CS 547: Foundation of Computer Security

S. Tripathy
IIT Patna

Previous Class

- Program security
  - Motivation and background

This Class

- Program security
- Buffer Overflow
  - Defense
- Incomplete Mediation
- TOCTTOU

Types of Program Flaws

Taxonomy of pgm flaws:

- Intentional
  - Malicious
  - Nonmalicious
- Inadvertent
  - Validation error (incomplete or inconsistent)
    - e.g., incomplete or inconsistent input data
  - Domain error
    - e.g., using a variable value outside of its domain
  - Serialization and aliasing
    - serialization – e.g., in DBMSs or OSs
    - aliasing - one variable or some reference, when changed, has an indirect (usually unexpected) effect on some other data
  - Inadequate ID and authentication
  - Boundary condition violation, etc.

Unintentional program errors

- Most security flaws are caused by unintentional program errors

- will look at some of the most common sources of unintentional security flaws
  - Buffer overflows
  - Incomplete mediation
  - TOCTTOU errors (race conditions)

Buffer Overflow/Buffer Overrun

- A buffer overflow, also known as a buffer overrun, is defined in the NIST:

"A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system."

- The single most common exploited type of security flaw

  Simple example:

- `#define LINELEN 120`
- `char buffer[LINELEN];`
- `gets(buffer);` ***or***
- `strcpy(buffer, argv[1]);`

Buffer Overflow Basics

- programming error when a process attempts to store data beyond the limits of a fixed-sized buffer
- overwrites adjacent memory locations
  - locations could hold other program variables, parameters, or program control flow data
  - buffer could be located on the stack, in the heap, or in the data section of the process
- consequences:
  - corruption of program data
  - unexpected transfer of control
  - execution of code chosen by attacker
  - memory access violations

- Two type of buffer overflow
  - Stack overflow (Memory allocated in stack)
  - Heap overflow (Memory allocated in heap)

Buffer Overflow

- Buffer overflow flaw —
  - often inadvertent (=>nonmalicious) but with serious security consequences
- Many languages require buffer size declaration
  - C language statement: char sample[10];
    - Execute statement: sample[i] = 'A'; where i=10
    - Out of bounds (0-9) → buffer overflow occurs
  - Some compilers don't check for exceeding bounds
    - C does *not* perform array bounds checking.
    - Similar problem caused by pointers
      - No reasonable way to define limits for pointers

Buffer Overflow

- Where does 'A' go?
  - Depends on what is adjacent to 'sample[10]'
    - Affects user's data - overwrites user's data
    - Affects users code - changes user's instruction
    - Affects OS data - overwrites OS data
    - Affects OS code - changes OS instruction

Sample Buffer overflow incidents!

- 1988: Morris worm – took down Internet
  - Includes buffer overflow via gets() in fingerd
- 1998: University of Washington IMAP (mail) server
- 1999: RSA crypto reference implementation
  - Subverted PGP, OpenSSH, Apache's ModSSL, etc.
- 2001: Code Red worm – buffer overflow in Microsoft's Internet Information Services (IIS) 5.0
- 2003: SQL Slammer worm compromised machines running Microsoft SQL Server 2000
- ~2008: Twilight hack – unlocks Wii consoles
  - Creates an absurdly-long horse name for "The Legend of Zelda: Twilight Princess" that includes a program

