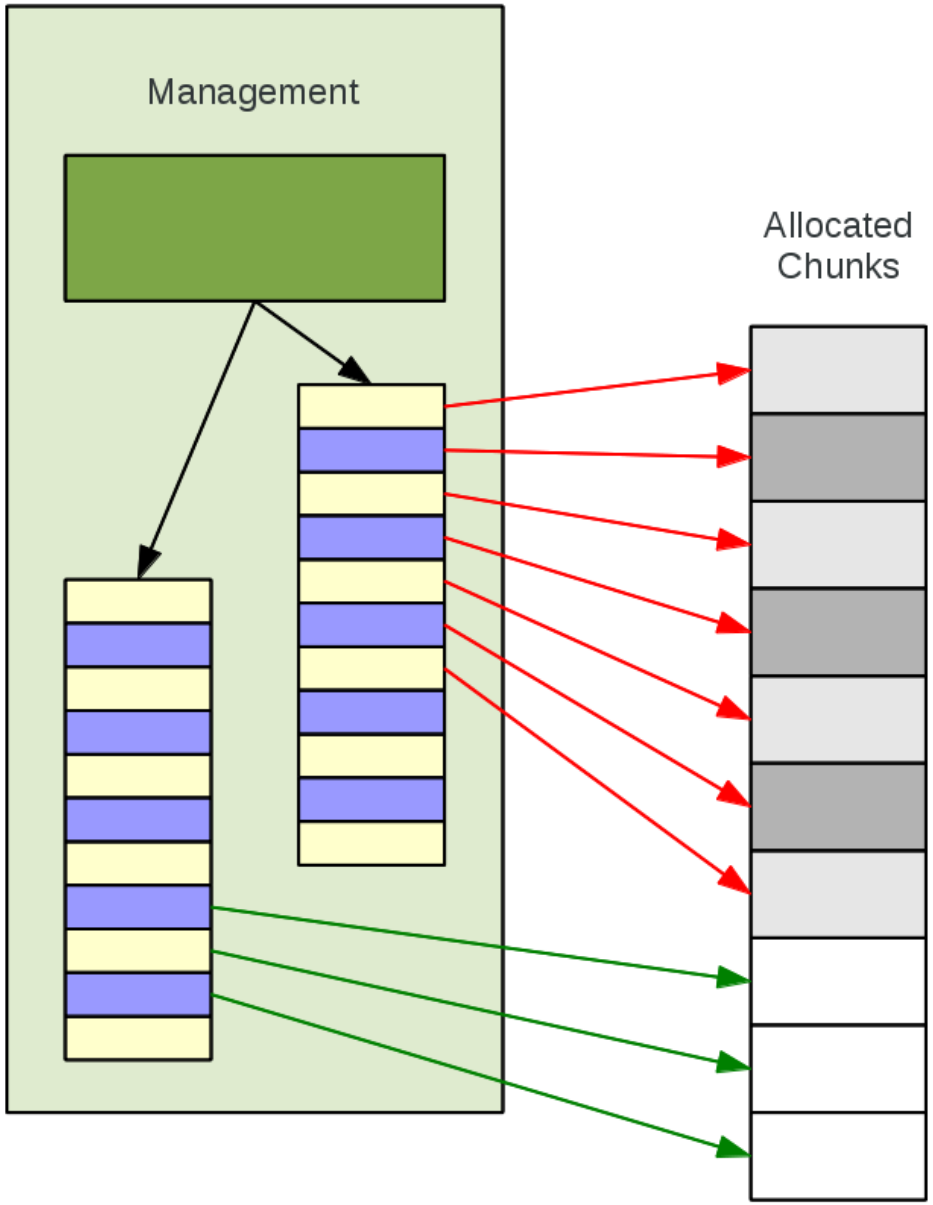
- Similar to -stack-protector but includes:
  - Local array definitions
  - Passing references of local frame addresses
- Not in gcc – patch is available

