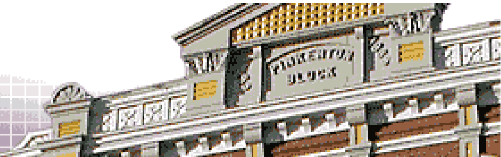
Network Security: Buffer Overflow Attacks

Joe McCarthy



Today’s Agenda

- What is Network Security?
- Why should you care?
- What is a network security attack?
- What is a buffer overflow attack?
- Where can you learn more?

All in 30 minutes …

What is Network Security?

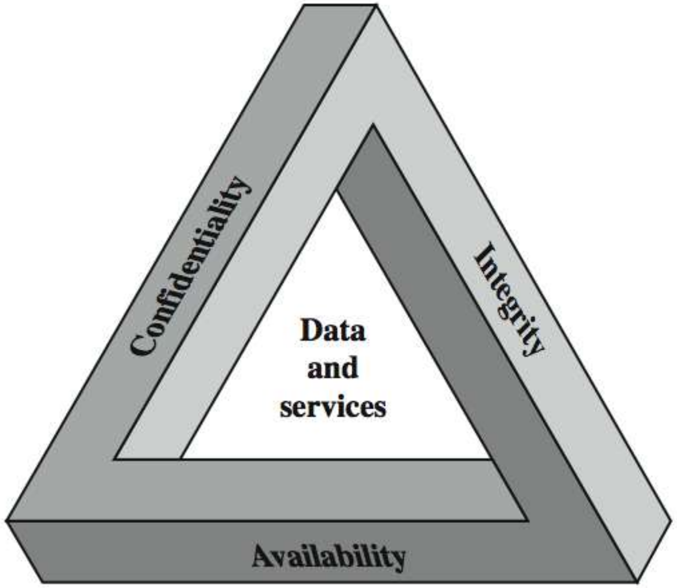
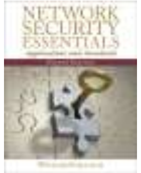


Figure 1.1 The Security Requirements Triad



Network Security Essentials, 4/E
William Stallings
Prentice Hall, 2011

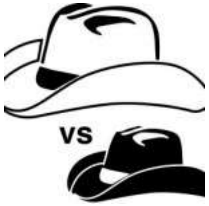
NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Computer Security
The protection afforded to an automated information system in order to attain the applicable objectives of preserving the ***integrity***, ***availability*** and ***confidentiality*** of information system resources (includes hardware, software, firmware, information/dat a, and telecommunications)

February 2004
http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf

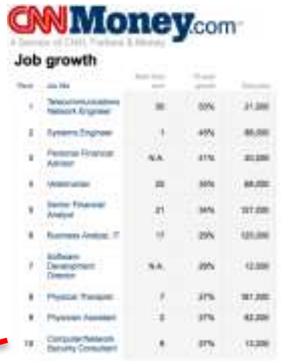
Why study Network Security?

- Multi-disciplinary
  - Computer science, mathematics, psychology, sociology, politics, ethics, economics, forensics, …
- New way of thinking: security mind set
  - Preventing undesirable behavior vs. enabling desirable behavior



- Personal relevance
  - Keeping your personal data & devices safe
- Professional relevance

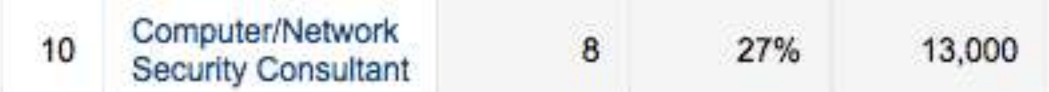
GAWKER



CNNMoney.com

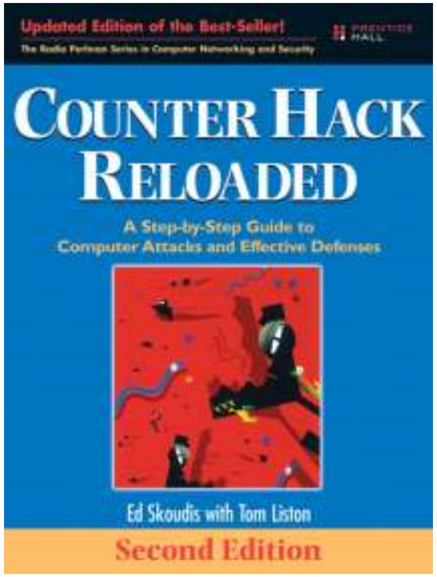
Job growth

Rank	Job title	[illegible]	[illegible]	[illegible]
1	Telecommunications Network Engineer	30	[illegible]	[illegible]
2	Systems Engineer	1	45%	[illegible]
3	Personal Financial Advisor	N.A.	41%	[illegible]
4	Veterinarian	22	[illegible]	[illegible]
5	Senior Financial Analyst	21	[illegible]	[illegible]
6	Business Analyst, IT	17	29%	[illegible]
7	Software Development Director	N.A.	[illegible]	[illegible]
8	Physical Therapist	7	27%	[illegible]
9	Physician Assistant	2	27%	[illegible]
10	Computer/Network Security Consultant	8	27%	[illegible]

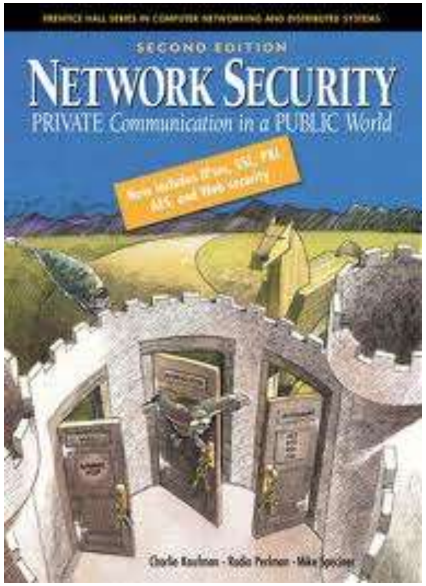


10	Computer/Network Security Consultant	8	27%	13,000

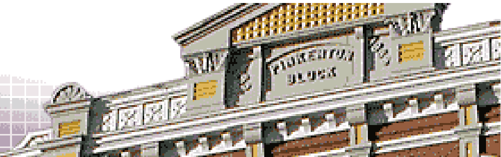
TCSS 431: Network Security


Updated Edition of the Best-Seller!
Prentice Hall
Counter Hack Reloaded
A Step-by-Step Guide to Computer Attacks and Effective Defenses
Ed Skoudis with Tom Liston
Second Edition

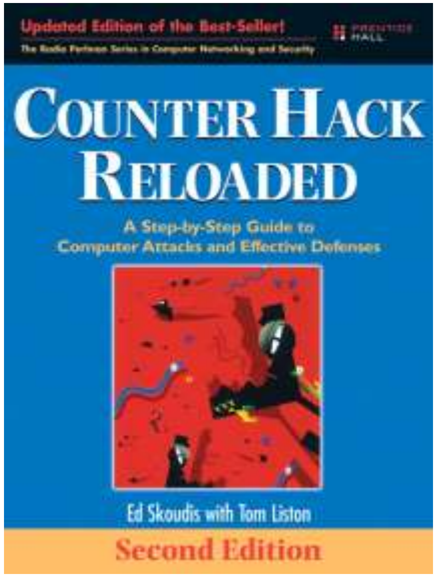
Counter Hack Reloaded:
A Step-by-Step Guide
to Computer Attacks and
Effective Defenses, 2/E
Ed Skoudis
Tom Liston
Prentice Hall, 2006


Second Edition
Network Security
Private Communication in a Public World
Charlie Kaufman · Radia Perlman · Mike Speciner

Network Security:
Private Communication
in a Public World, 2/E
Charlie Kaufman
Radia Perlman
Mike Speciner
Prentice Hall, 2002

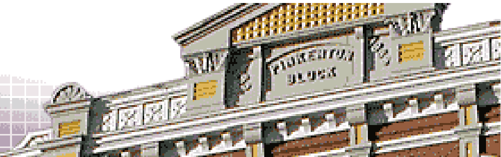


Today’s Agenda


Updated Edition of the Best-Seller!
PRENTICE HALL
The Radia Perlman Series in Computer Networking and Security
COUNTER HACK RELOADED
A Step-by-Step Guide to Computer Attacks and Effective Defenses
Ed Skoudis with Tom Liston
Second Edition

Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses, 2/E
Skoudis& Liston
Prentice Hall, 2006

1. Introduction
2. Networking Overview
3. Linux and UNIX Overview
4. Windows NT/000/XP/00 Overview
5. Phase 1: Reconnaissance
6. Phase 2: Scanning
7. **Phase 3: Gaining Access Using Application& OS Attacks**
    - Script Kiddie Exploit Trolling
    - Pragmatism for More Sophisticated Attackers
    - ***Buffer Overflow Exploits***
    - Password Attacks
    - Web Application Attacks
    - Exploiting Browser Flaws
8. Phase 4: Gaining Access Using Network Attacks
9. Phase 4: Denial-of-Service Attacks
10. Phase 4: Maintaining Access: Trojans, Backdoors&Rootkits
11. Phase 5: Covering Tracks& Hiding
12. Putting It All Together: Anatomy of an Attack
13. The Future, References & Conclusions



Anatomy of an Attack

- Reconnaissance
  - “casing the joint”
  - Discovery of physical & online sensitive information
    - Names, contact info (phone, email), IP addresses
  - Social engineering, dumpster diving, Google
- Scanning
  - “trying doorknobs & windows”
  - Search for openings, network topology, OS type(s)
    - Wireless access points, TCP ports, routers, gateways
  - Inventory of target system& possible vulnerabilities
- Gaining access
  - “breaking in”
  - Application & OS attacks (Chapter 7)
    - Stack-based & Heap-based Buffer Overflow Attacks

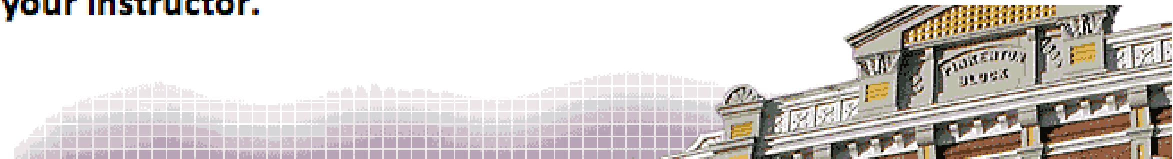
Network Security Class Student Agreement

- I understand that I am taking the TCSS431 Network Security class at the University of Washington, Tacoma (UWT) in which I will learn about computer access techniques that can be used to break in to, damage or otherwise alter ("hack") computer systems. I also understand that it is the purpose of the class that this knowledge be used to protect information resources and not to compromise or destroy them or otherwise break any laws or disrupt educational, commercial or other activities. Any access to a system without the administrator or owner's permission is illegal.

