Network Programming

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani

Pilani Campus

WRITING SECURE PRIVILEGED PROGRAMS

Privileged Programs

- Privileged programs have access to features and resources (files, devices) that are not available to ordinary users.
- A program can run with privileges by two general means:
  - The program was started under a privileged user ID. Many daemons and net-work servers, which are typically run as root, fall into this category.
  - The program has its set-user-ID or set-group-ID permission bit set.
- If a privileged program contains bugs, or can be subverted by a malicious user, then the security of the system or an application can be compromised.
- From a security viewpoint, we should write programs so as to minimize both the chance of a compromise and the damage that can be done if a compromise does occur.

Privileged Programs

innovate achieve lead

- Recommended practices for secure programming
- Pitfalls that should be avoided when writing privileged programs

Is a Set-User-ID or Set-Group-ID Program Required?

- Avoid writing them whenever possible.
- If there is an alternative way of performing a task that doesn't involve giving a program privilege, we should employ that.
  - Isolate the functionality that needs privilege into a separate program, and exec that program in a child process
  - Since root privileges are not always required, privileges can be given in other ways.
  - Example: writing on a file that doesn't allow user to write.
    - Create a group g. Assign group ownership of the file as g.
    - Set the set-group-id flag in the executable.

Operate with Least Privilege

- Hold privileges only while they are required

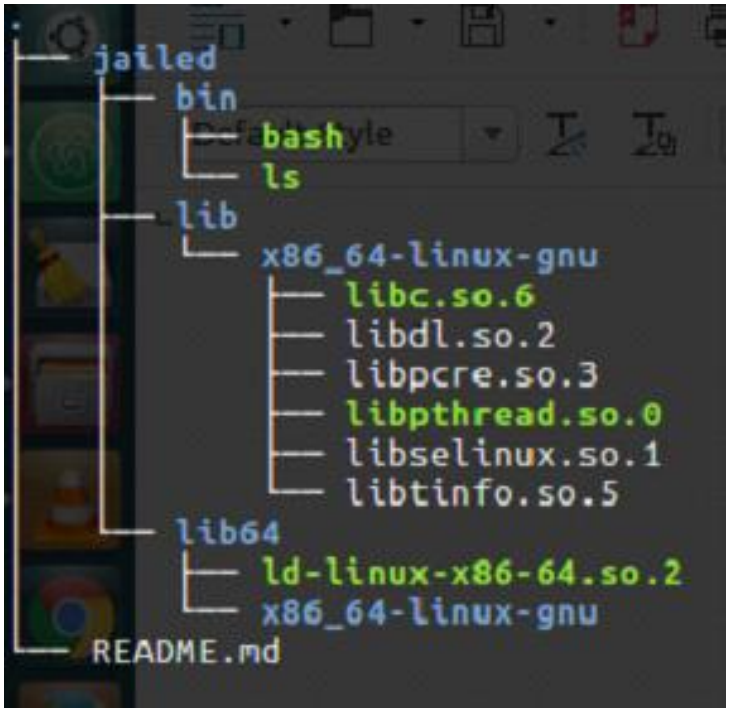
```
1  uid_t orig_euid;
2  orig_euid = geteuid();
3  if (seteuid(getuid()) == -1)            /* Drop privileges */
4      errExit("seteuid");
5  /* Do unprivileged work */
6  if (seteuid(orig_euid) == -1)           /* Reacquire privileges */
7      errExit("seteuid");
8  /* Do privileged work */
```

Be Careful When Executing a Program

- Drop privileges permanently before execing another program
- Close all unnecessary file descriptors before an exec()

Confine the Process

- Using a chroot jail
- A useful security technique in certain cases is to establish a chroot jail to limit the set of directories and files that a program may access
  - call chdir() to change the process's current working directory to a location within the jail
- However, that a chroot jail is insufficient to confine a set-user-ID-root program

