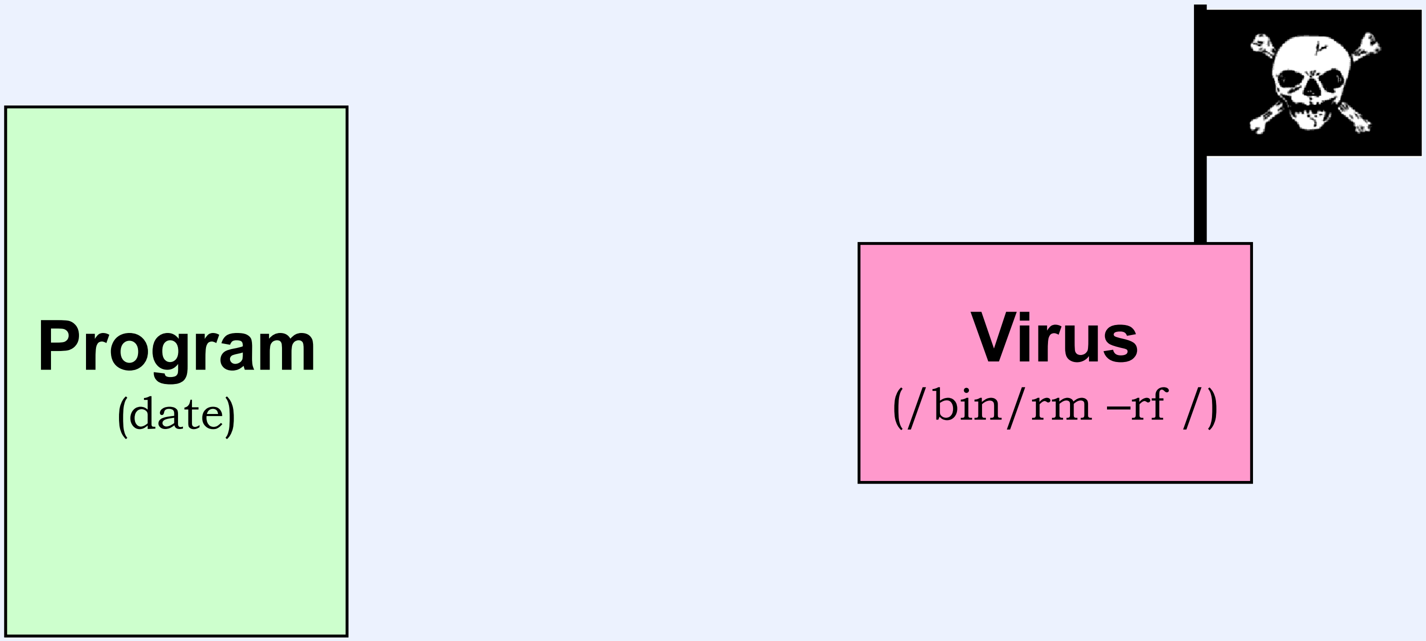
Security Part 2

Live Anonymous Q&A:
https://tinyurl.com/cs1670feedback

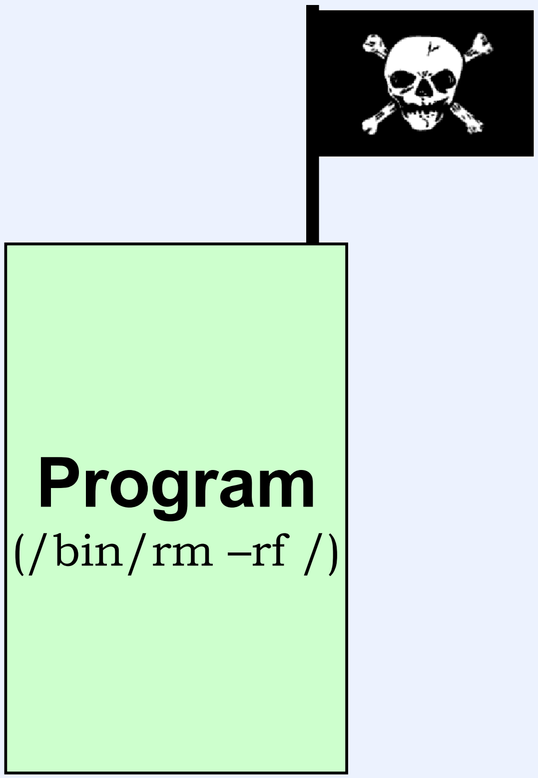
Viruses and Worms

- Virus: an “infection” of a program that replicates itself
- Worm: a standalone program that actively replicates itself

