Buffer Overruns

Week 8

Process Layout

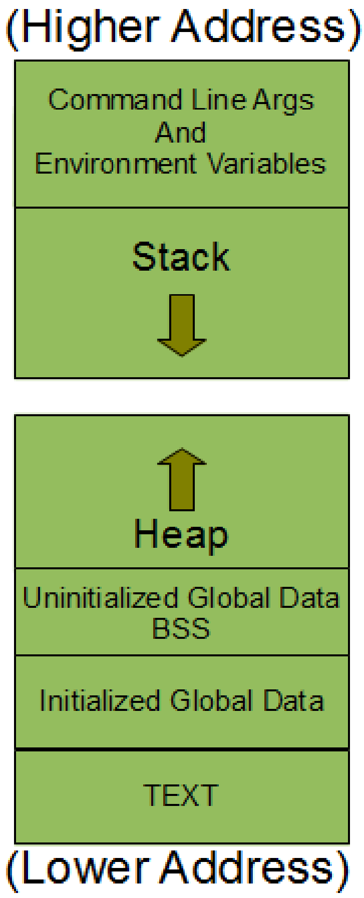

Image source : thegeekstuff.com

- TEXT segment
  - Contains machine instructions to be executed
- Global Variables
  - Initialized
  - Uninitialized
- Heap segment
  - Dynamic memory allocation
  - malloc, free
- Stack segment
  - Push frame: Function invoked
  - Pop frame: Function returned
  - Stores
    - Local variables
    - Return address, registers, etc
- Command Line arguments and Environment Variables

The Stack

- Contiguous block of memory containing data
- The stack pointer (SP) points to top of stack
- Bottom of stack is at a fixed address (FP- frame pointer)

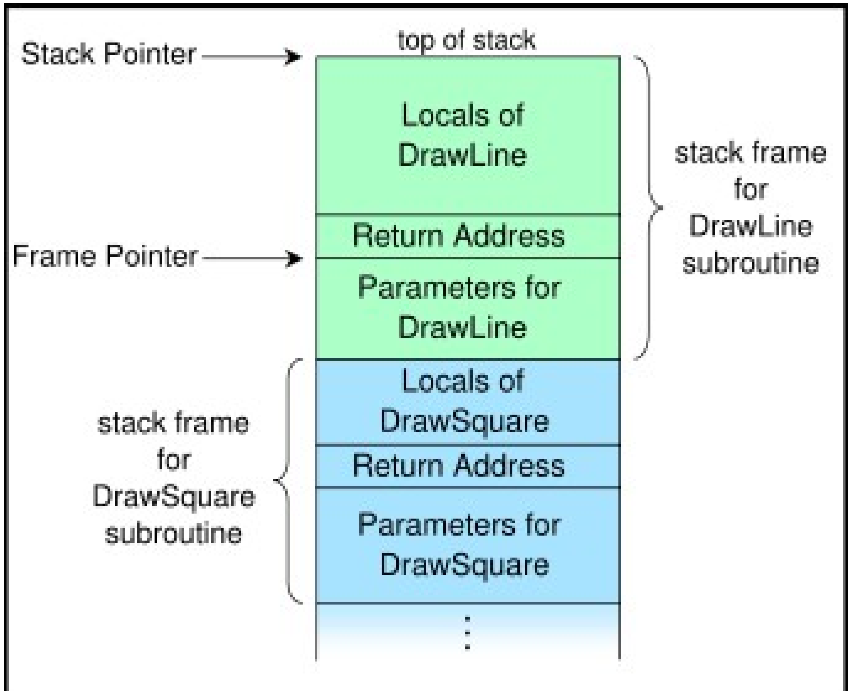

Image source : wikipedia

- ```
  DrawSquare(topleft, bottomright)
  {
          ...
          DrawLine(p1, p2);
          FillRect(topleft, bottomright, color);
          ...
  }
  ```
- ```
  DrawLine(pt1, pt2)
  {
          int local;
          float slope;
          ...
  }
  ```

The Frame

- Stack consists of logical stack frames that are pushed when calling a function and popped when returning
- Stack frame contains parameters to function, local variables, and data necessary to recover previous stack frame including the instruction pointer at the time of the function call
- Frame pointer (FP) points to fixed location within a frame and variables are referenced by offsets to the FP

Calling a Procedure

- Save previous FP
- Copies SP into FP to create the FP
- Advances SP to reserve space for the local variables

Buffer overflows

- Result of stuffing more data into a buffer than it can handle

Example

```
void function(char *str)
{
    char buffer[16];
    strcpy(buffer,str);
}
```

```
void main(int argc, char** argv)
{
     char large_string[256];
     int i;
     for( i = 0; i < 255; i++)
     {
       large_string[i] = 'A';
     }
     large_string[255] = '\0'
     function(large_string);
     printf("Hello world!\n");
}
```

```
bottom of                                                        top of
memory                                                           memory
                 buffer              sfp    ret    *str
<------         [                   ][     ][     ][     ]

top of                                                        bottom of
stack                                                             stack
```

What’s Going to Happen?

- What happened to buffer and the other regions in the stack?
- If the character ‘A’ hex value is 0x41, what is the return address?
- What happens when the function returns?

Shell Code

- Now that we can modify the return address and the flow of execution, what program to execute?
- We want to spawn a shell so we can execute anything else.
- We need to place instructions into the program’s address space

The Answer

- Place the code we are trying to execute in the buffer we are overflowing
- Overwrite the return address so it points back into the buffer

Prevention?

- Hardware?
- Software?

Laboratory

- Build thttpd 2.25b with the patch
  - Gunzip and untar it
  - Patch it
  - Configure it
  - Make it
- Run it on port (12400 to 12499)
  - ./thttpd –p <port>
- Do a simple request like
  - wget http://localhost:portnumber

Laboratory

- Crash the web server by sending it a suitably-formatted request
  - wget http://localhost:portnumber/AAAAAAAAA.....AAAA (about 7k As)
  - Where does the buffer overrun occur? Why?

Laboratory

- Run the web server under GDB and get traceback (bt) after the crash
  - ./thttpd –p <portnumber>
  - Find the pid for thttpd: ps –e | grep thttpd
  - gdb thttpd pid
  - Send your crashing request
  - Continue and when it crashes do bt
  - Include this in lab8.txt
- Describe how you would build a remote exploit in the modified thttpd
  - Smashing the stack for Fun and Profit will be helpful

Stack Protection

Here is a list of commands to generate the assembly files. You can do a compile-only (-c flag) on the thttpd.c file

StackProtection

Run these inside the "src" folder

```
gcc -m32 -c -S -o thttpd-nostackprot.s -fno-stack-protector -I . -I ../ thttpd.c
```

```
gcc -m32 -c -S -o thttpd-stackprot.s -fstack-protector -I . -I ../ thttpd.c
```

Homework

- Things to think about…
  - Significance of Damage
  - Ease of Exploitation
  - Widespread
  - Ease of repair/prevention