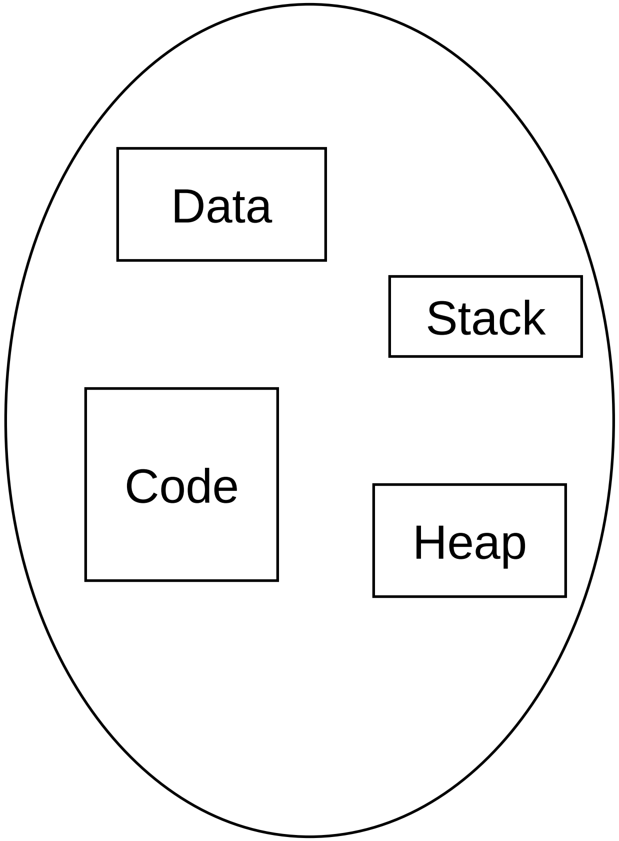
- The following actions are clearly not ethical:
  - Breaking into a computer system without the permission of the owner or administrator of that computer system.
  - Doing anything that substantially interferes with other user's access to computer-based services (i. e., denial of service attacks).
  - Accessing computer-based information without appropriate authorization.
  - Accessing any computer-based service without appropriate authorization.
  - Unauthorized monitoring of electronic communication.

- I agree that I will not access, damage or disrupt any computer systems or other students' work during this class. I also understand that I will be expected to work with other students to test security, but I agree that it will always be done with their knowledge. In addition I will not destroy or damage their work and will let them know what I have accessed on their computer system. I will cease accessing their system when asked.

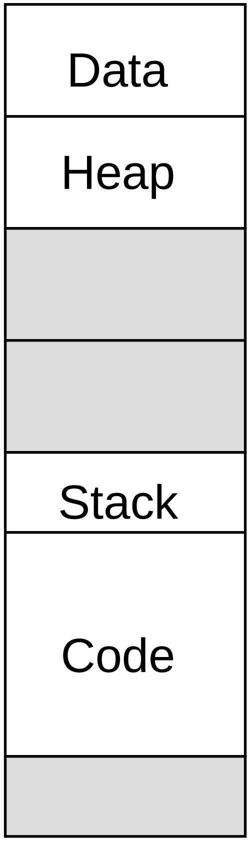
- **When in doubt, ask your instructor.**



Brief review of Main Memory



user view of memory

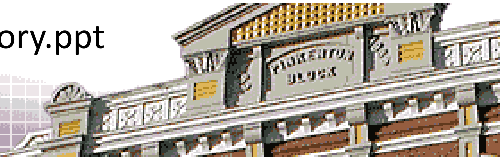


logical memory space

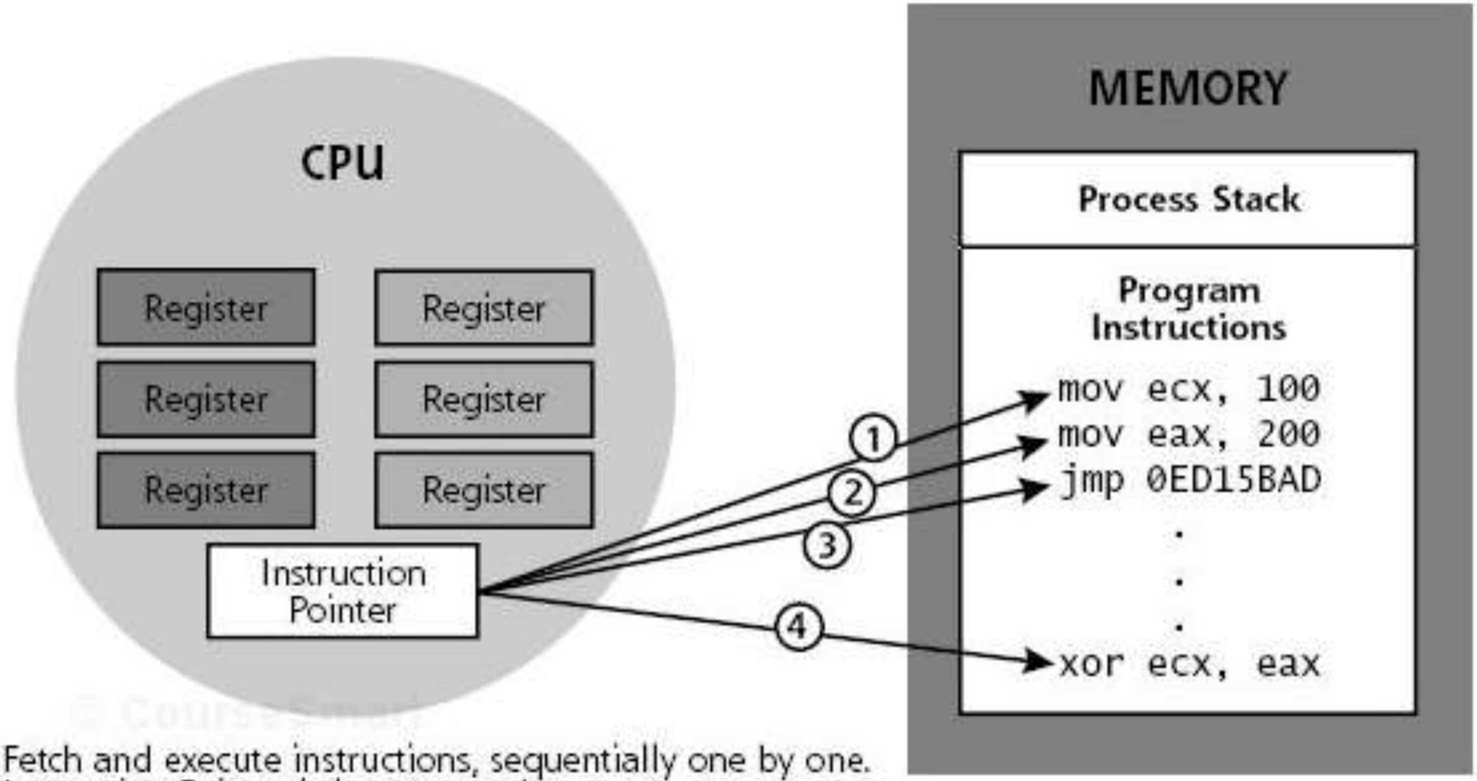
- Each page is only a piece of memory but has no meaning.
- A program is a collection of segments such as:
  - main program,
  - procedure,
  - function,
  - global variables,
  - common block,
  - stack,
  - symbol table

UNIVERSITY of WASHINGTON TACOMA

http://courses.washington.edu/css430/ppt/Memory.ppt



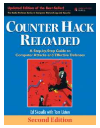
Stack-based Buffer Overflow Attacks



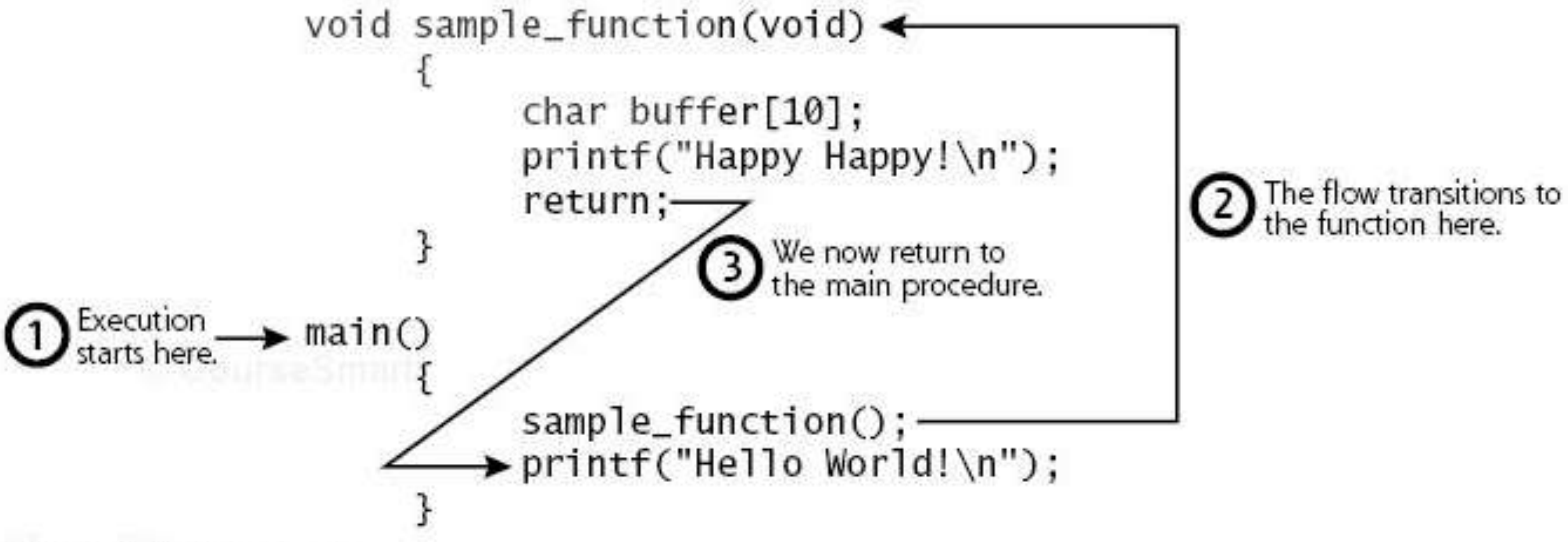
```
mov ecx, 100
mov eax, 200
jmp 0ED15BAD
.
.
.
xor ecx, eax
```

Fetch and execute instructions, sequentially one by one.
Instruction Pointer is incremented.
At Jump, Instruction Pointer is altered to begin fetching instructions in a different location.