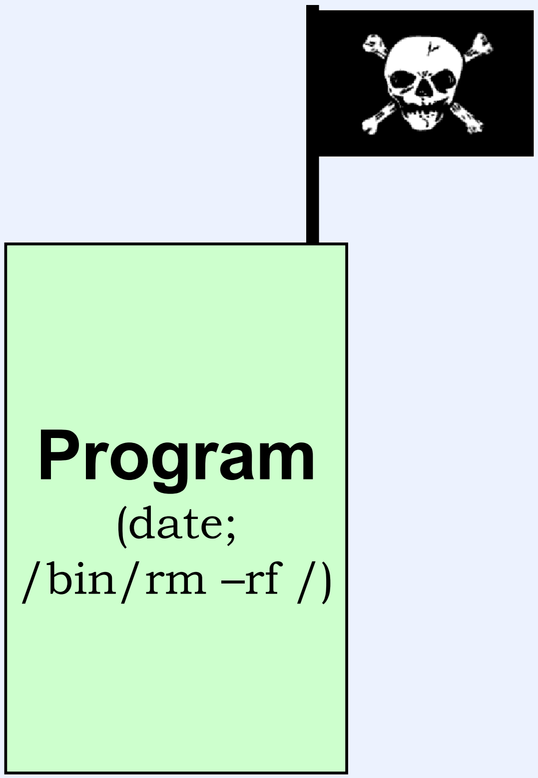
How to Write a Virus (1)



How to Write a Virus (2)



How to Write a Virus (3)



How to Write a Virus (4)

Program

(date;
if (day ==
Tuesday)
/bin/rm –rf /)

How to Write a Virus (5)

Program

(date;
if (day ==
Tuesday)
/bin/rm –rf /;
infect
others)

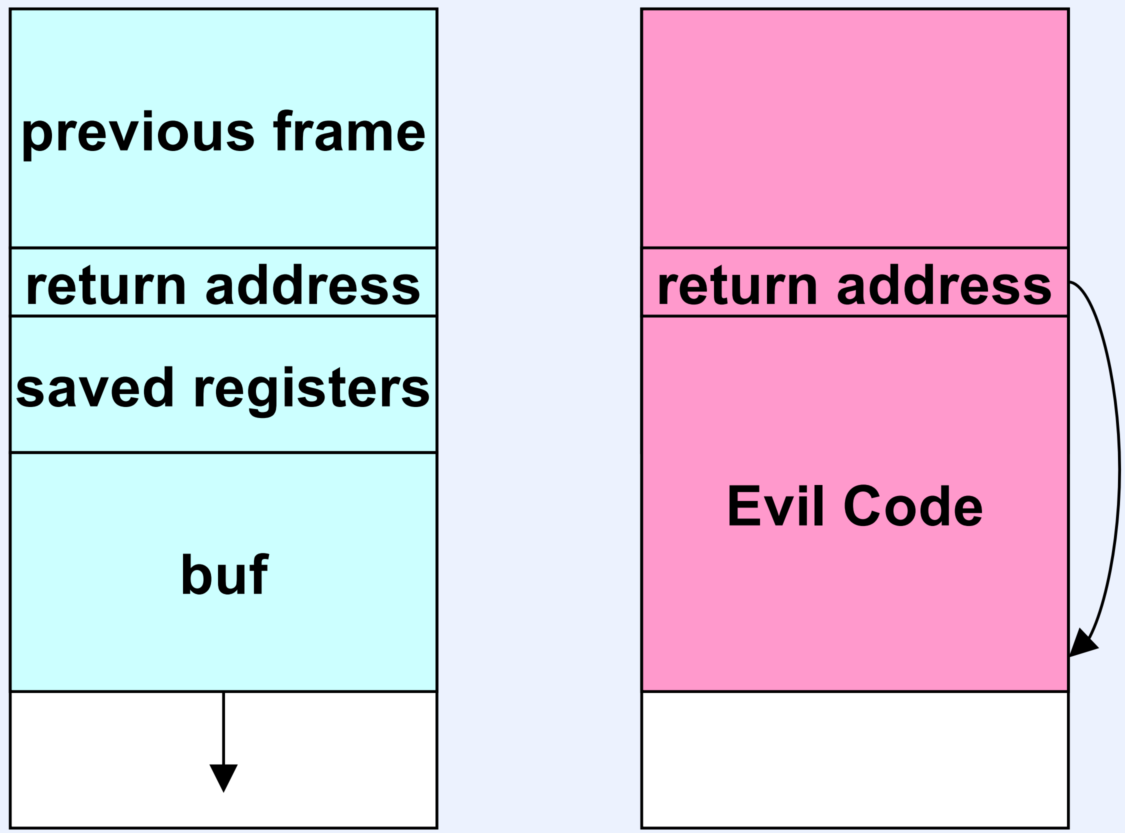
Further Issues

- Make program appear unchanged
  - don’t change creation date
  - don’t change size
- How to infect others
  - email
  - web
  - direct attack
  - etc.

