

# Internet Security 2

(aka Advanced InetSec)

## General Unix Security

Christian Platzer	<a href="mailto:cplatzer@seclab.tuwien.ac.at">cplatzer@seclab.tuwien.ac.at</a>
Paolo Milani Comparetti	<a href="mailto:pmilani@seclab.tuwien.ac.at">pmilani@seclab.tuwien.ac.at</a>
Clemens Kolbitsch	<a href="mailto:ck@seclab.tuwien.ac.at">ck@seclab.tuwien.ac.at</a>
Thorsten Holz	<a href="mailto:tho@seclab.tuwien.ac.at">tho@seclab.tuwien.ac.at</a>

# News from the Lab

*Int. Secure Systems Lab  
Technical University Vienna*

- Registration
  - has been opened today
  - <http://inetsec.iseclab.org/secanmeld/service.php>
- Challenge 1
  - has been issued
  - deadline is next Monday, October 19th
  - have fun :-)
- Challenges
  - open 17:00, close 15:00
  - again: no, they are strict deadlines, no matter what

# News from the Lab

*Int. Secure Systems Lab  
Technical University Vienna*

- UCSB iCTF
  - <http://ictf.cs.ucsb.edu/>
- Lots of fun, but work involved
  - we plan on having preparation meetings
  - should be fun, a lot of practical learning involved
  - (hopefully) organized by one of our master students
- Eager to join preparation meetings?
  - good! :-)
  - send an email to [ck@iseclab.org](mailto:ck@iseclab.org), I'll add you to our mailing list

# Overview

*Int. Secure Systems Lab  
Technical University Vienna*

- News from the lab
- OS layers / ring separation
  - system calls
  - vulnerabilities
- Unix security
  - users
  - groups
  - processes
  - authentication
- Shell
  - attacks
  - resource limits
  - signals
  - Libraries

# Unix

*Int. Secure Systems Lab  
Technical University Vienna*

- Multi-user operating system
- Operating system functionality
  - process management
  - (virtual) memory management
  - file system management
  - I/O management
- Structure
  - operating system kernel
  - user-space programs (daemons, applications, shell)

# Unix

*Int. Secure Systems Lab  
Technical University Vienna*

- Kernel
  - provides an hardware abstraction layer for user-space programs
  - complete access to all (physical) resources
  - trusted computing base
  - provides services via system calls
- System call
  - performs a transition from user mode to privileged (kernel) mode
  - this crosses the border between two security domains
  - usually implemented with hardware (processor) support
    - processor interrupt
    - x86 call gates

# Unix

*Int. Secure Systems Lab  
Technical University Vienna*

- System call
  - performs a transition from user mode to privileged (kernel) mode

User: CPU ring 3

```
code:          ba 0d 00 00 00      mov $0xd,%edx
code + 0x05:   8b 0d 10 a0 04 08    mov 0x804a010,%ecx
code + 0x0b:   bb 01 00 00 00      mov $0x1,%ebx
code + 0x10:   b8 04 00 00 00      mov $0x4,%eax
code + 0x15:   cd 80              int $0x80
```

```
code + 0x17:   bb 01 00 00 00      mov $0x1,%ebx
```

Kernel:

CPU ring 0  
Complete access  
to everything

fwrite to file

# Unix

*Int. Secure Systems Lab  
Technical University Vienna*

- System call
  - performs a transition from user mode to privileged (kernel) mode

User: CPU ring 3

```
code:          ba 0d 00 00 00      mov $0xd,%edx
code + 0x05:   8b 0d 10 a0 04 08      mov 0x804a010,%ecx
code + 0x0b:   bb 01 00 00 00      mov $0x1,%ebx
code + 0x10:   b8 04 00 00 00      mov $0x4,%eax
code + 0x15:   cd 80                int $0x80
```

```
code + 0x17:   bb 01 00 00 00      mov $0x1,%ebx
```

Kernel:

CPU ring 0  
Complete access  
to everything

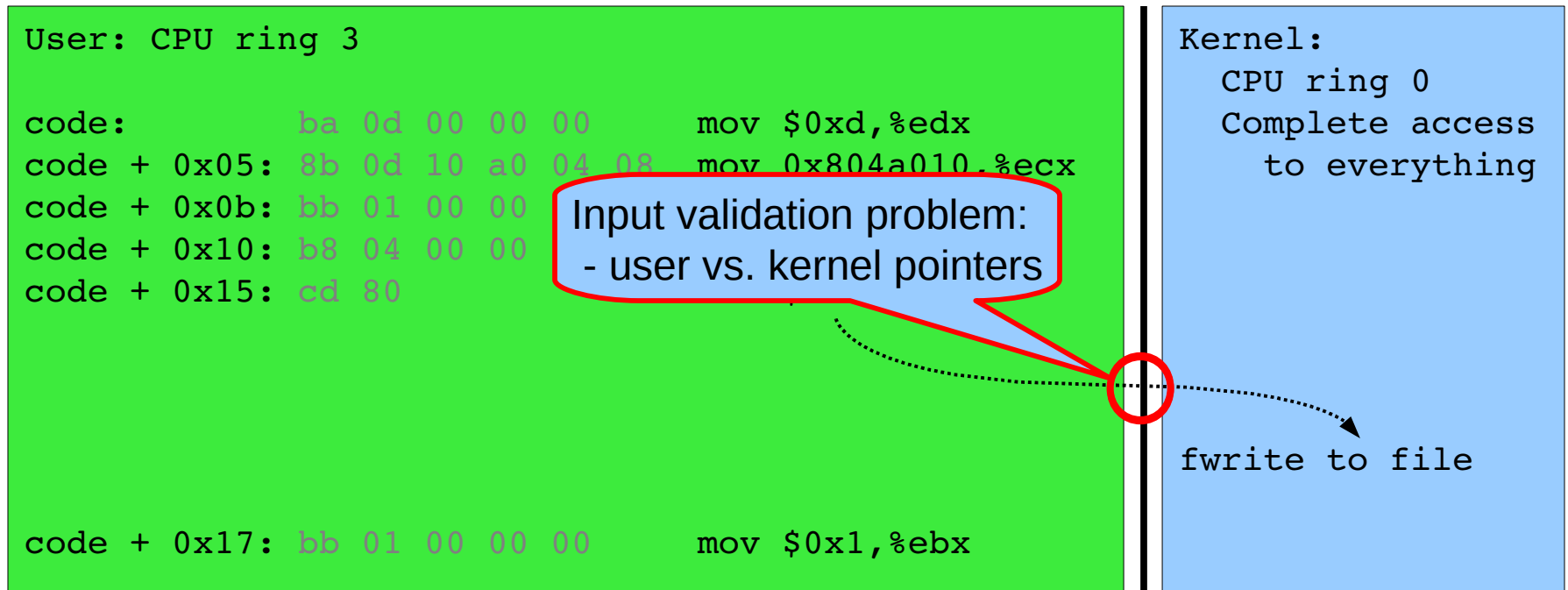
fwrite to file



# Unix

*Int. Secure Systems Lab  
Technical University Vienna*

- System call
  - performs a transition from user mode to privileged (kernel) mode



# Unix

Int. Secure Systems Lab  
Technical University Vienna

- Kernel vulnerability
  - usually leads to complete system compromise
  - attacks performed via system calls
- Solaris / NetBSD call gate creation input validation problem
  - malicious input when creating a LDT (x86 local descriptor table)
  - used in 2001 by *Last Stage of Delirium* to win *Argus Pitbull* competition
- Kernel Integer Overflows
  - FreeBSD *procf*s code (September 2003)
  - Linux *brk* () used to compromise debian.org (December 2003)
  - Linux *setsockopt* () (May 2004)
- Linux Memory Management
  - *mremap* () and *munmap* () (March 2004)
  - *kprobes* setup (February 2009)

# Unix

Int. Secure Systems Lab  
Technical University Vienna

- Device driver code is particularly vulnerable
  - (most) drivers run in kernel mode
    - either kernel modules or compiled-in
  - often not well audited
  - very large code based compared to core services
- Examples
  - *aironet*, *asus\_acpi*, *decnet*, *mpu401*, *msnd*, and *pss* all in May 2004
  - found by sparse (tool developed by Linus Torvalds)
    - annotation-based, static (compile-time) analysis tool
    - check (user vs. kernel) pointers
    - check locks
  - *Madwifi* remote DOS, October 2007

# Unix

Int. Secure Systems Lab  
Technical University Vienna

- Code running in user mode is **always** linked to a certain identity
  - security checks and access control decisions are based on user identity
- Unix is user-centric
  - no roles
- User
  - identified by user name (UID), group name (GID)
  - authenticated by password (stored encrypted)
- User *root*
  - superuser, system administrator
  - special privileges (access resources, modify OS)
  - cannot decrypt user passwords