Figure 7.2 How programs run.



Stack-based Buffer Overflow Attacks



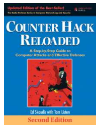
```
void sample_function(void)
     {
          char buffer[10];
          printf("Happy Happy!\n");
          return;
     }
main()
     {
          sample_function();
          printf("Hello World!\n");
     }
```

1 Execution starts here.

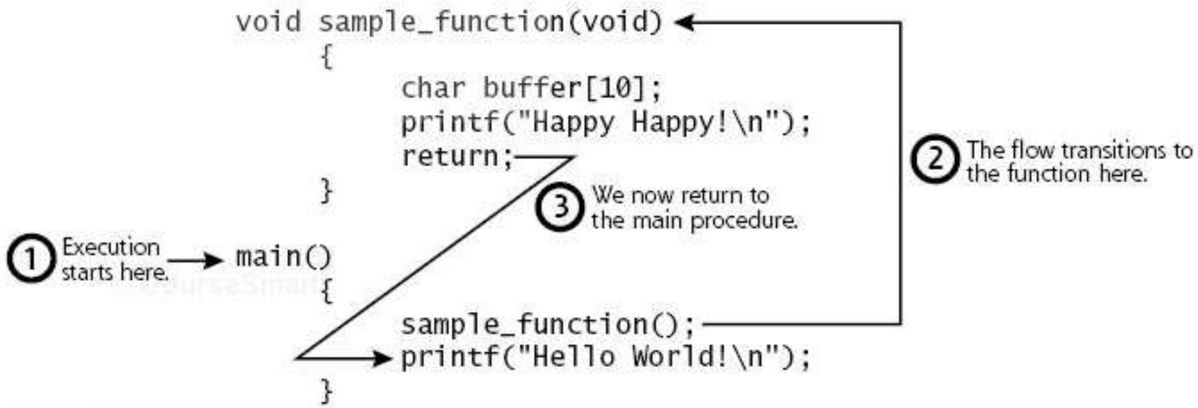
2 The flow transitions to the function here.

3 We now return to the main procedure.

Figure 7.3 Some C code.



Stack-based Buffer Overflow Attacks



```
void sample_function(void)
        {
                char buffer[10];
                printf("Happy Happy!\n");
                return;
        }
main()
        {
                sample_function();
                printf("Hello World!\n");
        }
```

① Execution starts here.

② The flow transitions to the function here.

③ We now return to the main procedure.

Figure 7.3 Some C code.

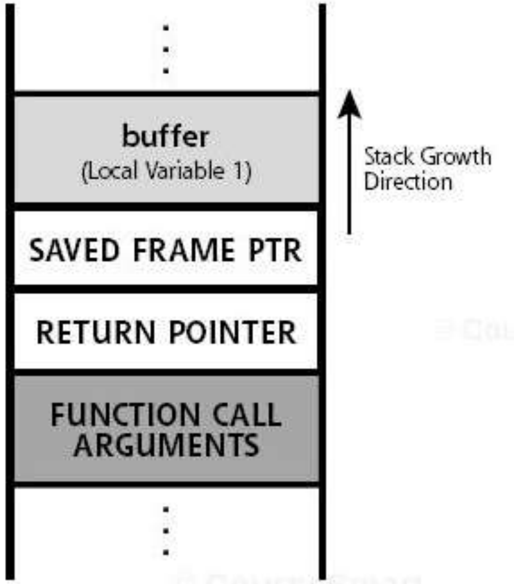
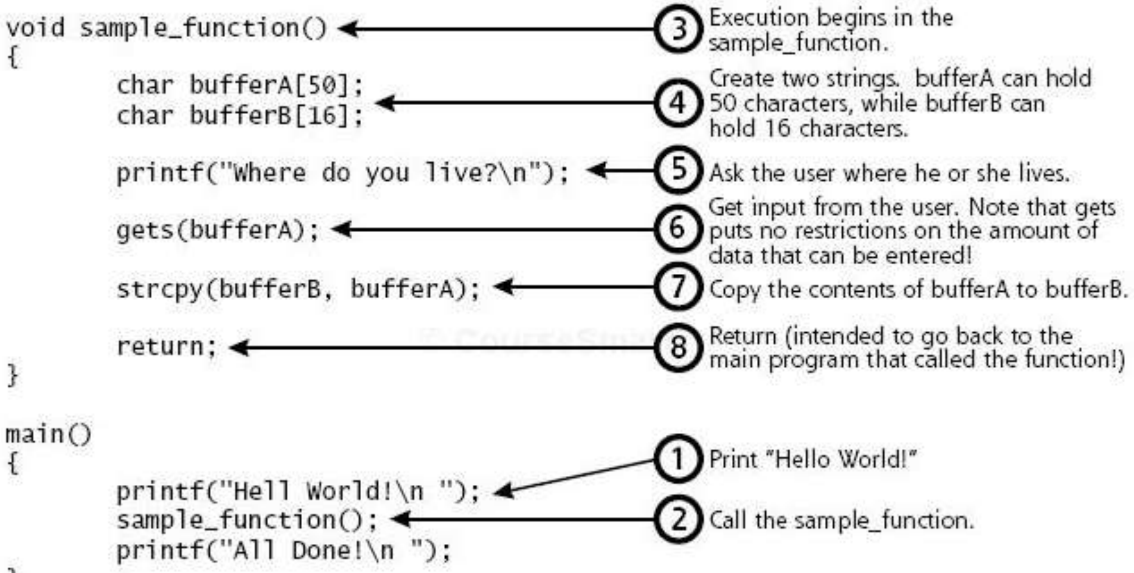


Figure 7.4 A normal stack.

Stack-based Buffer Overflow Attacks



```
void sample_function()
{
        char bufferA[50];
        char bufferB[16];

        printf("Where do you live?\n");

        gets(bufferA);

        strcpy(bufferB, bufferA);

        return;
}

main()
{
        printf("Hell World!\n ");
        sample_function();
        printf("All Done!\n ");
}
```

3 Execution begins in the sample_function.

4 Create two strings. bufferA can hold 50 characters, while bufferB can hold 16 characters.

5 Ask the user where he or she lives.

6 Get input from the user. Note that gets puts no restrictions on the amount of data that can be entered!

7 Copy the contents of bufferA to bufferB.

8 Return (intended to go back to the main program that called the function!)

1 Print "Hello World!"

2 Call the sample_function.

Figure 7.5 Some very vulnerable C code.

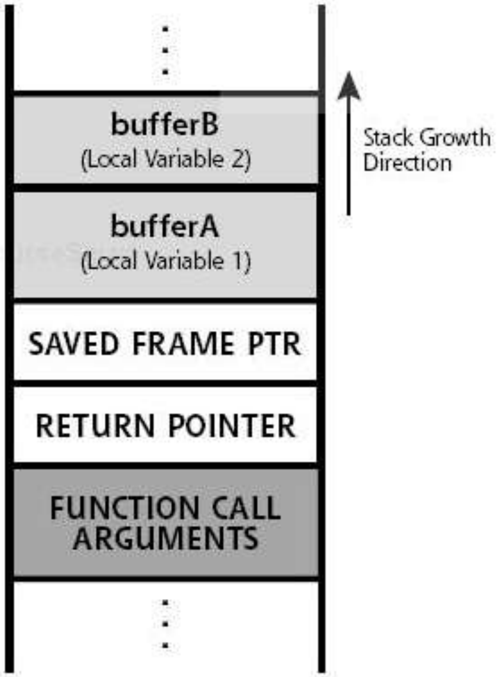
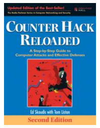
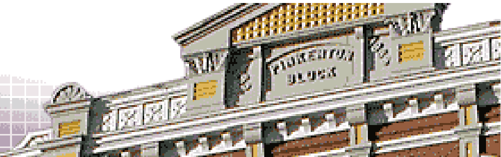


Figure 7.6 A view of the stack of the vulnerable program.





Stack-based Buffer Overflow Attacks

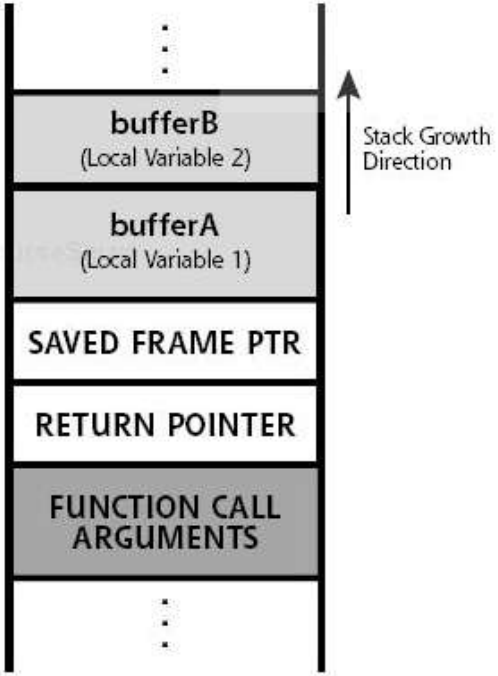


Figure 7.6 A view of the stack of the vulnerable program.