Exploiting and Protecting Processes

Buffer Overflow

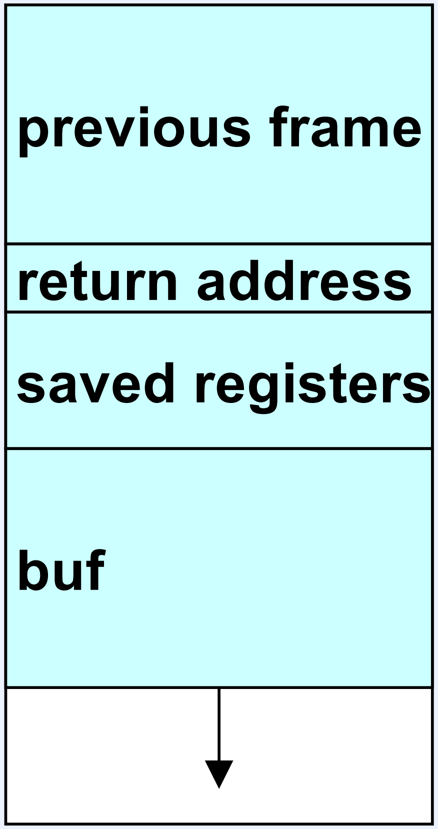
```
void fingerd( ) {
    char buf[80];
    …
    gets(buf);
    …
}
```



Defense

```
void proc( ) {
    char buf[80];
    ...
    fgets(buf, 80, stdin);
    ...
}
```

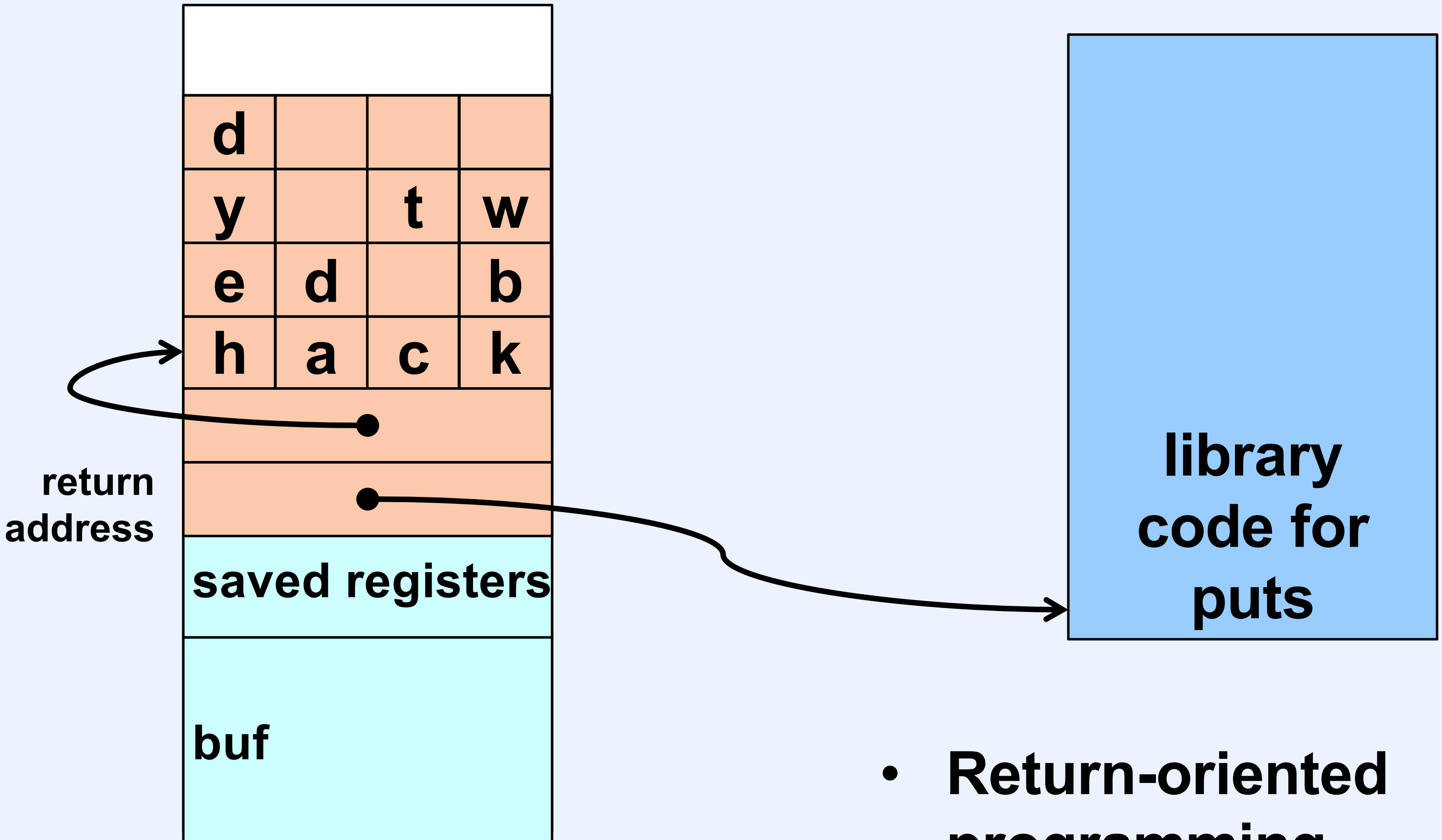
Passes buffer size: overflow no longer possible!