# Process Management

*Int. Secure Systems Lab  
Technical University Vienna*

- Process
  - implements user-activity
  - entity that executes a given piece of code
  - has its own execution stack, memory pages, and file descriptors table
  - separated from other processes using the virtual memory abstraction
- Thread
  - separate stack and program counter
  - share memory pages and file descriptor table

# Process Management

*Int. Secure Systems Lab  
Technical University Vienna*

- Process Attributes
  - process ID (PID)
    - uniquely identified process
  - user ID (UID)
    - ID of owner of process
  - effective user ID (EUID)
    - ID used for permission checks (e.g., to access resources)
  - saved user ID (SUID)
    - to temporarily drop and restore privileges
  - lots of management information
    - scheduling
    - memory management, resource management

# User Authentication

Int. Secure Systems Lab  
Technical University Vienna

- How does a process get a user ID?
  - Authentication (*login*)
- Passwords
  - user passwords are used as keys for *crypt ()* function
  - runs DES algorithm 25 times on a block of zeros
  - 12-bit “salt”
    - 4096 variations
    - chosen from date, not secret
    - prevent same passwords to map onto same string
    - make dictionary attacks more difficult
- Password cracking
  - dictionary attacks
  - *crack, JohnTheRipper*

# User Authentication

- Shadow passwords
  - password file is needed by many applications to map user ID to user names
  - thus, /etc/passwd is readable by user
  - encrypted passwords are not stored there
- /etc/shadow
  - holds encrypted passwords
  - account information
    - last change date
    - expiration (warning, disabled)
    - minimum/maximum change frequency
  - readable only by superuser and privileged programs
  - MD5 hashed passwords to slow down guessing

# Group Model

Int. Secure Systems Lab  
Technical University Vienna

- Users belong to one or more groups
  - primary group (stored in */etc/password*)
  - additional groups (stored in */etc/group*)
  - possibility to set group password
  - and become group member with *newgrp*
- */etc/group*  
groupname : password : group id : additional users  
root:x:0:root  
bin:x:1:root,bin,daemon  
users:x:100:ck
- Special group *wheel*
  - protect *root* account by limiting user accounts that can perform *su*
  - *sudo/sudoers* has similar functionality

# File System

Int. Secure Systems Lab  
Technical University Vienna

- File tree
  - primary repository of information
  - hierarchical set of directories
  - directories contain file system objects (FSO)
  - root is denoted “/”
- File system object
  - files, directories, symbolic links, sockets, device files
  - referenced by *inode* (index node)
  - „everything is a file“

# File System

- Access Control
  - permission bits
  - *chmod, chown, chgrp, umask*
  - file listing:

–            *rwX*            *rwX*            *rwX*  
*(file type) (user) (group) (other)*

Type	r	w	X	S	t
<b>File</b>	read access	write access	execute	suid / sgid inherit id	sticky bit
<b>Directory</b>	list files	insert and remove files	stat / execute files, chdir	new files have dir-gid	files only delete- able by owner

# SUID Programs

- Each process has *real* and *effective* user / group ID
  - usually identical
  - real IDs
    - determined by current user
    - *login, su*
  - effective IDs
    - determine the “rights” of a process
    - system calls (e.g., *setuid()*)
    - *suid / sgid* bits
  - attractive target for attacker
- You cannot SUID shell scripts anymore

# Shell

- Shell
  - one of the core Unix application
  - both a command language and programming language
  - provides an interface to the Unix operating system
  
  - rich features such as control-flow primitives, parameter passing, variables, and string substitution
  - communication between shell and spawned programs via redirection and pipes
  
  - different flavors
    - bash and sh, tcsh and csh, ksh

# Shell Attacks

*Int. Secure Systems Lab  
Technical University Vienna*

- Environment Variables
  - \$HOME and \$PATH can modify behavior of programs that operate with relative path names
  - \$IFS – internal field separator
    - used to parse tokens
    - usually set to [ \t\n] but can be changed to “/”
    - “/bin/ls” is parsed as “bin ls” calling bin locally
    - IFS now only used to split expanded variables
  - preserve attack (/usr/lib/preserve is SUID)
    - called “/bin/mail” when vi crashes to preserve file
    - change IFS, create bin as link to /bin/sh, kill vi