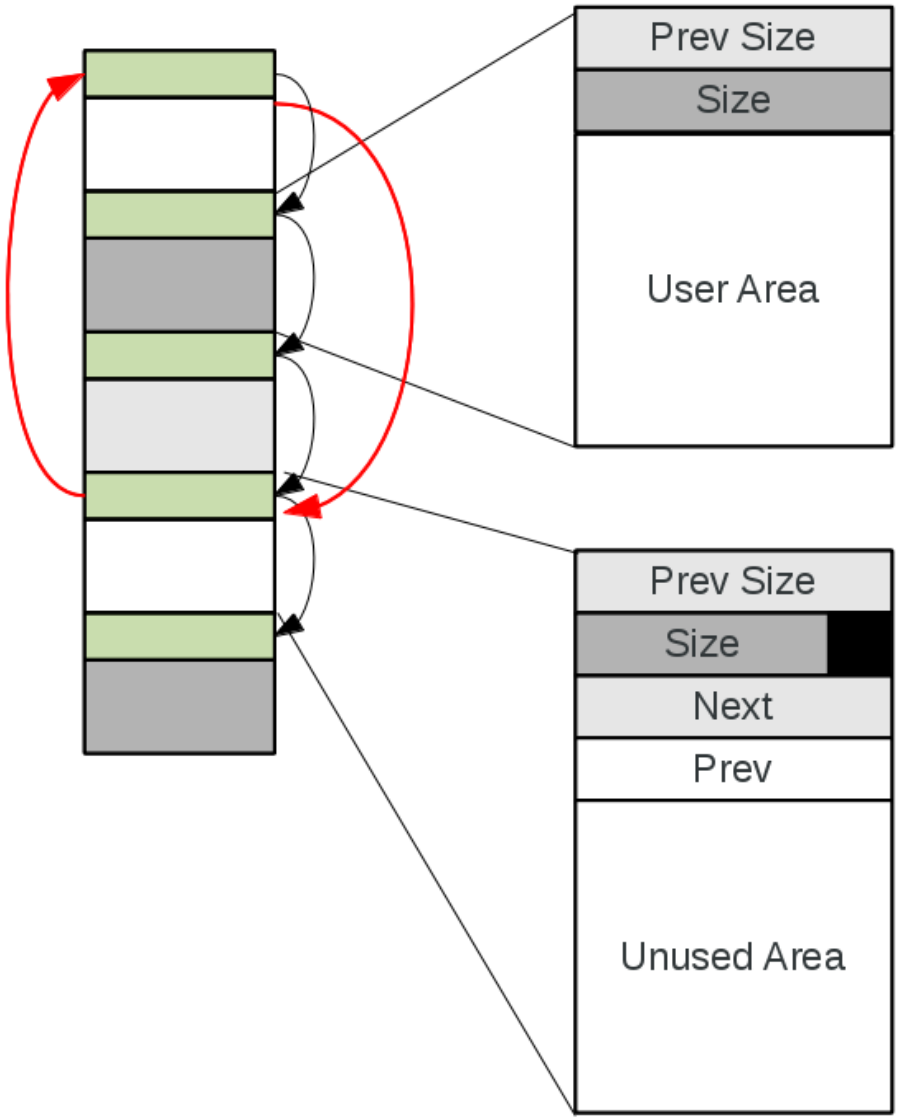
Stack Protector Comparison

	plain	strong	all
openssh	15%	34%	100%
openssl	4%	14%	100%
gnutls	11%	36%	100%
bind	14%	41%	100%
qemu	5%	19%	100%