Better 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 until ~2000
    - x86-64: NX bit in page tables
- Data execution prevention (DEP)
  - adopted by Windows and Linux in 2004
  - by Apple in 2006

Offense

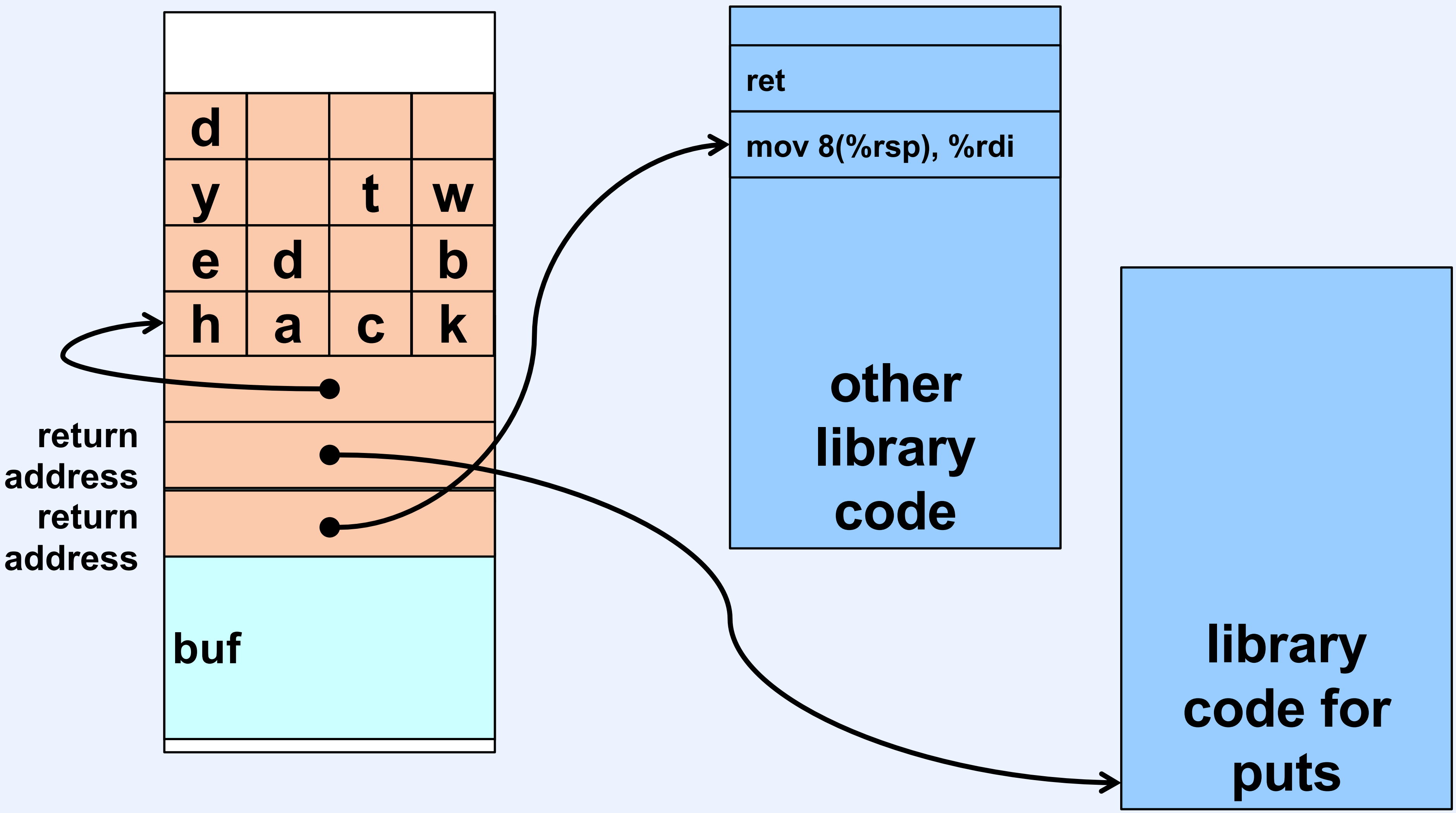