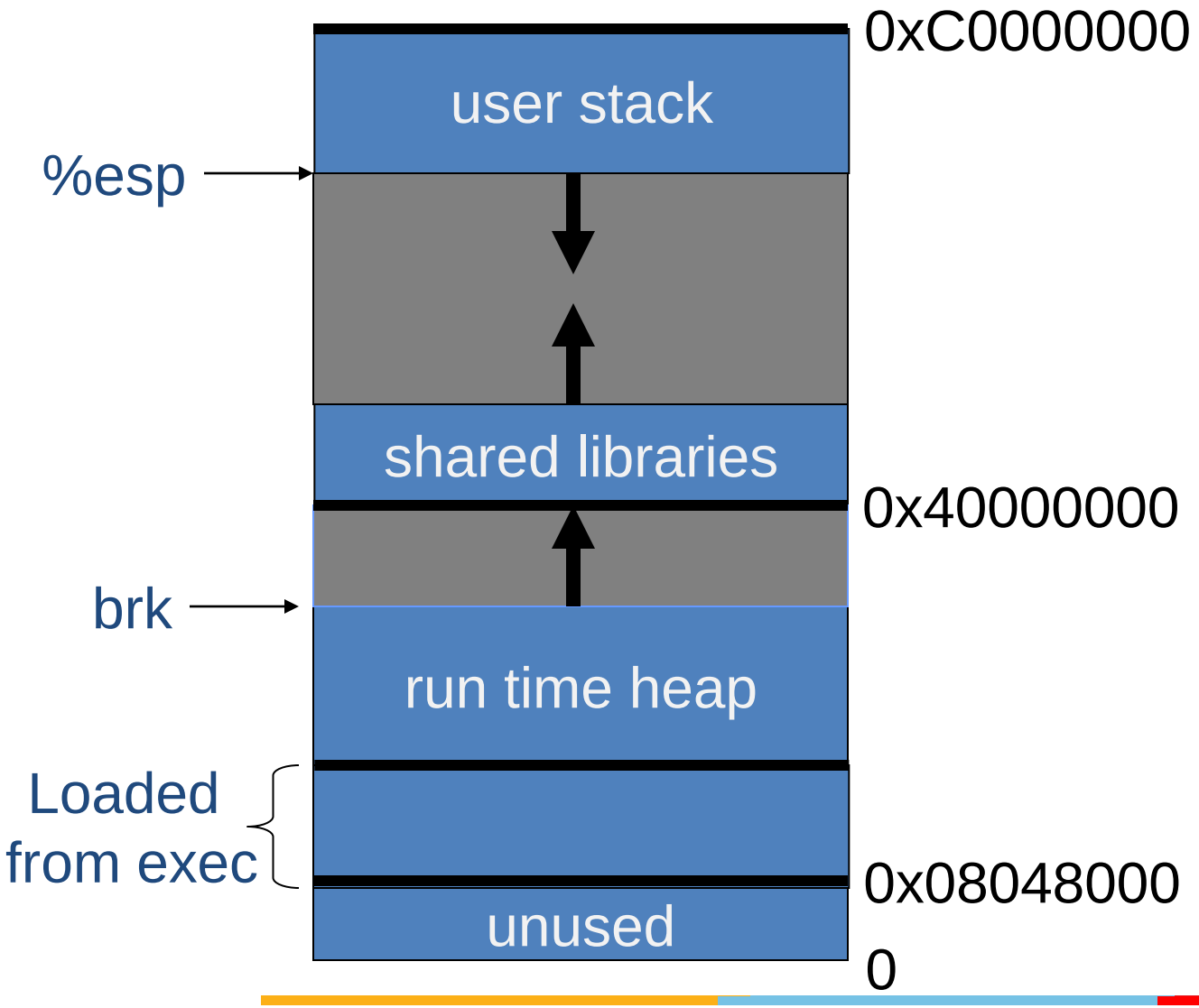


Beware of Buffer Overruns

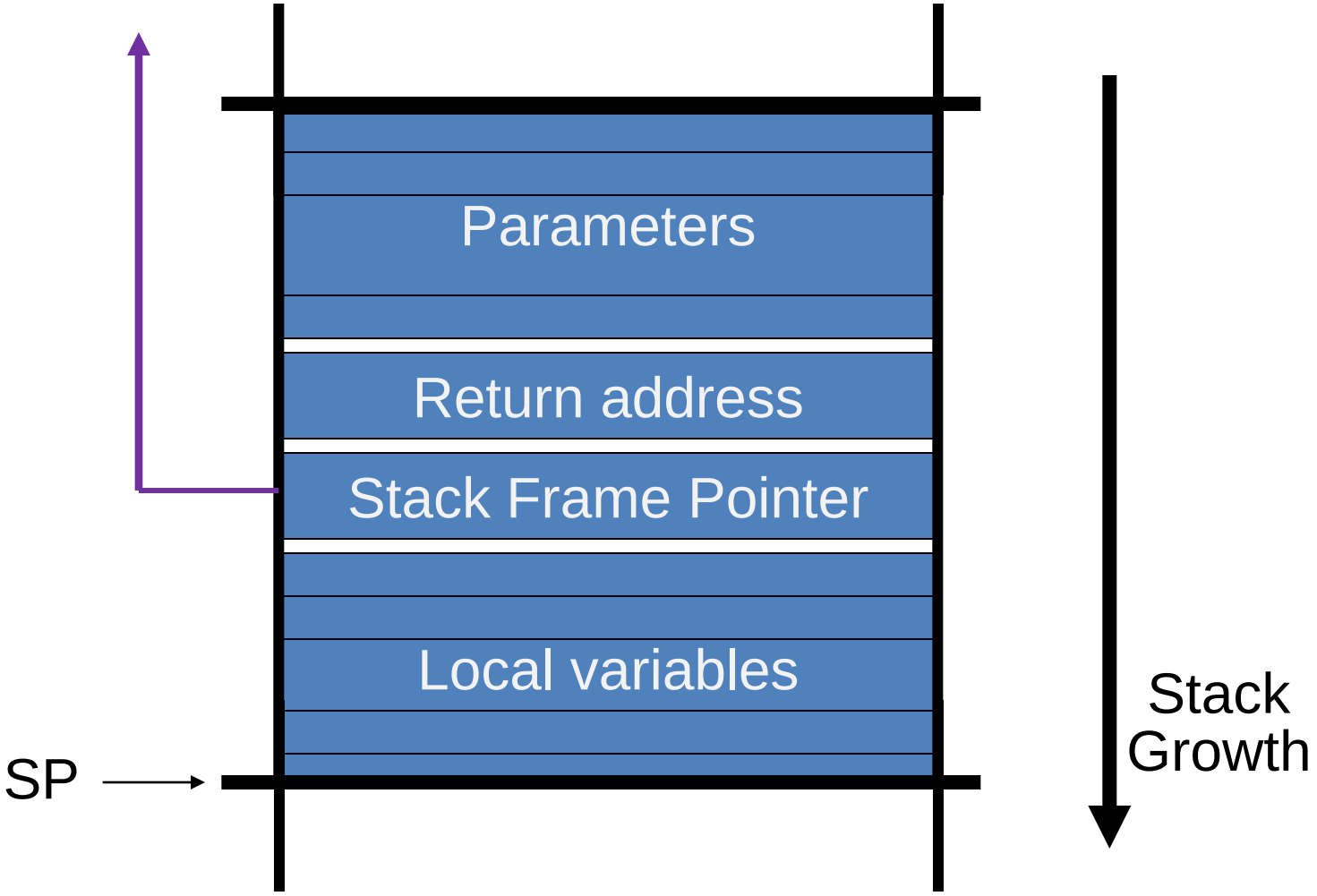
- Buffer overruns allow techniques such as stack crashing (also known as stack smashing)
  - a malicious user employs a buffer overrun to place carefully coded bytes into a stack frame in order to force the privileged program to execute arbitrary code.

Buffer Overflow

Linux process memory layout



Stack Frame


Parameters
Return address
Stack Frame Pointer
Local variables
SP
Stack
Growth

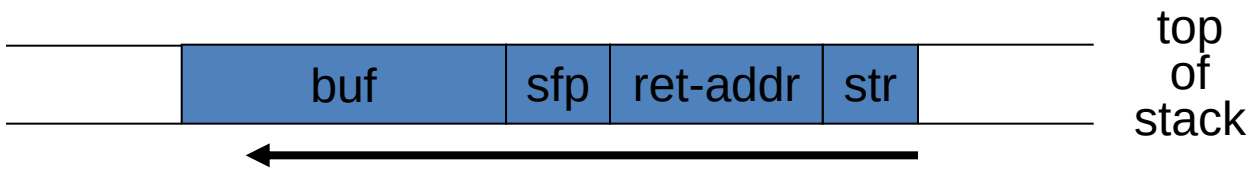
What are buffer overflows?