# Shell Attacks

*Int. Secure Systems Lab  
Technical University Vienna*

- Control and escape characters
  - can be injected into command string
  - modify or extend shell behavior
  - user input used for shell commands has to be rigorously sanitized
  - easy to make mistakes
  - classic examples are `;` and `&`
- Applications that are invoked via shell can be targets as well
  - increased vulnerability surface
- Restricted shell
  - invoked with `-r`
  - more controlled environment

# Shell Attacks

*Int. Secure Systems Lab  
Technical University Vienna*

- `system(char *cmd)`
  - function called by programs to execute other commands
  - invokes shell
  - executes string argument by calling `/bin/sh -c string`
  - makes binary program vulnerable to shell attacks
  - especially when user input is utilized
- `popen(char *cmd, char *type)`
  - forks a process, opens a pipe and invokes shell for cmd

# File Descriptor Attacks

*Int. Secure Systems Lab  
Technical University Vienna*

- SUID program opens file
- forks external process
  - sometimes under user control
- on-execute flag
  - if `close-on-exec` flag is not set, then new process inherits file descriptor
  - malicious attacker might exploit such weakness
- Linux Perl 5.6.0
  - `getpwuid()` leaves `/etc/shadow` opened (June 2002)
  - problem for Apache with `mod_perl`

# Resource Limits

- File system limits
  - *quotas*
  - restrict number of storage blocks and number of inodes
  - hard limit
    - can never be exceeded (operation fails)
  - soft limit
    - can be exceeded temporarily
  - can be defined per mount-point
  - defend against resource exhaustion (denial of service)
- Process resource limits
  - number of child processes, open file descriptors

# Signals

- Signal
  - simple form of interrupt
  - asynchronous notification
  - can happen anywhere for process in user space
  - used to deliver segmentation faults, reload commands, ...
  - *kill* command
- Signal handling
  - process can install signal handlers
  - when no handler is present, default behavior is used
    - ignore or kill process
  - possible to catch all signals except SIGKILL (-9)

# Signals

*Int. Secure Systems Lab  
Technical University Vienna*

- Security issues
  - code has to be re-entrant
    - atomic modifications
    - no global data structures
  - race conditions
  - unsafe library calls, system calls
    - e.g., signal during fprintf, call to fprintf during signal handler
  - examples
    - wu-ftpd 2001, sendmail 2001 + 2006, stunnel 2003, ssh 2006
- Secure signals
  - write handler as simple as possible
  - block signals in handler
  - call only asynchronous-safe functions

# Shared Libraries

Int. Secure Systems Lab  
Technical University Vienna

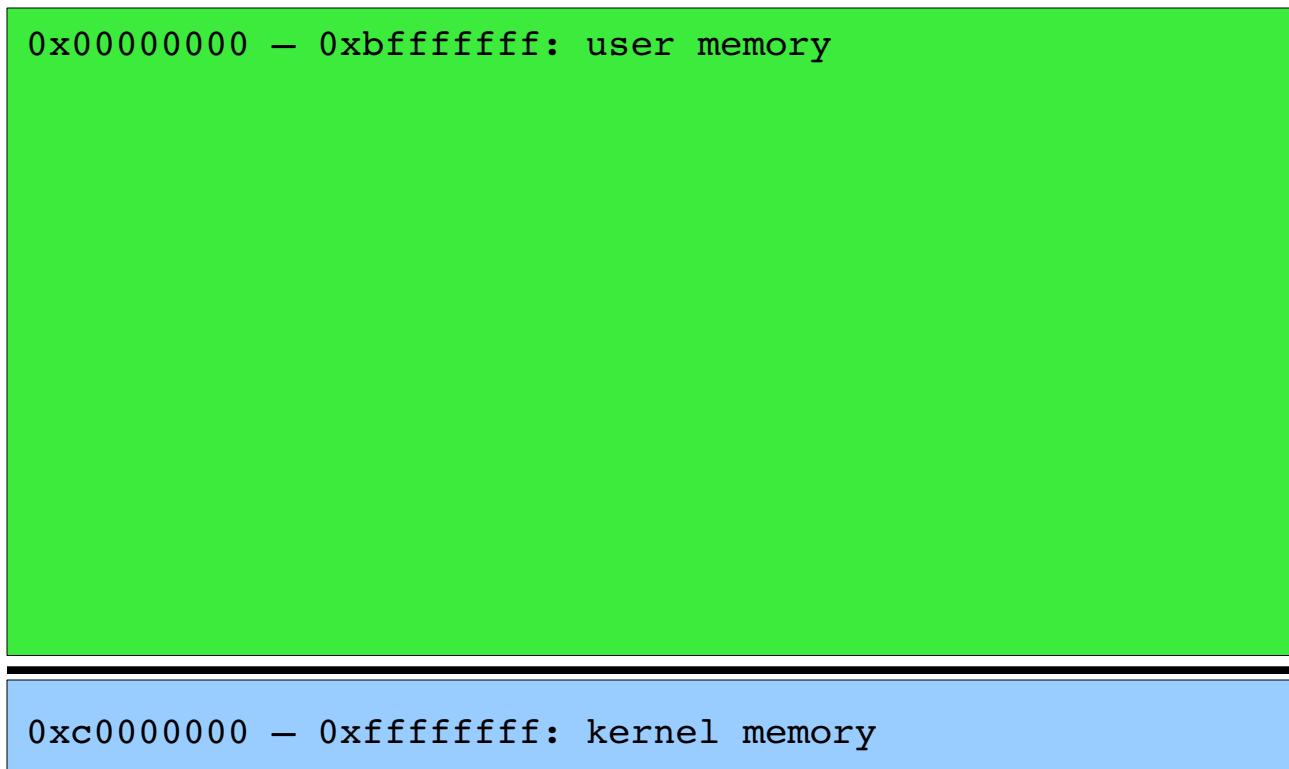
- Library
  - collection of object files
  - included into (linked) program as needed
  - code reuse
- Shared library
  - multiple processes share a **single** library copy
  - save disk space (program size is reduced)
  - save memory space (only a single copy in memory)
  - used by virtually all Unix applications (at least libc.so)
  - check binaries with *ldd*