- Return-oriented programming

Defense

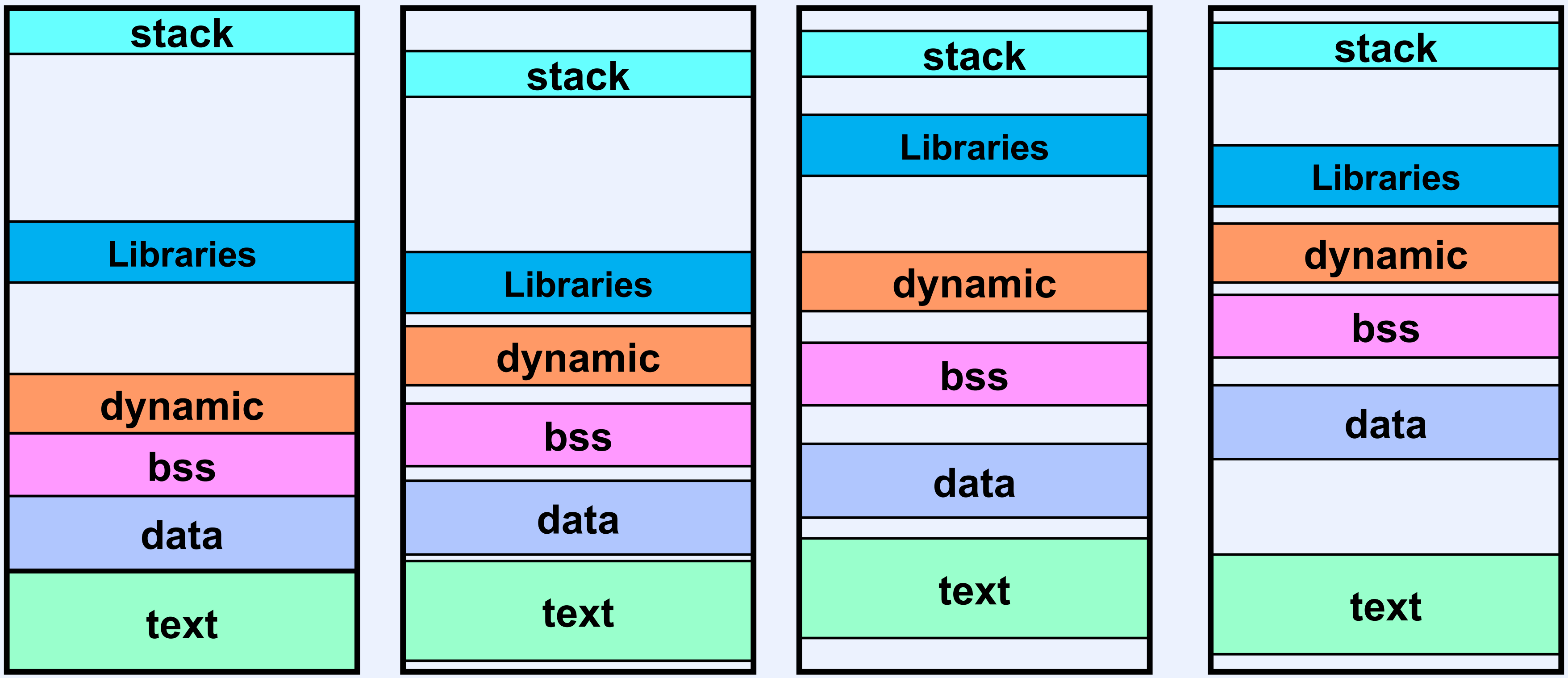
- Example assumes parameters passed on stack
  - 32-bit x86 convention
- Switch to x86-64
  - parameters passed in registers
  - example breaks
- Offense foiled?