innovate achieve lead

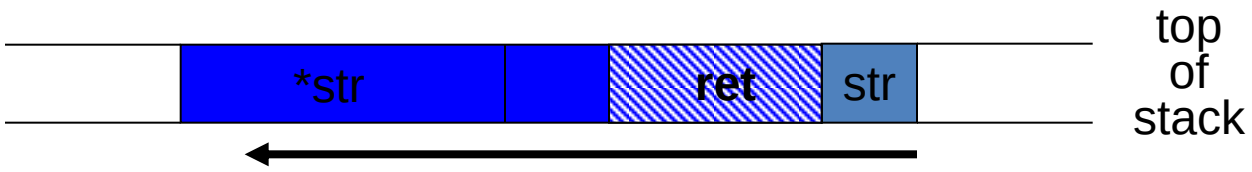
- Suppose a web server contains a function:

```
1  void func(char *str) {
2      char buf[128];
3      strcpy(buf, str);
4      do-something(buf);
5  }
```

- When the function is invoked the stack looks like:



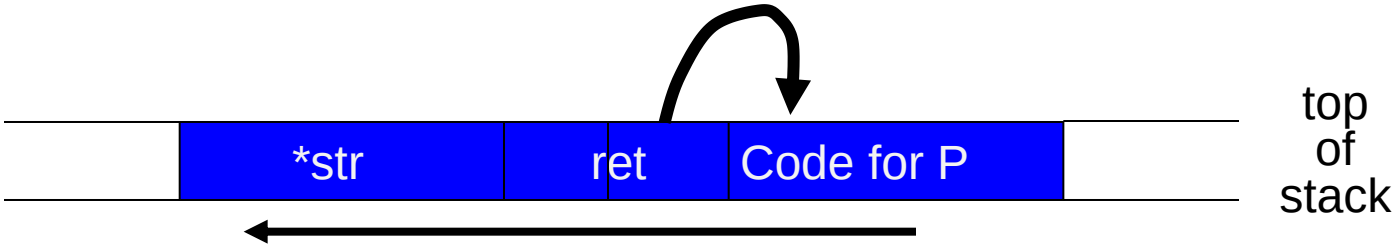
- What if `*str` is 136 bytes long? After `strcpy`:



Basic stack exploit

innovate achieve lead

- Problem: no range checking in strcpy().
- Suppose *str is such that after strcpy stack looks like:



Program P: `exec( "/bin/sh" )`

(exact shell code by Aleph One)

- When func() exits, the user will be given a shell !
- Note: attack code runs *in stack*.
- To determine ret guess position of stack when func() is called

Many unsafe C lib functions

strcpy (char *dest, const char *src)

strcat (char *dest, const char *src)

gets (char *s)

scanf (const char *format, …)

- **There are safe alternatives to many of the functions mentioned above—for example, snprintf(), strncpy(), and strncat()—that allow the caller to specify the maximum number of characters that should be copied.**
  - “Safe” versions strncpy(), strncat() are misleading
  - strncpy() may leave buffer unterminated.
  - May pad null bytes

Buffer Overflow Solutions

- Linux implements address-space randomization.
  - This technique randomly varies the location of the stack over an 8 MB range at the top of virtual memory.
- More recent x86-32 architectures provide hardware support for marking page tables as NX (“no execute”).
  - This feature is used to prevent execution of program code on the stack, thus making stack crashing more difficult.

Beware of Denial-of-Service Attacks

- The server should perform load throttling, dropping requests when the load exceeds some predetermined limit.
- A server should employ timeouts for communication with a client
  - if the client (perhaps deliberately) doesn't respond, the server is not tied up indefinitely waiting on the client.
- In the event of an overload, the server should log suitable messages so that the system administrator is notified of the problem.
- Bounds checking should be rigorously per formed to ensure that excessive requests don't overflow a data structure.

Check Return Statuses and Fail Safely

innovate achieve lead

- A privileged program should always check to see whether system calls and library functions succeed, and whether they return expected values.

Acknowledgements

Q&A

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You