Stack Buffer Overflows

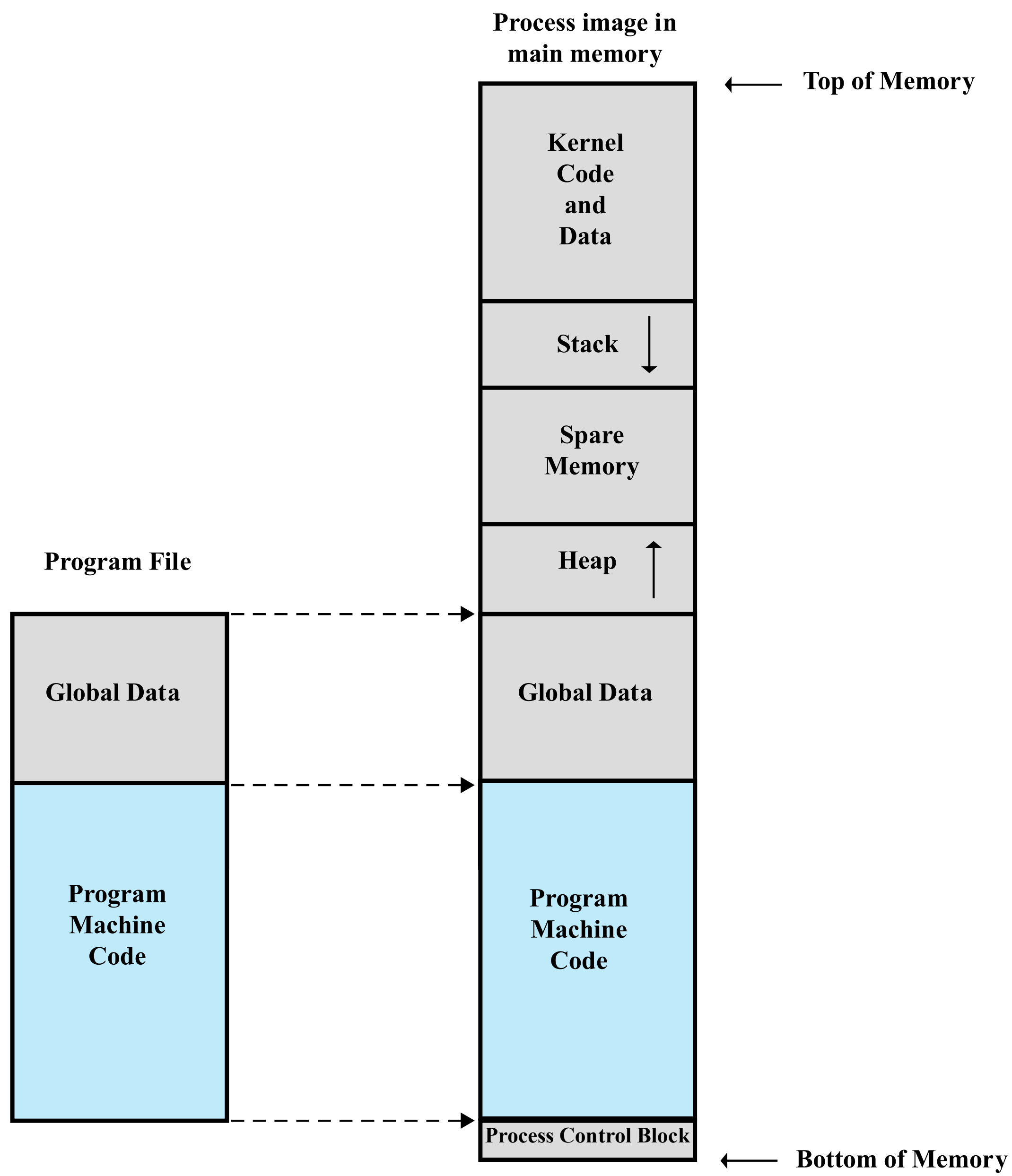
- occur when buffer is located on stack
  - also referred to as stack smashing
  - exploits included an unchecked buffer overflow
- still being widely exploited
- stack frame
  - when one function calls another it needs somewhere to save *the return address*
  - also needs locations to save the parameters to be passed in to the called function and to possibly save register values