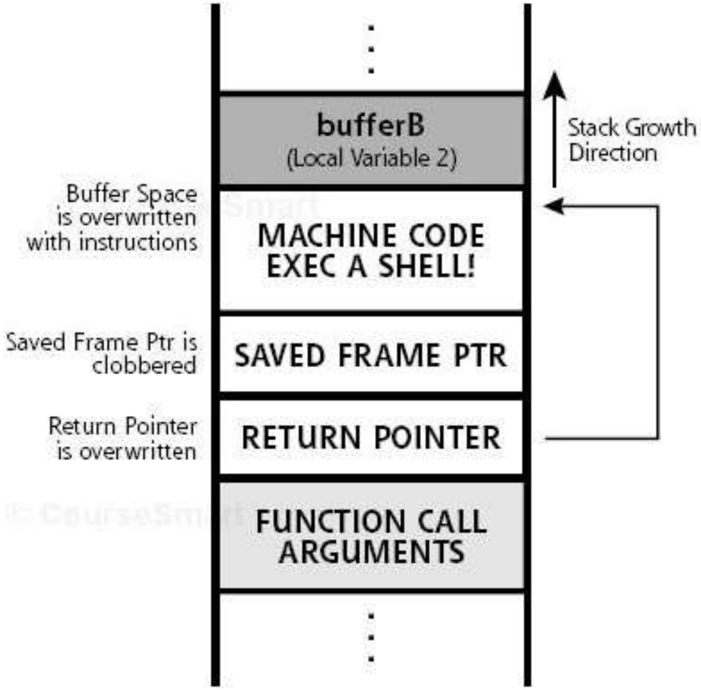
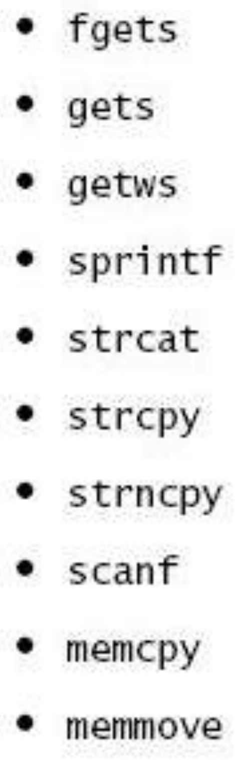

bufferB
(Local Variable 2)
Buffer Space is overwritten with instructions
MACHINE CODE EXEC A SHELL!
Saved Frame Ptr is clobbered
SAVED FRAME PTR
Return Pointer is overwritten
RETURN POINTER
FUNCTION CALL ARGUMENTS
Stack Growth Direction

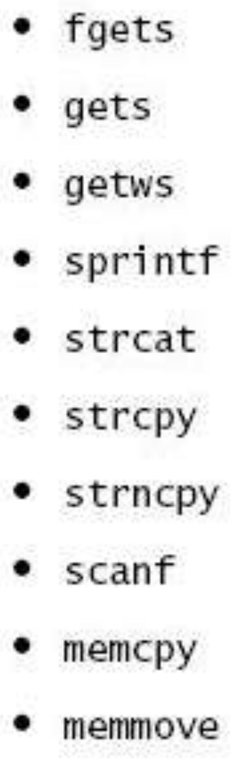
Figure 7.7 A smashed stack.

C library functions considered harmful

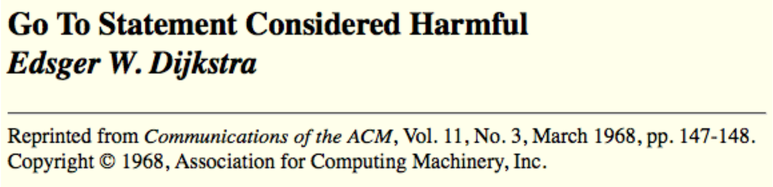


- fgets
- gets
- getws
- sprintf
- strcat
- strcpy
- strncpy
- scanf
- memcpy
- memmove

C library functions considered harmful



- fgets
- gets
- getws
- sprintf
- strcat
- strcpy
- strncpy
- scanf
- memcpy
- memmove

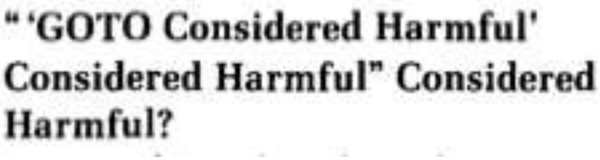


Go To Statement Considered Harmful
Edsger W. Dijkstra

Reprinted from *Communications of the ACM*, Vol. 11, No. 3, March 1968, pp. 147-148.
Copyright © 1968, Association for Computing Machinery, Inc.

"GOTO Considered Harmful" Considered Harmful

Frank Rubin.
(March 1987)
Communications of the ACM
30 (3): 195–196.

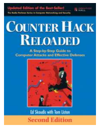


" 'GOTO Considered Harmful' Considered Harmful" Considered Harmful?

Donald Moore, Chuck Musciano, Michael J. Liebhaber, Steven F. Lott and Lee Starr.
(May 1987)
Communications of the ACM
30 (5): 351–355.

http://en.wikipedia.org/wiki/Considered_harmful

UNIVERSITY of WASHINGTON TACOMA

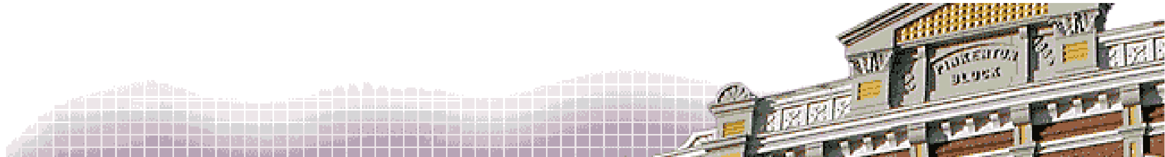


Finding stack-based buffer overflow vulnerabilities

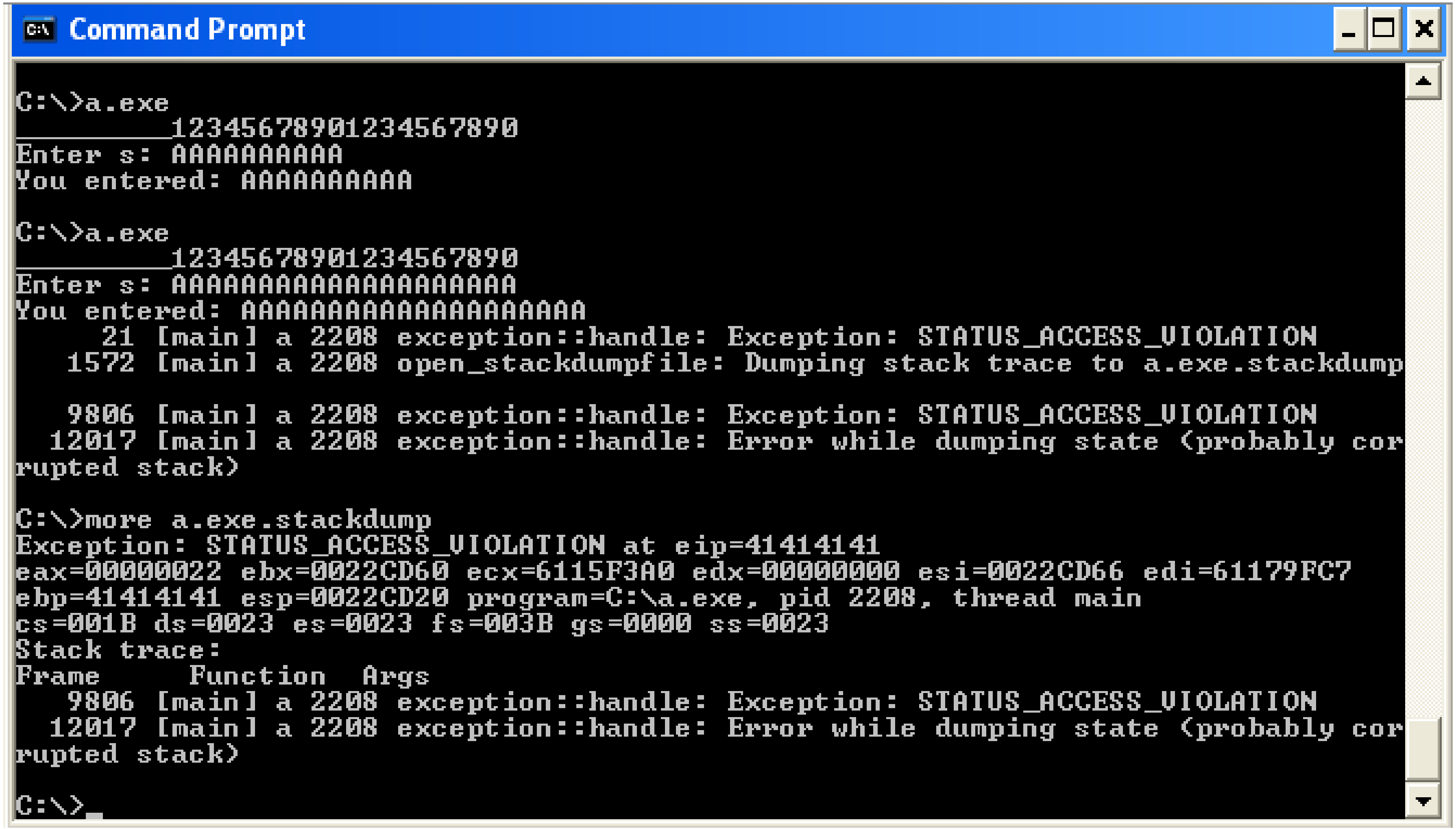
- Examine source code (if available)
- Use debugger on executable to find exploitable library
- Apply brute force
  - Inundate application with input data
  - Examine stack traces after crashes
  - But what would you input … & what would you look for?