# Shared Libraries

- Static shared library
  - address binding at link-time
  - not very flexible when library changes
  - code is fast
- Dynamic shared library
  - address binding at load-time
  - uses procedure linkage table (PLT) and global offset table (GOT)
  - code is slower (indirection)
  - loading is slow (binding has to be done at run-time)
  - classic *.so* or *.dll* libraries
- PLT and GOT entries are very popular attack targets
  - more when discussing buffer overflows

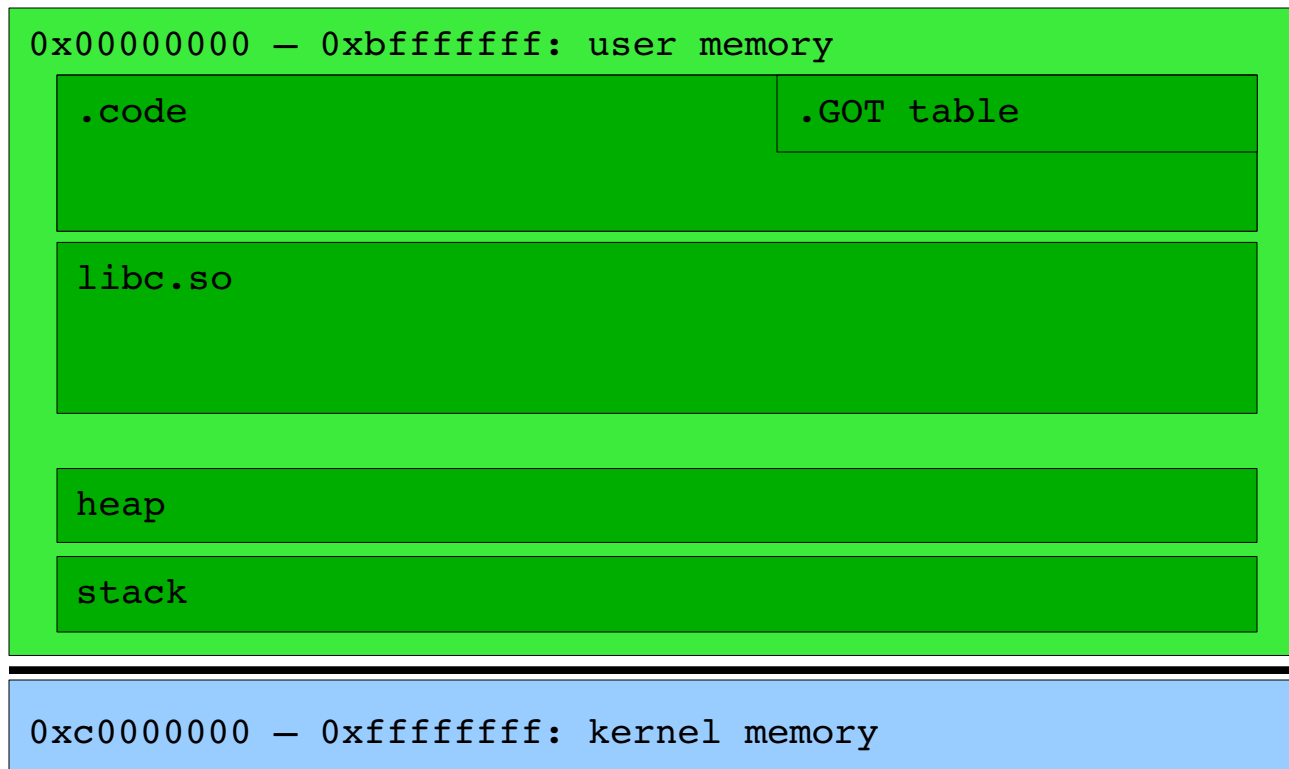
# Shared Libraries

- Process layout (32 bit systems)



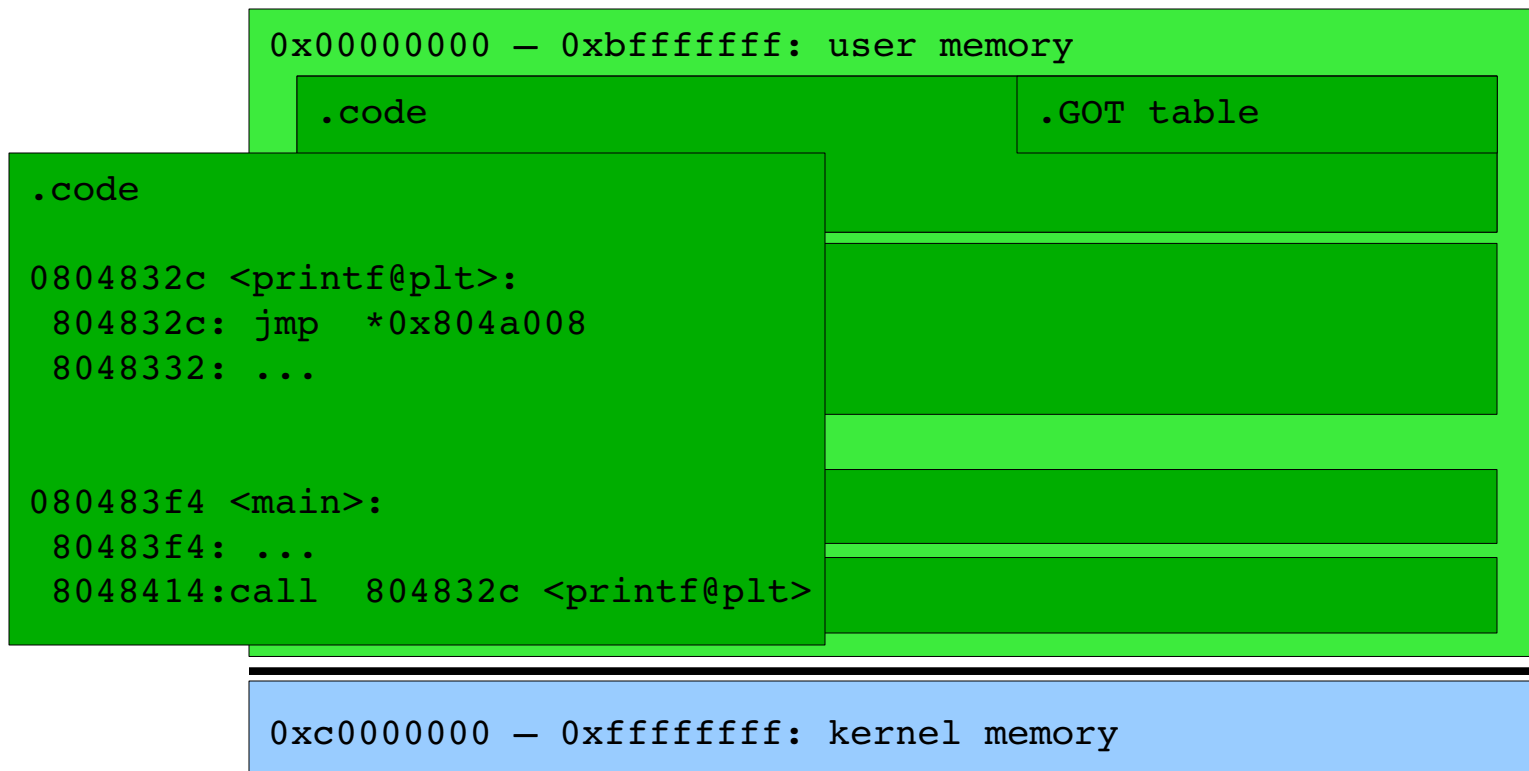
# Shared Libraries

- Process layout (32 bit systems)



