

Buffer Overruns

Week 7

Memory

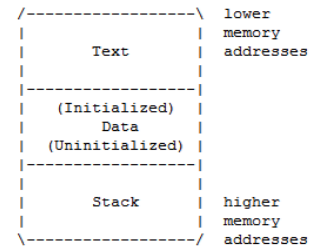


Fig. 1 Process Memory Regions

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

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)      void main()
{
    char buffer[16];
    strcpy(buffer, str);
}

                                {
                                char large_string[256];
                                int i;
                                for(i = 0; i < 255; i++)
                                    large_string[i] = 'A';
                                function(large_string);
                                }

bottom of memory                top of memory

                                buffer      sfp   ret   *str
                                <-----   ][   ][   ][   ]

top of stack                    bottom of 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 tthttpd 2.25b with the patch
 - Gunzip and untar it
 - Patch it
 - Configure it
 - Make it
- Run it on port 8080
 - ./tthttpd -p 8080
- Do a simple request like
 - wget http://localhost:8080

Laboratory

- Crash the web server by sending it a suitably-formatted request
 - `wget http://localhost:8080/?111111111...1`
 - Where does the buffer overrun occur? Why?

Laboratory

- Run the web server under GDB and get traceback (bt) after the crash
 - `./thttpd -p 8080`
 - 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 lab7.txt
- Describe how you would build a remote exploit in the modified thttpd
 - Smashing the stack for Fun and Profit will be helpful

Homework

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