Sample program

```
#include <stdio.h>

void f() {
  char s[9];
printf( "__________12345678901234567890\n" );
printf( "Enter s: " );
  gets( s );
printf( "You entered: %s\n", s );
  return;
}

main() {
f();
}
```

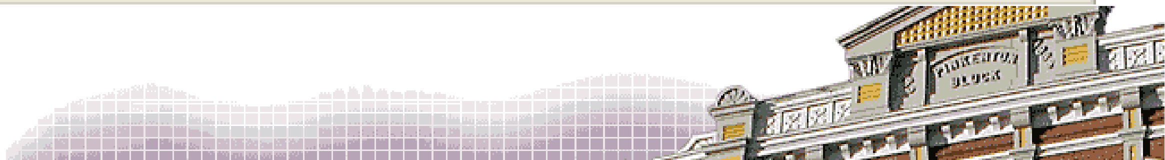


Running the program

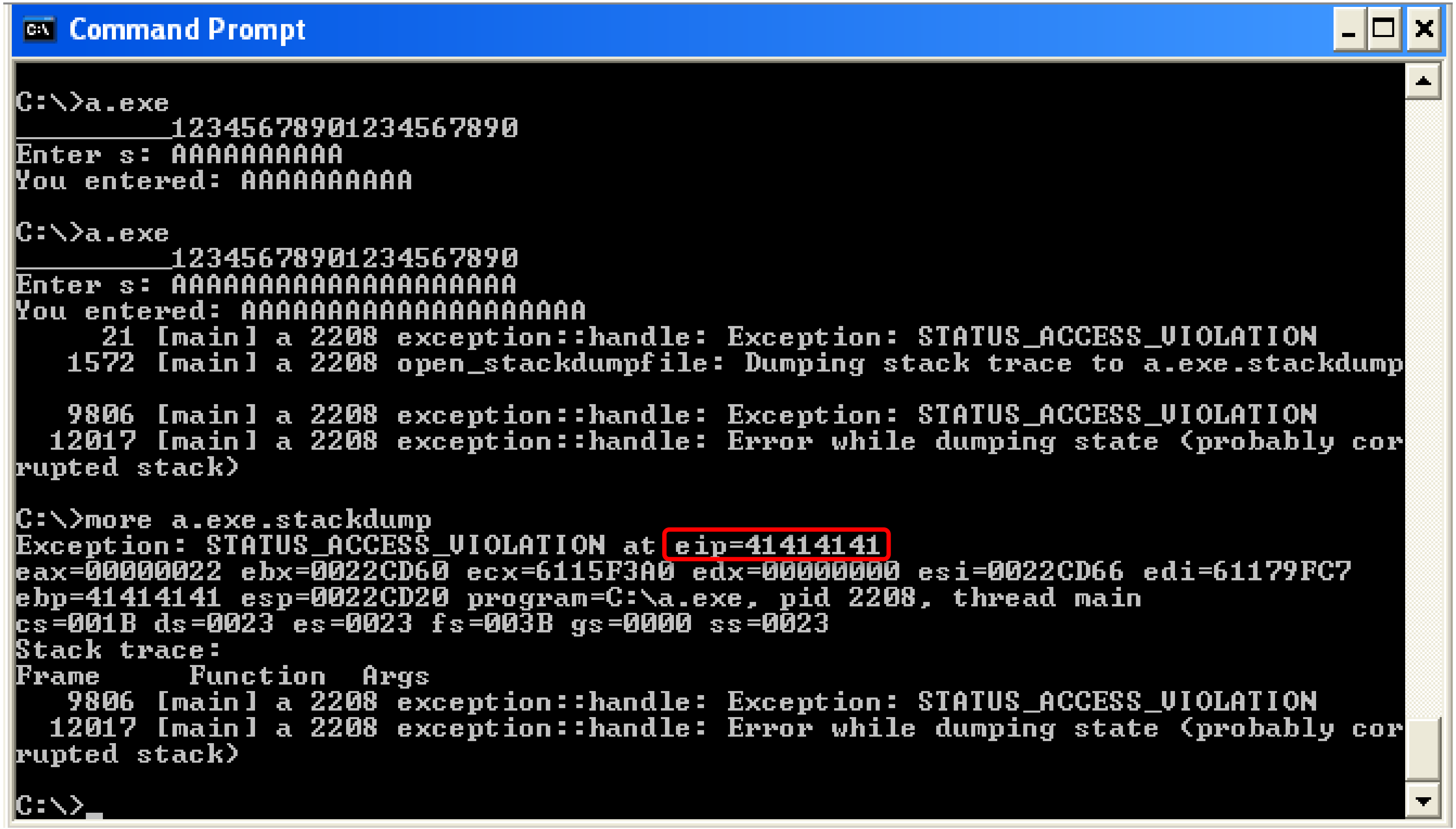


```
C:\>a.exe
_________12345678901234567890
Enter s: AAAAAAAAAA
You entered: AAAAAAAAAA

C:\>a.exe
_________12345678901234567890
Enter s: AAAAAAAAAAAAAAAAAAAA
You entered: AAAAAAAAAAAAAAAAAAAA
     21 [main] a 2208 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   1572 [main] a 2208 open_stackdumpfile: Dumping stack trace to a.exe.stackdump

   9806 [main] a 2208 exception::handle: Exception: STATUS_ACCESS_VIOLATION
  12017 [main] a 2208 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>more a.exe.stackdump
Exception: STATUS_ACCESS_VIOLATION at eip=41414141
eax=00000022 ebx=0022CD60 ecx=6115F3A0 edx=00000000 esi=0022CD66 edi=61179FC7
ebp=41414141 esp=0022CD20 program=C:\a.exe, pid 2208, thread main
cs=001B ds=0023 es=0023 fs=003B gs=0000 ss=0023
Stack trace:
Frame     Function  Args
   9806 [main] a 2208 exception::handle: Exception: STATUS_ACCESS_VIOLATION
  12017 [main] a 2208 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>_
```