Basic Buffer Overflow Example

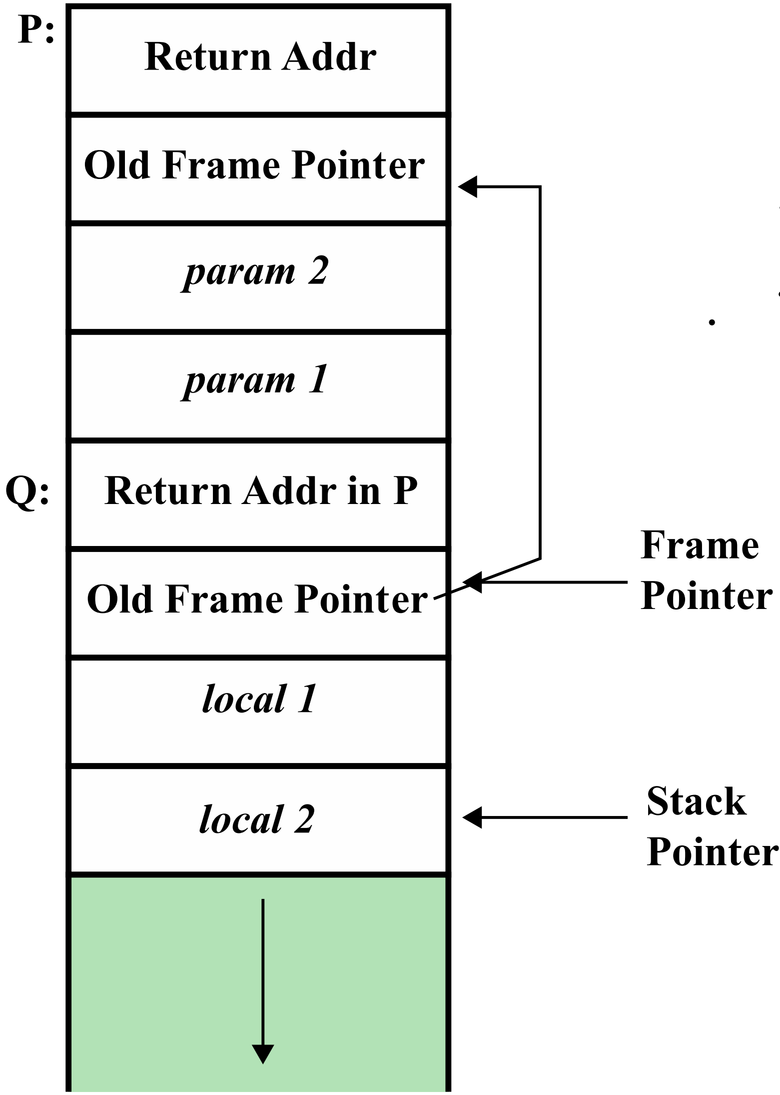
```
int main(void)
{   char buff[15];
    int pass = 0;
    printf("\n Enter the password : \n");
    gets(buff);
    if(strcmp(buff, "computersec"))
          printf ("\n Wrong Password \n");
else
    { printf ("\n Correct Password \n");
        pass = 1;
    }
    if(pass)
    { /* Now Give root or admin rights to user*/
printf ("\n Root privileges given to the user \n");
    }
    return 0;
}
```

```
/home/fac/som$./buf

 Enter the password :
computersec

 Correct Password
 Root privileges given to the user

/home/fac/som$./buf

 Enter the password :
aaaaaaaaaaaaaaaaaa

 Wrong Password

/home/fac/som$./buf

 Enter the password :
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

 Wrong Password
 Root privileges given to the user

/home/fac/som$
```

Programs and Processes



Stack Frame with Function P calls Q



- Ex.: C program:

```
void main() {
  f(1,2,3);
}
```

- The invocation of f() might generate assembly:

```
pushl $3 ; constant 3
pushl $2 ; Most C compilers push in reverse order
pushl $1
call f;  pushes instruction pointer (IP) on stack
```

  - In this case, the position in “main()” just after f(…)
  - Saved IP named the return address (RET)
  - CPU then jumps to start of “function”

Stack: Overflowing buffer

Lower-numbered addresses

Stack pointer (SP) (current top of stack)

Frame pointer (FP) – use this to access local variables & parameters

Higher-numbered addresses

Stack grows, e.g., due to procedure call

- Thanks