Breaking In

- SQL Injection and other application attacks can get you data, which has value.
- System control can get you data, plus it can provide an anonymous platform for further attacks.
-

Systemic Attacks

- Temporary attacks get command control for a brief period of time and can be used to get more control, to obtain data or simply to wreak havoc.
- Permanent attacks install software on the system that allow access over a long period of time – typically called a rootkit.

Attack Vectors

- Buffer overflows and related exploits.
- Worms and their slithery ilk.
- Password attacks (including XSS, phishing and the like).
- Spoofing and pharming.
- Security failures like broken encryption algorithms, plaintext transmission.
- Trojans and viruses.

A Scenario

I want to break into a particular system for the purpose of gaining long term control of it. Currently, all I know is that it is named bozo.clowns.org.

What are my choices?

HowTo?

- Trojan - I would need someone dumb enough to either place my malware on the system or I would need to find a way to get it installed.
- Virus – not bad. I just need to send an email that has some executable code that someone will open. Or an Office macro.
- Worm - I need to figure a way to get it in.

HowTo?

- Direct password attack – only if their password security is very poor. Check it out.
- An indirect attack like spoofing, pharming, XSS, phishing. This would not work well unless there is a significant group of remote users. Hopefully some that are careless.
- Security failures – difficult from the outside.

HowTo?

- Buffer overflow – possible, but difficult. It requires some information about the applications running on the system and the structure of those applications.
- Other application-level vulnerabilities – as in buffer overflow.

Lets Find Out

- The question is “what external facing applications is it running?”.
  - Fingerprinting – determine the type of the system; Windows, Linux , ...
  - Port Scanning – determine what applications are running at what ports

Fingerprinting

- By looking at the response of the system to certain types of TCP/IP requests you can determine the OS.
- Web Server types, SSH signature, SMB response etc.
- You can even determine things like the patch level.
- Nessus is an example of a tool that has good fingerprinting utilities.

Port Scanning

- External facing applications typically use a well-known port. For example, ssh (22), smtp (25), web server (80), mysql (3306), gnutella (6346), Windows Messenger (2001-2120).
- A port scanner tries to connect to each port in turn and read the signature of the application by the response.

Port Scanning

- You can do this manually within reason.
  - telnet bozo.clowns.org 25 will try to connect to whatever is running on port 25. If its the expected mail server, you should see:
  -

Connected to esus.cs.montana.edu (153.90.199.47).
Escape character is '^]'.
220 esus.cs.montana.edu ESMTP Sendmail 8.13.1/8.13.1; Tue, 6 Nov 2007 09:58:52 -0700

Port Scanning

- Because it can provide so much useful information, there are tools to prevent it.
- Firewalls can block ports from all but authorized systems.
- Tools like portsentry will watch port access and try to detect malicious behavior. This has led to port scanners that evade detection and the so clever “port knocking”.

Finding an Attack Vector

- We have a list of applications running. Which one to attack?
- cve.mitre.org – cve is the Common Vulnerabilities and Exposures index.
- It lists all known vulnerabilities, often with links to exploit code. Even better, it has a list of candidates. 34 on 11/5.
- bugtraq often has good stuff.
- Zero-day (or zero-hour) exploits

Picking an Attack Vector

- Something that matches the application and your access to it and/or the people.
- Find out if someone has already written an exploit for it.
- There are, unfortunately, sites that provide this information.

Buffer Overflow

- Assume there is nothing that fits, but you have discovered that the system is running an application named daisy that you think might be vulnerable.
- It runs with an external interface with lots of user inputs, so a buffer overflow vulnerability might exist.
- We find that daisy is written in C.