UNIVERSITY *of* WASHINGTON
TACOMA

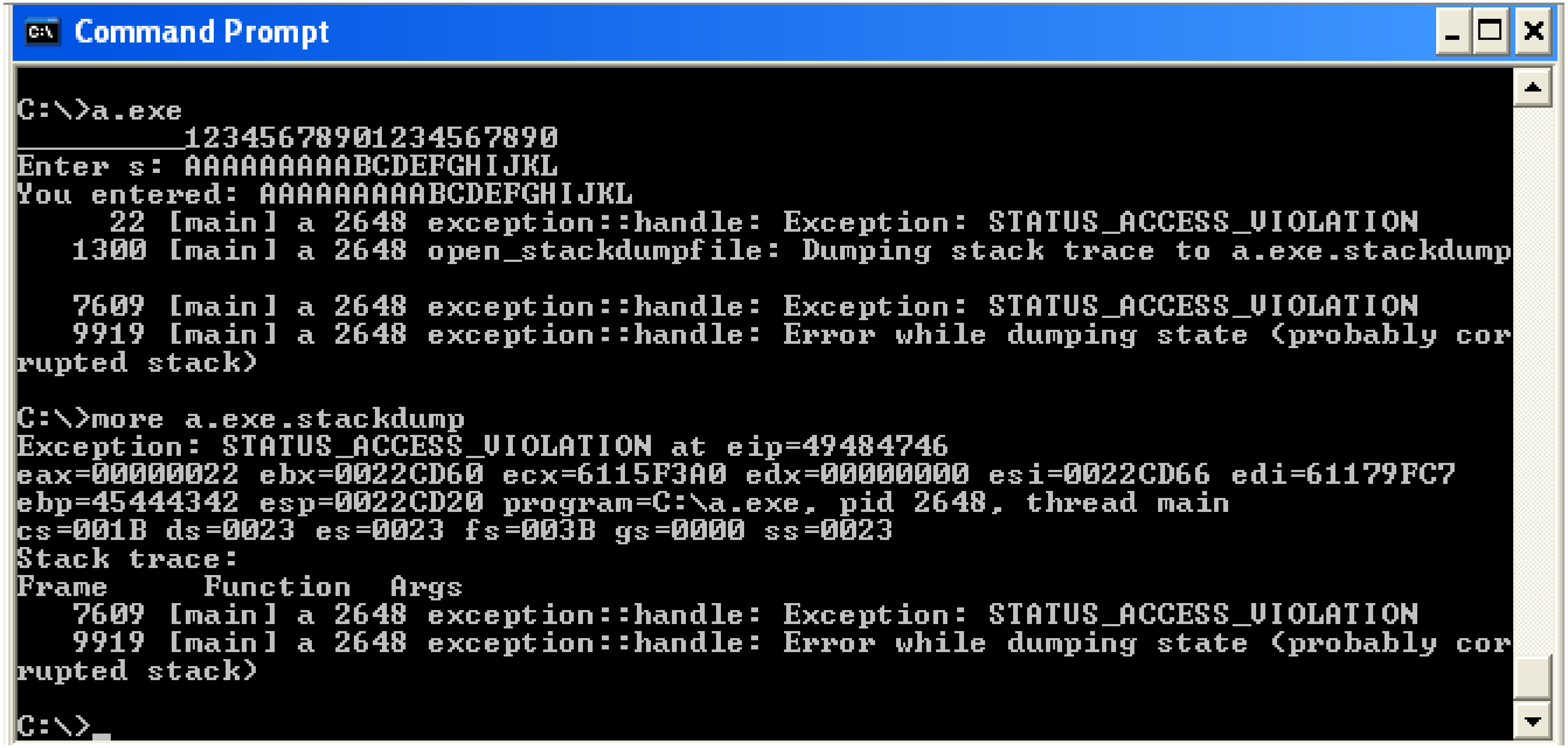


Running the program



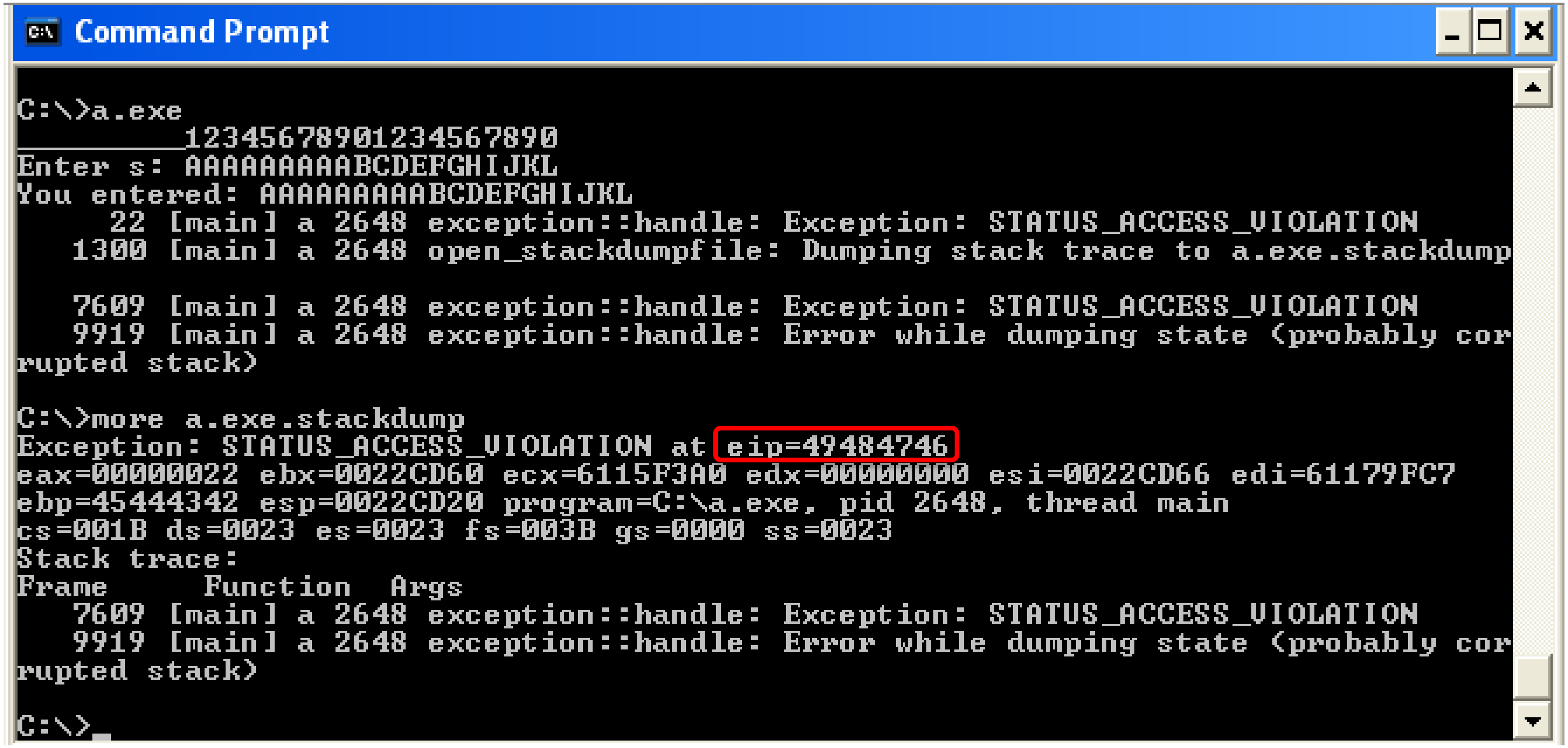
```
C:\>a.exe
_________1234567890123456789O
Enter s: AAAAAAAAAA
You entered: AAAAAAAAAA

C:\>a.exe
_________1234567890123456789O
Enter s: AAAAAAAAAAAAAAAAAAAA
You entered: AAAAAAAAAAAAAAAAAAAA
     21 [main] a 2208 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   1572 [main] a 2208 open_stackdumpfile: Dumping stack trace to a.exe.stackdump

   9806 [main] a 2208 exception::handle: Exception: STATUS_ACCESS_VIOLATION
  12017 [main] a 2208 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>more a.exe.stackdump
Exception: STATUS_ACCESS_VIOLATION at eip=41414141
eax=00000022 ebx=0022CD60 ecx=6115F3A0 edx=00000000 esi=0022CD66 edi=61179FC7
ebp=41414141 esp=0022CD20 program=C:\a.exe, pid 2208, thread main
cs=001B ds=0023 es=0023 fs=003B gs=0000 ss=0023
Stack trace:
Frame     Function  Args
   9806 [main] a 2208 exception::handle: Exception: STATUS_ACCESS_VIOLATION
  12017 [main] a 2208 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>
```

Running the program



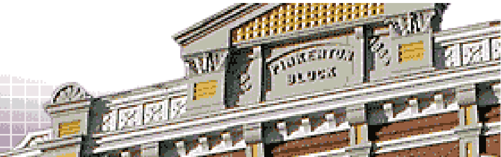
```
C:\>a.exe
_________12345678901234567890
Enter s: AAAAAAAAABCDEFGHIJKL
You entered: AAAAAAAAABCDEFGHIJKL
     22 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   1300 [main] a 2648 open_stackdumpfile: Dumping stack trace to a.exe.stackdump

   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>more a.exe.stackdump
Exception: STATUS_ACCESS_VIOLATION at eip=49484746
eax=00000022 ebx=0022CD60 ecx=6115F3A0 edx=00000000 esi=0022CD66 edi=61179FC7
ebp=45444342 esp=0022CD20 program=C:\a.exe, pid 2648, thread main
cs=001B ds=0023 es=0023 fs=003B gs=0000 ss=0023
Stack trace:
Frame     Function  Args
   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>_
```

Running the program

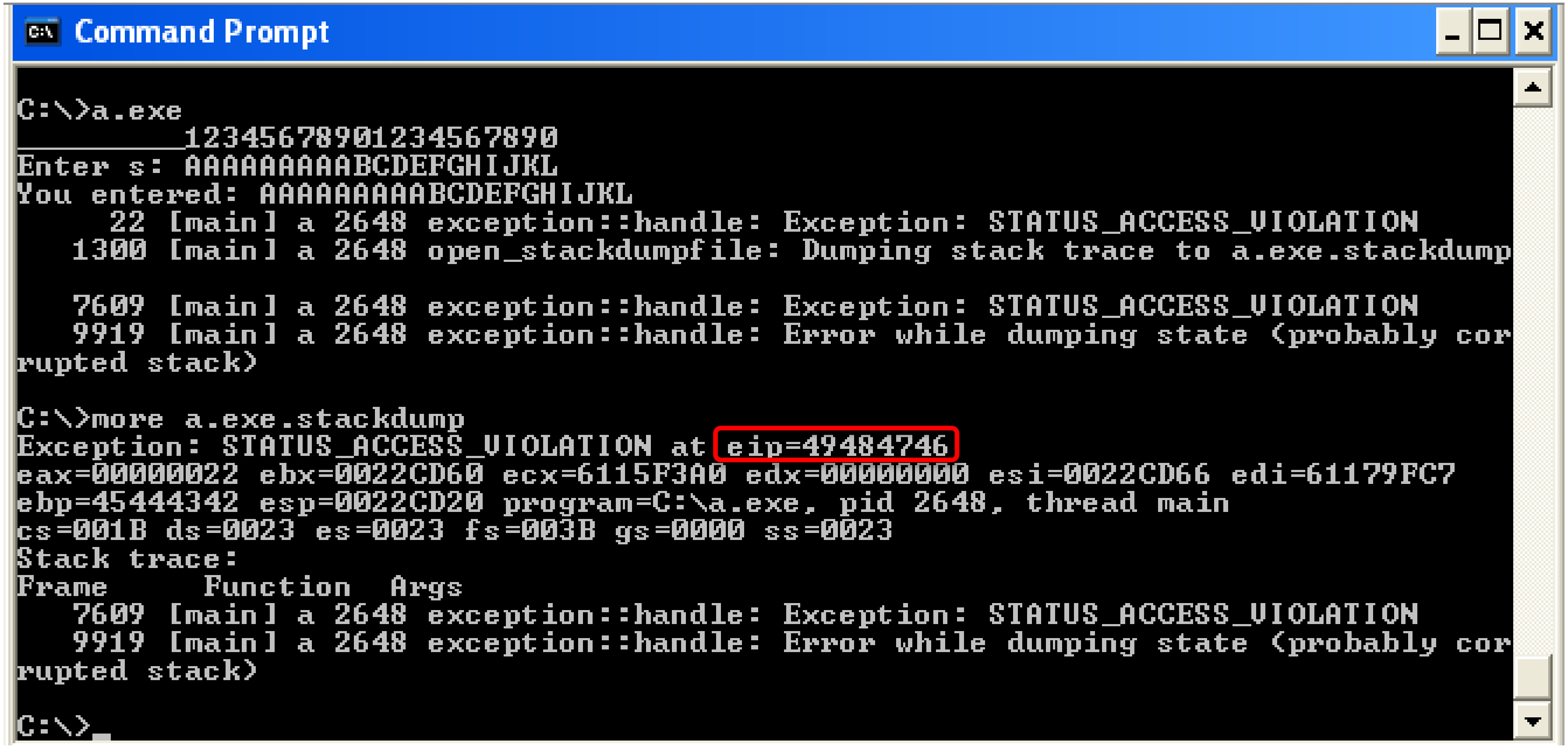


```
C:\>a.exe
_________12345678901234567890
Enter s: AAAAAAAAABCDEFGHIJKL
You entered: AAAAAAAAABCDEFGHIJKL
     22 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   1300 [main] a 2648 open_stackdumpfile: Dumping stack trace to a.exe.stackdump

   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>more a.exe.stackdump
Exception: STATUS_ACCESS_VIOLATION at eip=49484746
eax=00000022 ebx=0022CD60 ecx=6115F3A0 edx=00000000 esi=0022CD66 edi=61179FC7
ebp=45444342 esp=0022CD20 program=C:\a.exe, pid 2648, thread main
cs=001B ds=0023 es=0023 fs=003B gs=0000 ss=0023
Stack trace:
Frame     Function  Args
   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>_
```