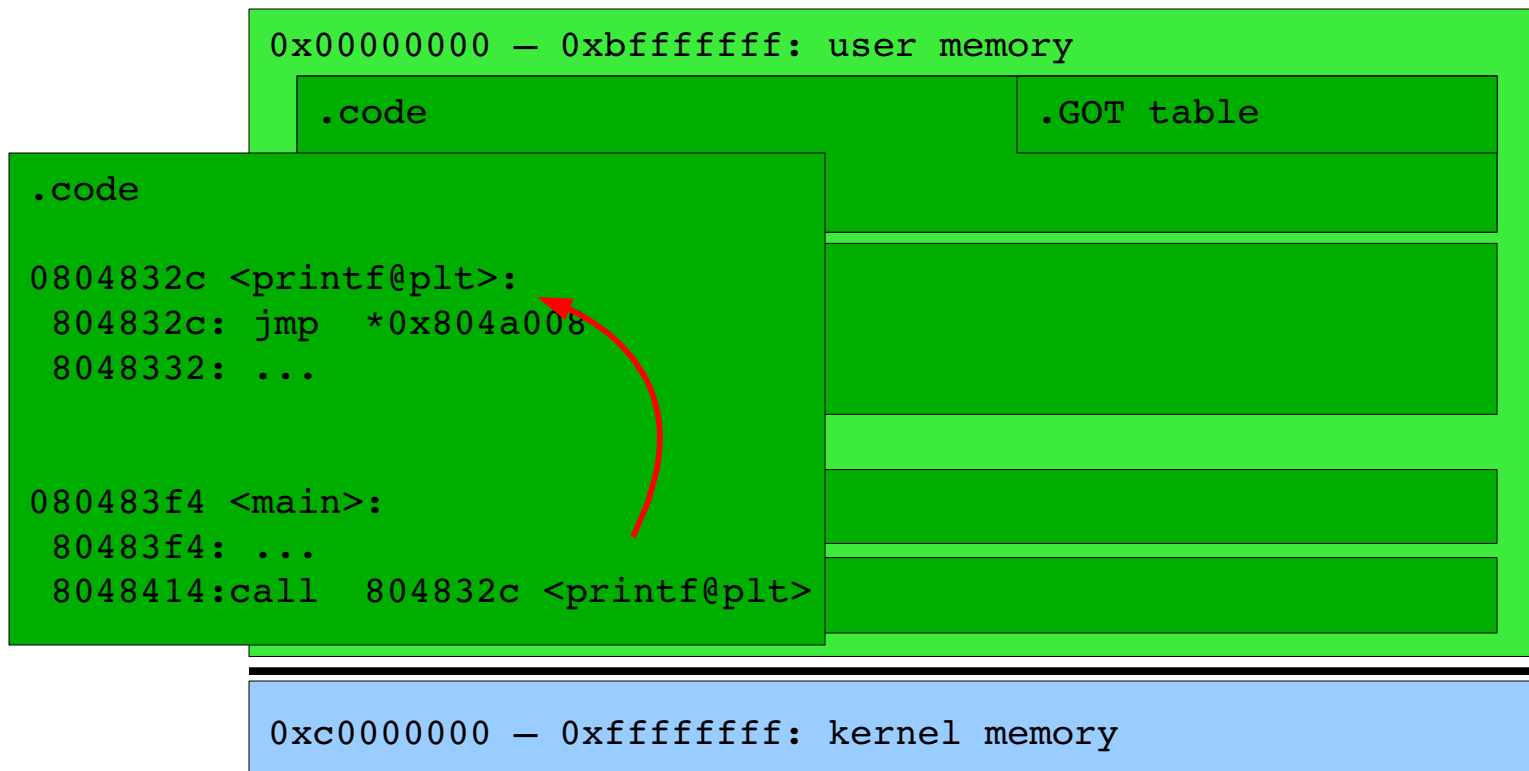
# Shared Libraries

- Process layout (32 bit systems)