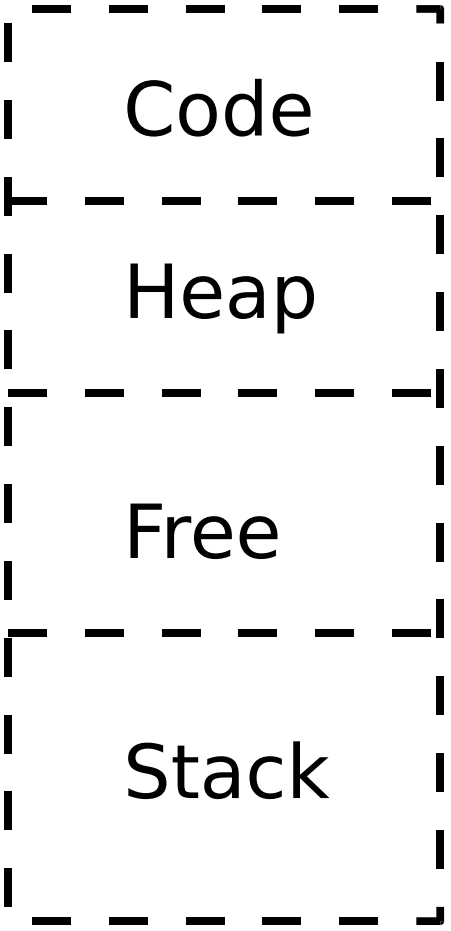
Finding Buffer Overflow

- We are looking for inputs that give us a systemic error when we type in a really long string.
- If the application tests the string for length, you should get an application error.
- For a console app, try little-used options to see if you get something interesting.

Still Looking

- We are looking for inputs that give us a systemic error when we type in a really long string.
- If the application tests the string for length, you should get an application error.
- For a console app, try little-used options to see if you get something interesting.

Memory Organization

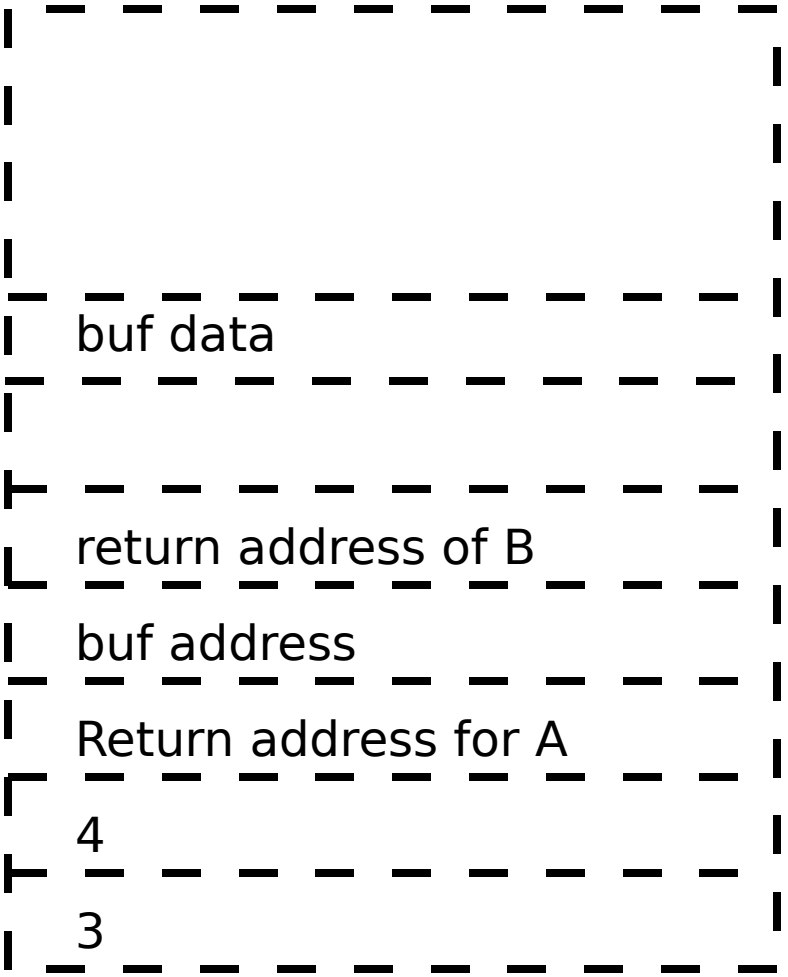


Memory Layout

```
int main ()
{
   char buf [4];
   A(3,4);
   B(buf);
}
int A (int a, int b)
{
   return a_b;
}
void B (char *c)
{
   *(c+1) = 'a';
   return;
}
```