UNIVERSITY of WASHINGTON TACOMA



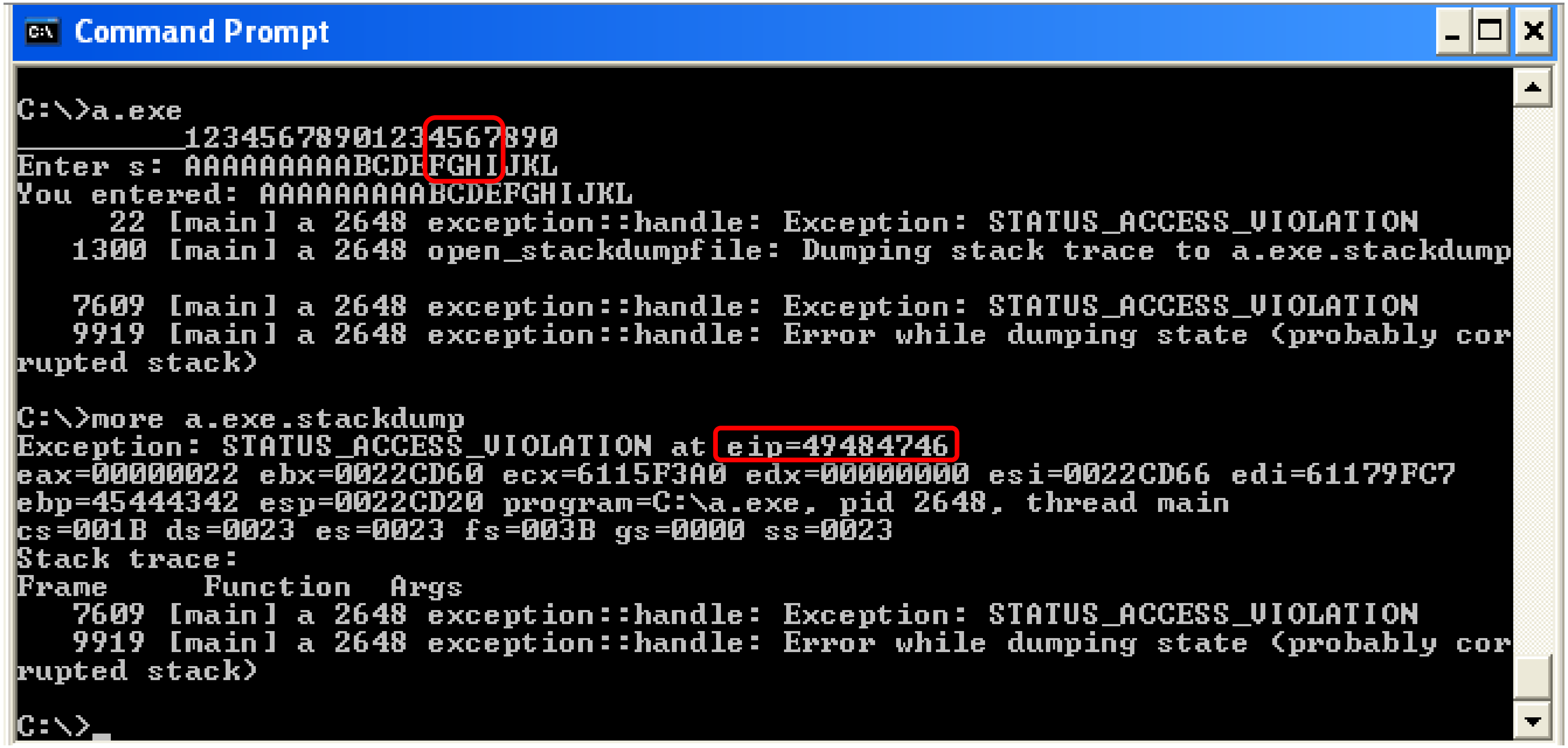
Running the program



```
C:\>a.exe
_________12345678901234567890
Enter s: AAAAAAAAABCDEFGHIJKL
You entered: AAAAAAAAABCDEFGHIJKL
     22 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   1300 [main] a 2648 open_stackdumpfile: Dumping stack trace to a.exe.stackdump

   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>more a.exe.stackdump
Exception: STATUS_ACCESS_VIOLATION at eip=49484746
eax=00000022 ebx=0022CD60 ecx=6115F3A0 edx=00000000 esi=0022CD66 edi=61179FC7
ebp=45444342 esp=0022CD20 program=C:\a.exe, pid 2648, thread main
cs=001B ds=0023 es=0023 fs=003B gs=0000 ss=0023
Stack trace:
Frame     Function  Args
   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>_
```

0x49 = “I”, 0x48 = “H”, 0x47 = “G”, 0x46 = “F”

Running the program

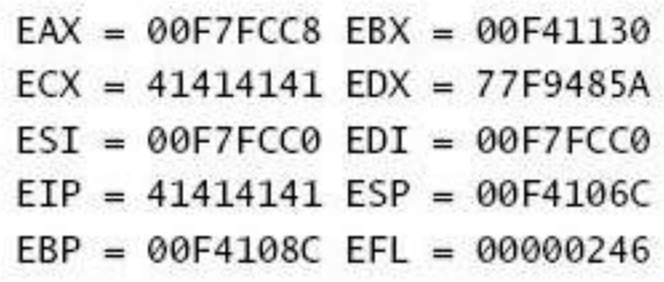


```
C:\>a.exe
_________12345678901234567890
Enter s: AAAAAAAAABCDEFGHIJKL
You entered: AAAAAAAAABCDEFGHIJKL
     22 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   1300 [main] a 2648 open_stackdumpfile: Dumping stack trace to a.exe.stackdump

   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>more a.exe.stackdump
Exception: STATUS_ACCESS_VIOLATION at eip=49484746
eax=00000022 ebx=0022CD60 ecx=6115F3A0 edx=00000000 esi=0022CD66 edi=61179FC7
ebp=45444342 esp=0022CD20 program=C:\a.exe, pid 2648, thread main
cs=001B ds=0023 es=0023 fs=003B gs=0000 ss=0023
Stack trace:
Frame     Function  Args
   7609 [main] a 2648 exception::handle: Exception: STATUS_ACCESS_VIOLATION
   9919 [main] a 2648 exception::handle: Error while dumping state (probably cor
rupted stack)

C:\>_
```

0x49 = “I”, 0x48 = “H”, 0x47 = “G”, 0x46 = “F”

Strategy & Structure of a “Sploit”

- “Fuzzing”
  - Repeated input patterns
  - AAAA… (“A” = 0x41)
  - ABCDEFG…
  - DEF1, DEF2, DEF3, …
- NOP (No Operation)
  - 0x90 on x86
  - Also:
    - Add 0
    - Multiply by 1
    - Jump to next instruction
    - …



```
EAX = 00F7FCC8 EBX = 00F41130
ECX = 41414141 EDX = 77F9485A
ESI = 00F7FCC0 EDI = 00F7FCC0
EIP = 41414141 ESP = 00F4106C
EBP = 00F4108C EFL = 00000246
```

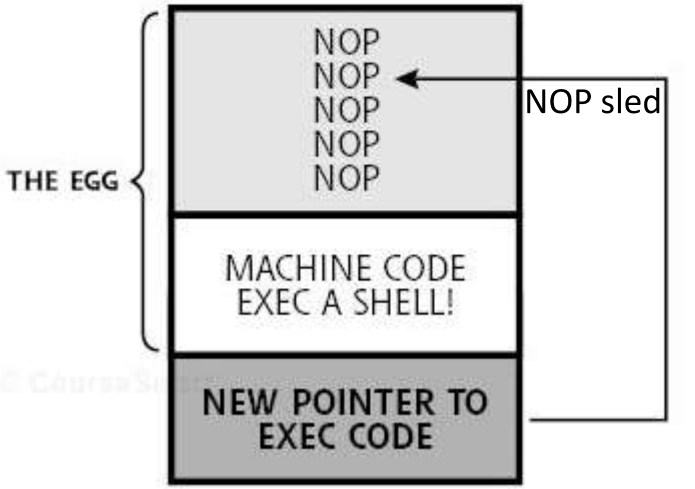
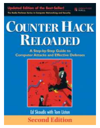
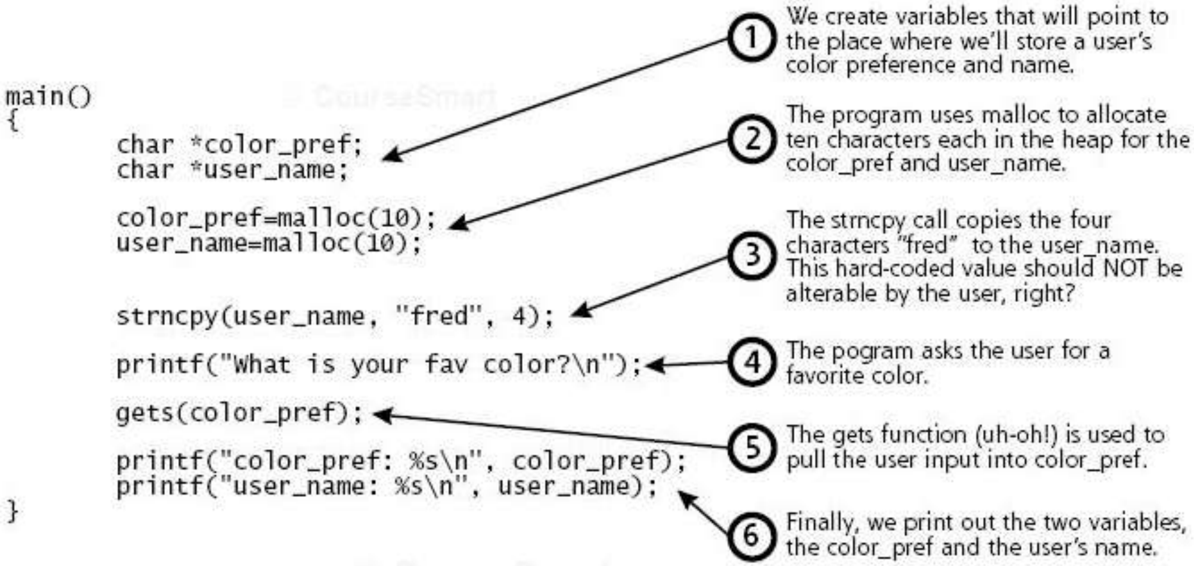


Figure 7.8 The structure of an exploit (also known as a sploit)



Heap-based Buffer Overflow Attacks



```
main()
{
        char *color_pref;
        char *user_name;

        color_pref=malloc(10);
        user_name=malloc(10);

        strncpy(user_name, "fred", 4);

        printf("What is your fav color?\n");

        gets(color_pref);

        printf("color_pref: %s\n", color_pref);
        printf("user_name: %s\n", user_name);
}
```

1. We create variables that will point to the place where we'll store a user's color preference and name.
2. The program uses malloc to allocate ten characters each in the heap for the color_pref and user_name.
3. The strncpy call copies the four characters "fred" to the user_name. This hard-coded value should NOT be alterable by the user, right?
4. The pogram asks the user for a favorite color.
5. The gets function (uh-oh!) is used to pull the user input into color_pref.
6. Finally, we print out the two variables, the color_pref and the user's name.

Figure 7.9 A program with a heap-based buffer overflow vulnerability.