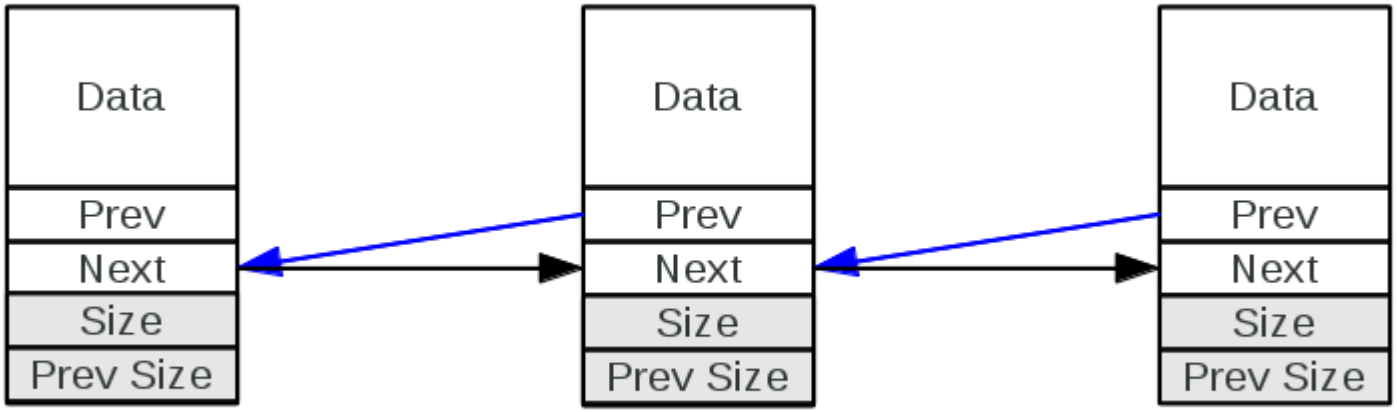
Heap

- Data allocated via malloc(3) or realloc(3)
- From attack perspective
  - Function pointers
  - Contexts
  - Access rights
  - Crypto secrets