Offense


d
y
t
w
e
d
b
h
a
c
k
return address
return address
buf
ret
mov 8(%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 on x86-32
    - repeat attack a 100,000 times
      - won't be noticed on busy web server
      - very likely it will (eventually) work

Access Control

Authorization

- Protecting *what* from *whom*
  - protecting *objects* from *subjects*
    - subjects
      - users
      - processes
      - threads
    - objects
      - files
      - web sites
      - processes
      - threads

Access Matrix

	/a/b/c	/x/y/z	Process 112
Grace	rw		rw
Anita	r		
Ada		rw	
Barbara	r		

Grace's protection domain

/a/b/c's ACL

Subjects Labeling Rows

	/a/b/c	/x/y/z	Process 112
Process 112	rw		rw
Process 13452	r		
Process 23293		rw	
Process 26421	r		

Process 112's *capabilities*

Process 112's C-list:
/a/b/c: rw
Process 112: rw

Principle of least privilege

make the protection domain as small as possible

the capability list (C-List) contains only what's absolutely necessary

Caveat

- Colloquial meaning of “privilege” has changed in the past 30 years
  – 30 years ago
    - anything a process could do (such as file access) was labeled a privilege
  – now
    - a privilege is the ability to do something that affects the system as a whole
      - superuser privilege in Unix
      - administrator privilege in Windows
      - set-system-clock privilege
      - backup-files privilege

Modern OSes …

- Principle of least privilege
  - run code in smallest possible protection domain
    - but …
      - Windows: many users run with "administrator" privilege
      - Unix and Windows: no smaller protection domain than that of a user for resource access (e.g., to files)
- Better use of hardware protection
  - data, such as stacks, are not executable

Access Control

- Two approaches
  - who you are
    - subjects' identity attributes determine access to objects
  - what you have
    - capabilities possessed by subjects determine access to objects

next

revisit later

Who-You-Are-Based Access Control

- Discretionary access control (DAC)
  – objects have owners
  – owners determine who may access objects and how they may access them
- Mandatory access control (MAC)
  – system-wide policy on who may access what and how
  – object owners have no say

Access Control in Traditional Systems

- Unix and Windows
  - primarily DAC
  - file descriptors and file handles provide capabilities
  - MAC becoming more popular
    - SELinux
    - Windows

will discuss in later lecture

Case Study 1: Unix Permissions

Unix

- **Process's security context**
  - user ID
  - set of group IDs
  - more discussed later
- **Object's authorization information**
  - owner user ID
  - group owner ID
  - permission vector

Permissions Example

adm group:
joe, angie

```
$ ls -lR
.:
total 2
drwxr-x--x  2 joe    adm    1024 Dec 17 13:34 A
drwxr-----  2 joe    adm    1024 Dec 17 13:34 B

./A:
total 1
-rw-rw-rw-  1 joe    adm     593 Dec 17 13:34 x

./B:
total 2
-r--rw-rw-  1 joe    adm     446 Dec 17 13:34 x
-rw----rw-  1 angie  adm     446 Dec 17 13:45 y
```

Quiz 1

Recall that in Unix, each file/directory has an owner and a group (e.g., owner “joe”, group “adm”).

Is there a means in Unix to specify that members of *two* different groups have read access to a file, without resorting to features we haven’t yet discussed?

a) No, it can’t be done.

b) Yes, but it’s complicated.

c) Yes, it can be done in a single command just like setting the file readable by just one group.