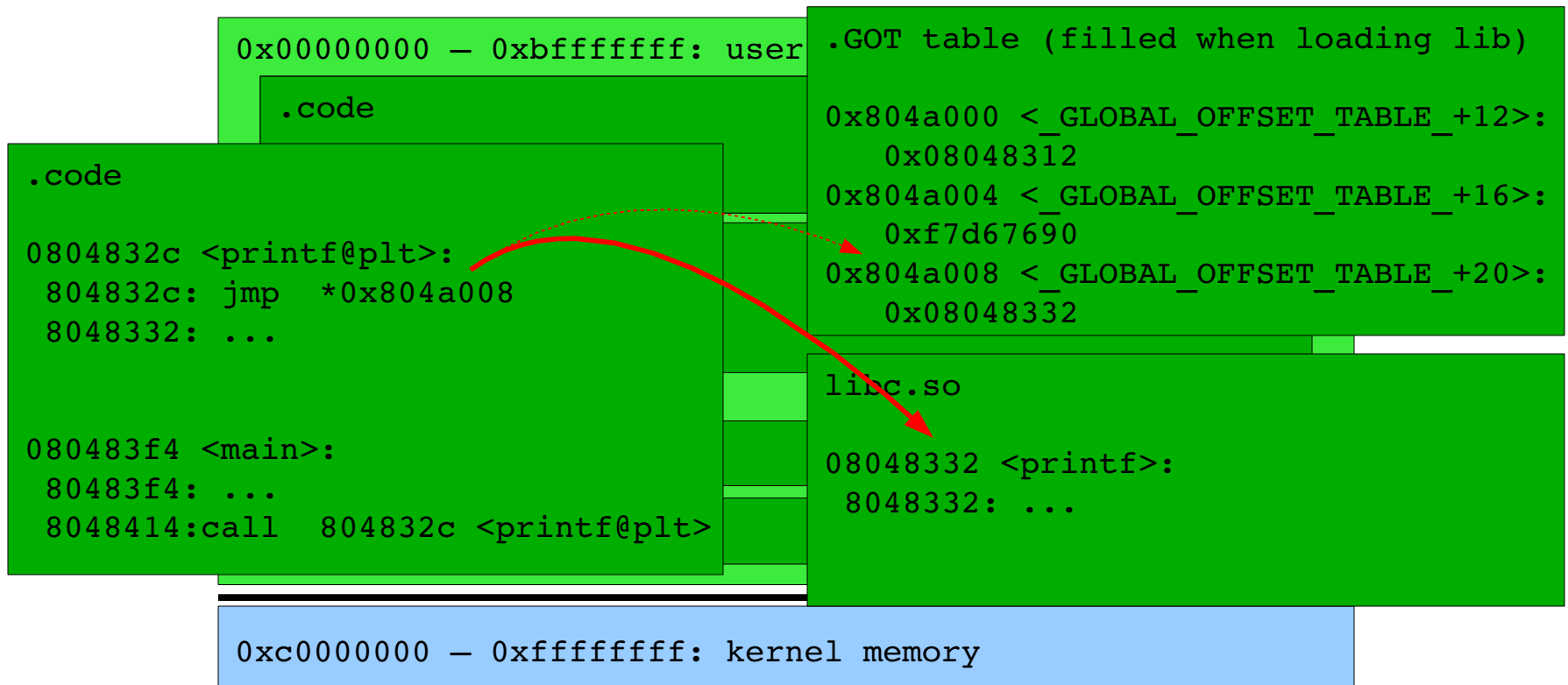
# Shared Libraries

- Process layout (32 bit systems)



# Shared Libraries

- Process layout (32 bit systems)



# Shared Libraries

Int. Secure Systems Lab  
Technical University Vienna

- Management
  - stored in special directories (listed in */etc/ld.so.conf*)
  - manage cache with *ldconfig*
- Preload
  - override (substitute) with other version
  - use */etc/ld.so.preload*
  - can also use environment variables for override
  - possible security hazard
  - now disabled for SUID programs (old Solaris vulnerability)

# Conclusion

*Int. Secure Systems Lab  
Technical University Vienna*

- We looked at UNIX security
  - Authentication
  - Process management
  - Shell attacks
- Outlook:
  - Next week: Advanced buffer overflows
  - 2 challenges for exploiting overflows
- Enjoy challenge 1!