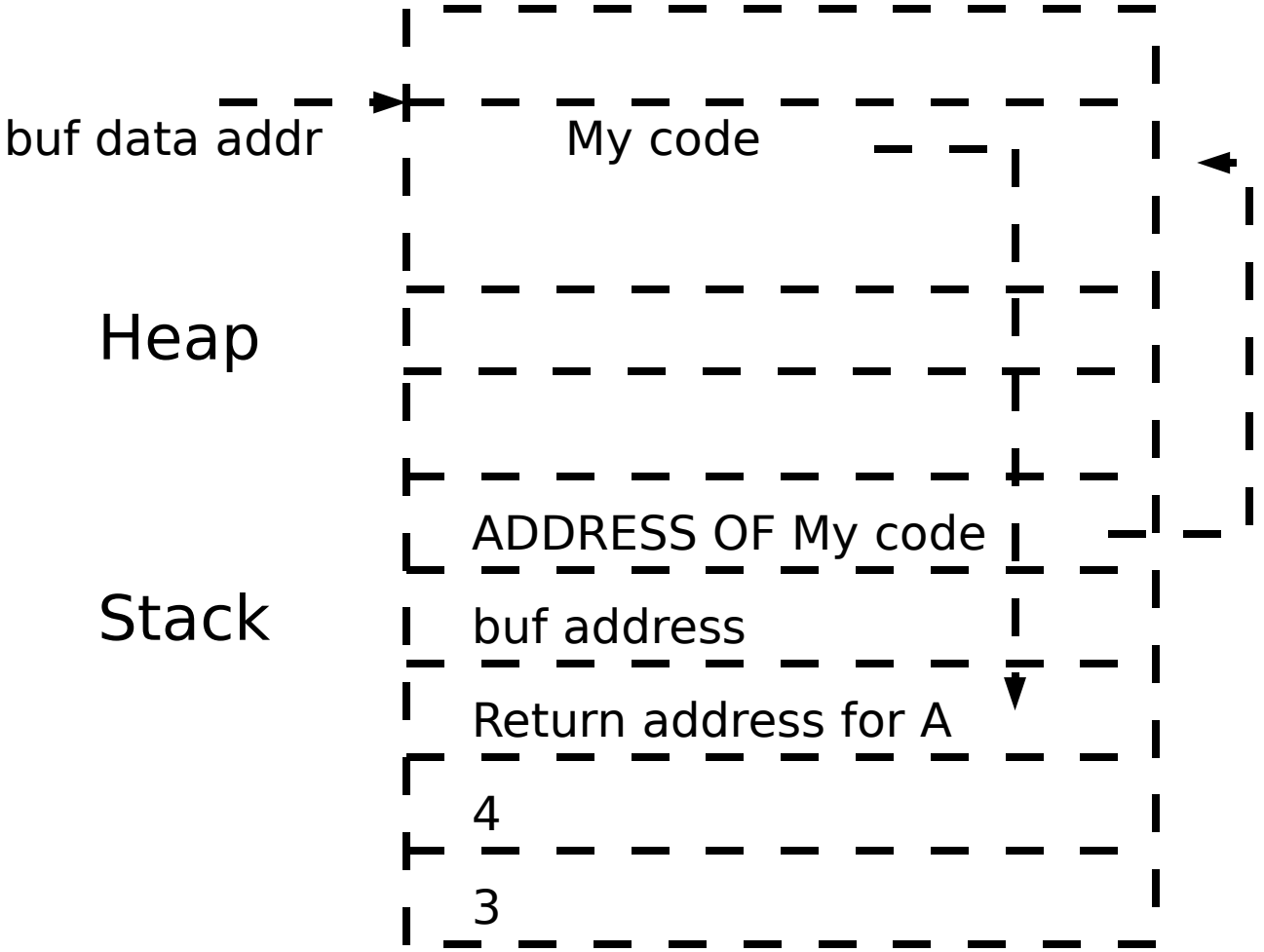
Heap

Stack



The Exploit

```
int main ()
{
   char buf [4];
   A(3,4);
   B(buf);
}
int A (int a, int b)
{
   return a_b;
}
void B (char *c)
{
   *(c+1) = 'a';
   return;
}
```



How!!!

- If we have the code in hand, we can look at a dynamic dump and figure it out exactly.
- If not, we can look at the memory map, reduce the possibilities and experiment.
- daisy is doomed!

The Memory Map

- If we have the code in hand, we can look at the assembled machine code and figure it out.
- If not, we can look at a disassembly, reduce the possibilities and experiment.
- daisy is doomed!

A Simple Example

```
#include <stdio.h>
 int main()
{
   ReadMe();
}
int ReadMe()
{
   char Buffer[4];
   scanf(“%s”, Buffer);
   printf(“%s”,s);
}
```

Main Disassembly

```
0x80483f8 <main>:        push   %ebp
0x80483f9 <main+1>:      mov    %esp,%ebp
0x80483fb <main+3>:      call   0x8048404 <ReadMe>
0x8048400 <main+8>:      leave
0x8048401 <main+9>:      ret
0x8048402 <main+10>:     mov    %esi,%esi
End of assembler dump.
```

ReadMe Disassembly

```
0x8048404 <ReadMe>:    push   %ebp
0x8048405 <ReadMe+1>:   mov    %esp,%ebp
0x8048407 <ReadMe+3>:   sub    $0x4,%esp
0x804840a <ReadMe+6>:   lea    0xfffffffc(%ebp),%eax
0x804840d <ReadMe+9>:   push   %eax
0x804840e <ReadMe+10>:  push   $0x8048480