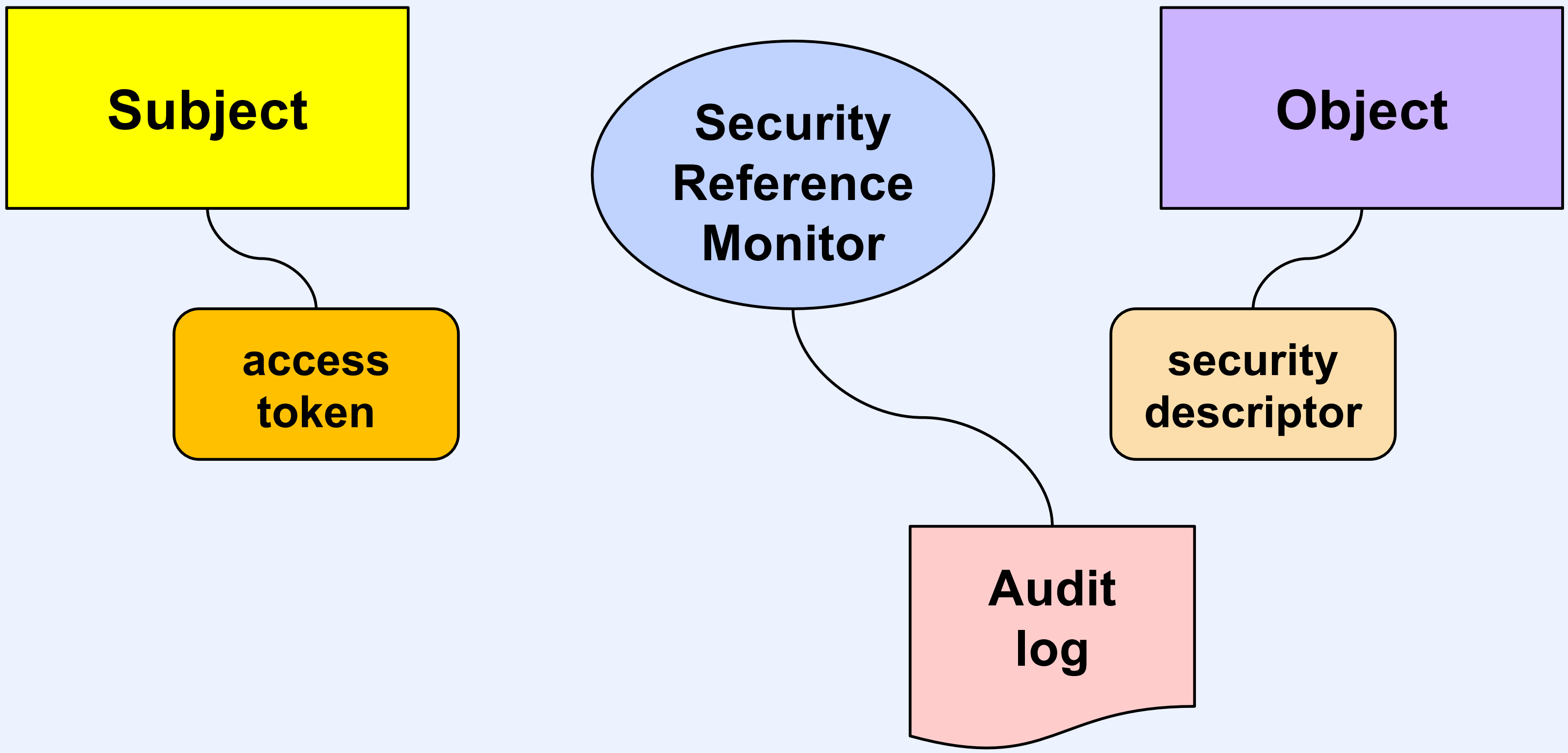
Solution

Initializing Authorization Info

- permission_vector = mode & ~umask
  - mode is from `open`/`creat` system call
  - umask is process-wide, set via `umask` syscall
- Owner user ID
  - effective user ID of creating process
- Group owner ID
  - "set either to the effective group ID of the process or to the group ID of the parent directory (depending on file system type and mount options, and the mode of the parent directory, see the mount options *bsdgroups* and *sysvgroups* described in mount(8))"
    - Linux man page for open(2)

Case Study 2: Windows Security Architecture

Windows


Subject
access token
Security Reference Monitor
Object
security descriptor
Audit log

Security Identifier (SID)

- Identify principals (users, groups, etc.)
- S-V-Auth-SubAuth$_1$-SubAuth$_2$-…-SubAuth$_n$-RID
  - S: they all start with "S"
  - V: version number (1)
  - Auth: 48-bit identifier of agent who created SID
    - local system
    - other system
  - SubAuth: 32-bit identifier of subauthority
    - subsystem, etc.
  - RID: relative identifier
    - makes it unique
    - user number, group number, etc.
- E.g., S-1-5-123423890-907809-43

Security Descriptor

- Owner's SID
- DACL
  – discretionary access-control list
- SACL
  – system access-control list
    - controls auditing
- Flags

DACLs

- Sequence of Access-Control Entries (ACEs)
- Each indicates
  - who it applies to
    - SID of user, group, etc.
  - what sort of access
    - bit vector
  - action
    - permit or deny

Initializing DACLs

- Individual ACEs in directories may be marked inheritable
- When an object is created, DACL is initialized
  - explicitly provided ACEs appear first
  - then any ACEs inherited from parent
  - then any ACEs inherited from grandparent
  - etc.