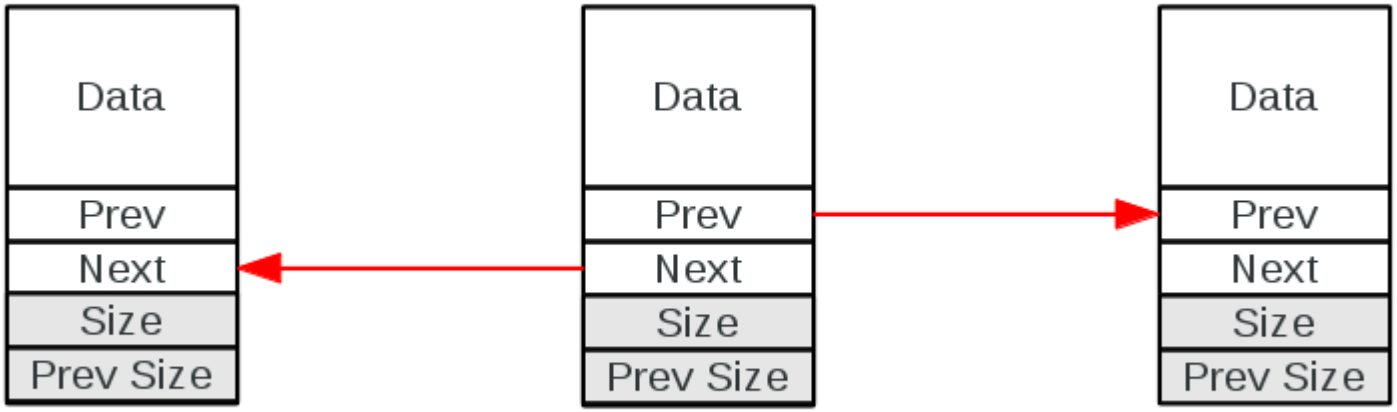

Management
Allocated
Chunks


Prev Size
Size
User Area
Prev Size
Size
Next
Prev
Unused Area

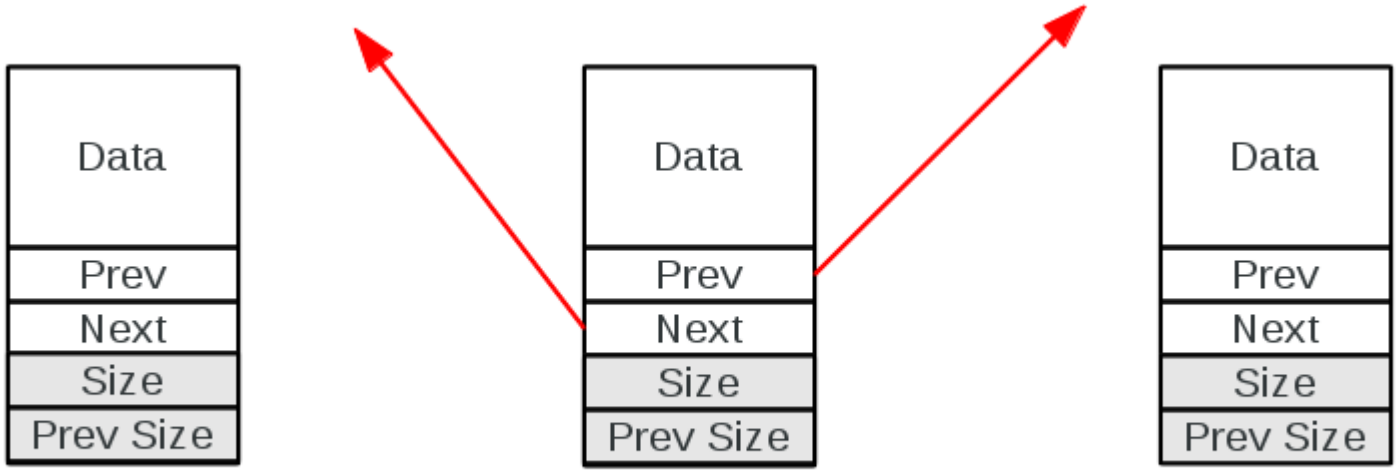
Normal empty list



Unlinking during free



Unlinking during use after free



Unlinking during free

```
if (n->prev)
    n->prev->next = n->next;
if (n->next)
    n->next->prev = n->prev;
```

The fix

```
if (n->next->prev == n && n->prev->next == n) {
    if (n->prev)
        n->prev->next = n->next;
    if (n->next)
        n->next->prev = n->prev;
} else {
    backtrace(...);
}
```

Glib2

- Convenience Library
  - Has doubly linked list algorithms
  - It's possibly vulnerable!
    - g_list_remove_link()

Jemalloc

- Claims to be faster than glibc
- Include its headers, no re-coding
- Management not near chunks
  - Harder to attack state
- Allocations are adjacent
- Does not detect or enforce integrity between chunks

Jemalloc Properties

```
$ ./all-bits
heap        19 bits
pie-heap    19 bits

$ ./all-mask
heap        0x000001FFFFC00000
pie-heap    0x000001FFFFC00000



$ cat heap.log | ./summary | head
0x00007f7ab300e040                      5
0x00007fe56480e040                      5
0x00007f152b80e040                      5
0x00007f917c00e040                      4
0x00007fba9780e040                      4
0x00007f386680e040                      4
0x00007f180340e040                      4
```

To free or not to free?

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        FILE *f;
        char *ptr = malloc(MAX_PATH);
        snprintf(ptr, MAX_PATH, "/proc/%s", argv[1]);
        f = fopen(ptr, "r");

        do_some_work(f);

        return 0;
}
```

Moral: use valgrind!

FORTIFY_SOURCE

- Sometimes sizes are known at compile time
  - Problems can be stopped early
  - Helps enforce heap integrity
- Requires a -O flag of some kind
- Enabled by default in Fedora
- In CFLAGS -D_FORTIFY_SOURCE=2


Learn
to
Fly

FORTIFY_SOURCE Gotcha's

- If define given and -O not given, older glibc quietly ignores FORTIFY_SOURCE
- If header not included, no protection
- If a buffer is in one function and its overrun in another, fortify misses it
- If a buffer is in one file and its overrun in another (or in a library), fortify misses it
- Gnulib probably does not have FORTIFY support

FORTIFY_SOURCE

```
#include <stdio.h>
#include <stdlib.h>
//#include <string.h>

void *memcpy(void *dest, const void *src, size_t n);

int main(void)
{
   char *ptr = malloc(5);
   memcpy(ptr, "a", 24);
   free(ptr);

   return 0;
}
```

```
#include <stdio.h>
#include <string.h>

void test2(char *buf)
{
 memset(buf, 'a', 80);
 buf[80] = 0;
 printf("buf1:%s\n", buf);
}

void test1(void)
{
  char buf1[5];

  sprintf(buf1, " ");
  test2(buf1);
}

int main(void)
{
  test1();
  return 0;
}
```

RELRO

- Using PIE causes indirection
  - Must be writable lookup table
- Make RELocations ReadOnly
  - Fixes weakness introduced by PIE
  - Re-arranges the elf sections so less likely to get corrupted
  - -Wl,-z,relro - lazy bindings
  - -Wl,-z,now - resolve all bindings at startup
    - Slows startup – but best for secure apps

Suggested Policy

- Daemon, setuid, network facing, parser of untrusted data
  - PIE, FULL RELRO, FORTIFY_SOURCE, -fstack-protector-strong, no executable stack, no executable memory
- Other apps
  - Partial RELRO, FORTIFY_SOURCE, -fstack-protector, no executable stacks, no executable memory

Questions?

sgrubb@redhat.com
http://people.redhat.com/sgrubb/swa