0x8048413 <ReadMe+15>:  call   0x804830c <scanf>
0x8048418 <ReadMe+20>:  add    $0x8,%esp
0x804841b <ReadMe+23>:  lea    0xfffffffc(%ebp),%eax
0x804841e <ReadMe+26>:  push   %eax
0x804841f <ReadMe+27>:  push   $0x8048480
0x8048424 <ReadMe+32>:  call   0x804833c <printf>
0x8048429 <ReadMe+37>:  add    $0x8,%esp
0x804842c <ReadMe+40>:  leave
0x804842d <ReadMe+41>:  ret
0x804842e <ReadMe+42>:  nop
0x804842f <ReadMe+43>:  nop
```

Test It

Note that the ebp register is the frame pointer which is pushed on the stack and it is pop'ed off the stack on return.

If we run the code and enter AAA, it prints AAA. If we type in AAAAAAAA0000, it halts with a segmentation fault.

Memory

Buffer	EBP	Return Addr	EAX
AAAA	0x8048408	0x8048400	EAX
AAAA	0x41414141	0x00000000	EAX

Whoa!!

We can make these anything we want. If we have root privilege, we can jump into the kernel. We can do any darn thing we want, as long as the user running daisy has the right access privileges.

The Error

eax	0x7	7
ecx	0x40071fd0	1074208720
edx	0x4010a980	1074833792
ebx	0x4010c1ec	1074840044
esp	0xbffffb58	-1073743016
ebp	0x414141	4276545

We can't return to this address, hence the seg fault.

Exploiting the Exploit

Write up a little code to do something.

…
system (“acctcreate -pmagic backdoor”);
rm (“/var/log/syslog”);
system (“echo ha ha > /etc/motd”);
…

Compile it to get the binary code and go.