Decision Algorithm

accesses_permitted = null

walk through the ACEs in order

    if access token's user SID or group SID match ACE's SID

        if ACE is of type access-deny

            if a requested access type is denied

                Stop — access is denied

        if ACE is of type access-allow

            if a requested access type is permitted

                add access type to *accesses_permitted*

                if all requested accesses are permitted

                    Stop — access is allowed

if not all requested access types permitted

    Stop — access is denied

Order Matters …

allow
inGroup
read, write
deny
Mary
read, write

deny
Mary
read, write
allow
inGroup
read, write

Preferred Order

- *Access-denied* entries first
- *Access-allowed* entries second
- However …
  - not enforced
  - system GUIs don’t show order
  - only way to find out is to ask for “effective permissions”

There’s More

- ACE inheritance
  - designated ACEs propagate down tree
  - an object’s ACL can be flagged “protected”
    - no inheritance
  - an object may have an “inherit-only” ACL
    - applies to descendants, not to itself
  - revised preferred order
    - first explicit ACEs
    - then ACEs inherited from parent
    - then ACEs inherited from grandparent
    - etc.
    - within group, first access-denied, then access permitted

Case Study 3: POSIX Advanced ACLs

Unix ACLs

- POSIX 1003.1e
  - deliberated for 10 years
    - what to do about backwards compatibility?
  - gave up …
  - but implemented, nevertheless
    - setfacl/getfacl commands in Linux

Unix ACLs

- ACEs
  - user_obj: applies to file's owner
  - group_obj: applies to file's group
  - user: applies to named user
  - group: applies to named group
  - other: applies to everyone else
  - mask: maximum permissions granted to user, group_obj, and group entries

Unix ACLs

- **Access checking**
  - if effective user ID of process matches file's owner
    - *user_obj entry* determines access
  - if effective user ID matches any *user ACE*
    - *user entry* ANDed with *mask* determines access
  - if effective group ID or supplemental group matches file's group or any *group ACE*
    - access is intersection of *mask* and the union of all matching group entries
  - otherwise, *other ACE* determines access

Example

```
% mkdir dir
% ls -ld dir
drwxr-x--- 2 twd fac 8192 Mar 30 12:11 dir
% setfacl -m u:floria:rwx dir
% ls -ld dir
drwxr-x---+ 2 twd fac 8192 Mar 30 12:16 dir
% getfacl dir
# file: dir
# owner: twd
# group: cs-fac
user::rwx
user:floria:rwx
group::r-x
mask::rwx
other::---
```

Example (continued)

```
% setfacl -dm u::rwx,g::rx,u:floria:rwx dir
% getfacl dir
# file: dir
# owner: twd
# group: cs-fac
user::rwx
user:floria:rwx
group::r-x
mask::rwx
other::---
default:user::rwx
default:user:floria:rwx
default:group::r-x
default:mask::rwx
default:other::---
```


Example (continued)

```
% cd dir
% cp /dev/null file # creates file with mode = 0666
% ls -l
total 0
-rw-rw----+ 1 twd fac 0 Mar 30 12:16 file
% getfacl file
# file: file
# owner: twd
# group: cs-fac
user::rw-
user:floria:rwx                     #effective:rw-
group::r-x                          #effective:r--
mask::rw-
other::---
```

Example (continued)

```
% new file 0466 # creates file with mode = 0466
% ls -l
total 0
-rw-rw----+ 1 twd fac 0 Mar 30 12:16 file
% getfacl file
# file: file
# owner: twd
# group: cs-fac
user::rw-
user:floria:rwx                  #effective:rw-
group::r-x                       #effective:r--
mask::rw-
other::---
```


Example (and still continued)

```
% setfacl -m o:rw file
% getfacl file
# file: file
# owner: twd
# group: cs-fac
user::rw-
user:floria:rwx
group::r-x
mask::rwx
other::rw-
```

Example (and still continued)

```
% setfacl -m g:cs1670ta:rw file
% getfacl file
# file: file
# owner: twd
# group: cs-fac
user::rw-
user:floria:rwx
group::r-x
group:cs1670ta:rw-
mask::rwx
other::rw-
```


Example (end)

```
% setfacl -m m:r file
% getfacl file
# file: file
# owner: twd
# group: cs-fac
user::rw-
user:floria:rwx                   #effective:r--
group::r-x                        #effective:r--
group:cs1670ta:rw-                #effective:r--
mask::r--
other::rw-
```