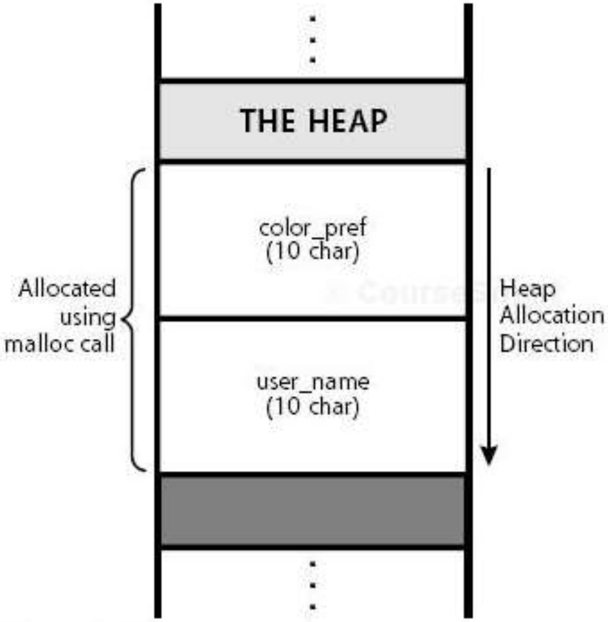
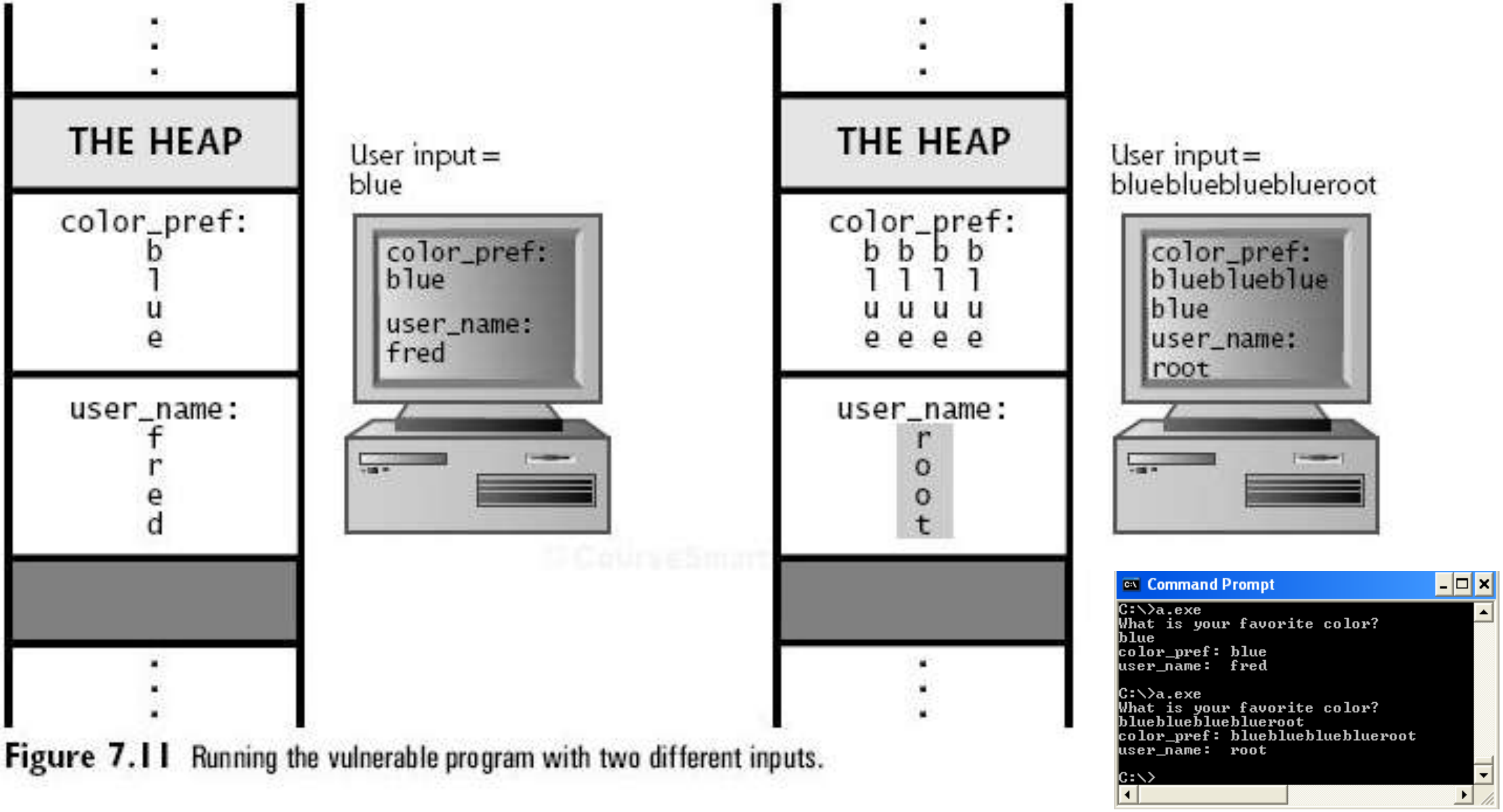


Figure 7.10 The heap holds the memory we malloc'ed.

Heap-based Buffer Overflow Attacks



```
C:\>a.exe
What is your favorite color?
blue
color_pref: blue
user_name:  fred

C:\>a.exe
What is your favorite color?
bluebluebluebluerooot
color_pref: bluebluebluebluerooot
user_name:  root

C:\>
```

Figure 7.11 Running the vulnerable program with two different inputs.

Script Kiddies & Exploit Collections

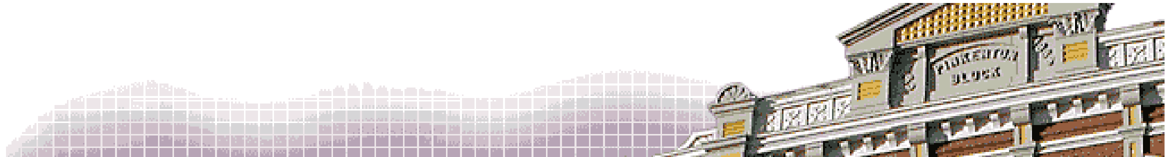
- Attacks (exploits) are widely available
  - French Security Response Team (FrSIRT)
    - http://www.vupen.com/english/
    - “Only available to trusted organizations”
  - Packet Storm Security
    - http://packetstormsecurity.org/
  - Security Focus Bugtraq Archives
    - http://www.securityfocus.com/bid
  - Metasploit Project
    - http://www.metasploit.com
- Little or no knowledge required

VUPEN security

packet storm

Symantec. | Connect

metasploit



Exploitation Engines

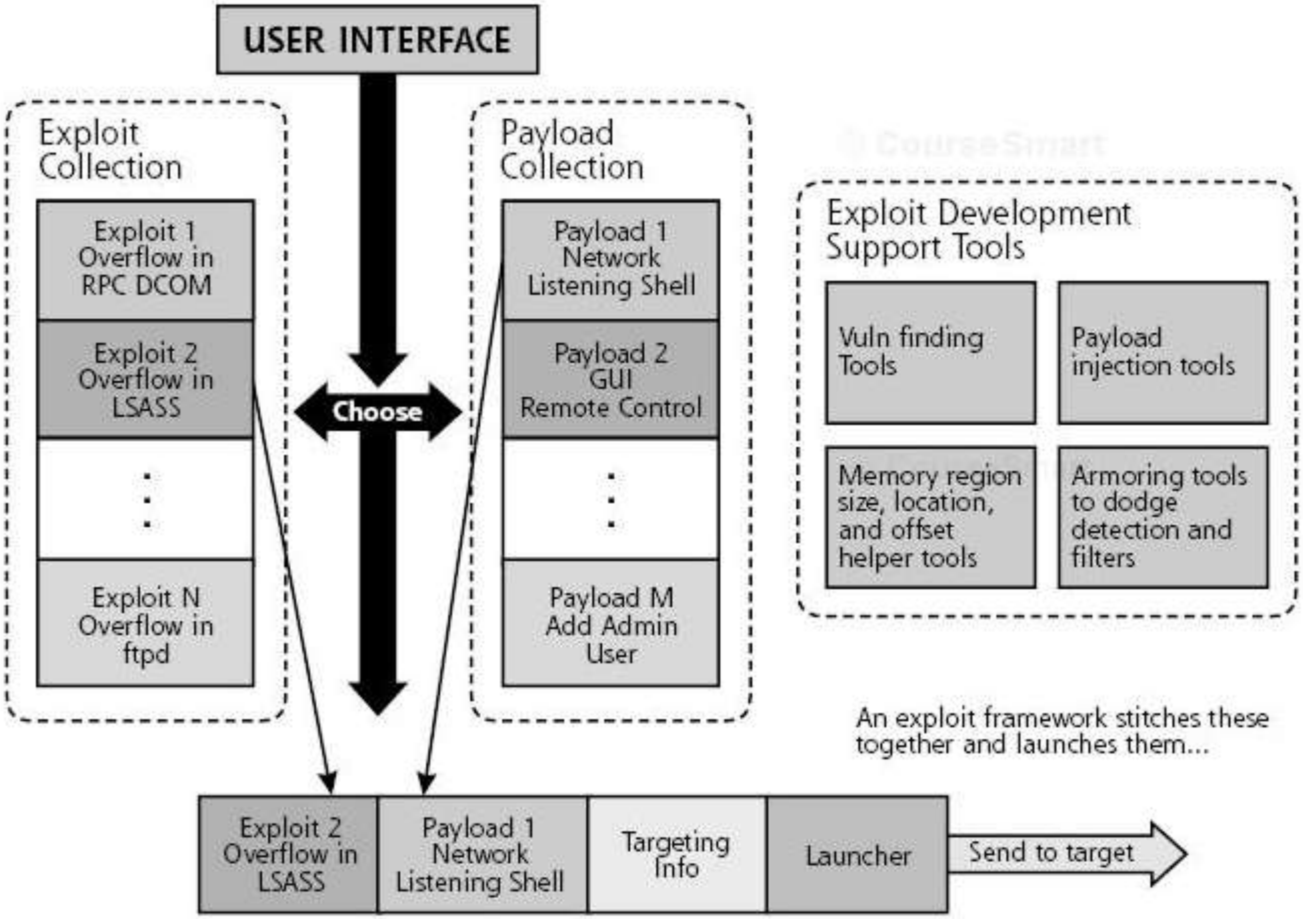
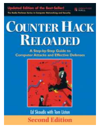

USER INTERFACE
Exploit Collection
Exploit 1 Overflow in RPC DCOM
Exploit 2 Overflow in LSASS
Exploit N Overflow in ftpd
Choose
Payload Collection
Payload 1 Network Listening Shell
Payload 2 GUI Remote Control
Payload M Add Admin User
Exploit Development Support Tools
Vuln finding Tools
Payload injection tools
Memory region size, location, and offset helper tools
Armoring tools to dodge detection and filters
An exploit framework stitches these together and launches them...
Exploit 2 Overflow in LSASS
Payload 1 Network Listening Shell
Targeting Info
Launcher
Send to target

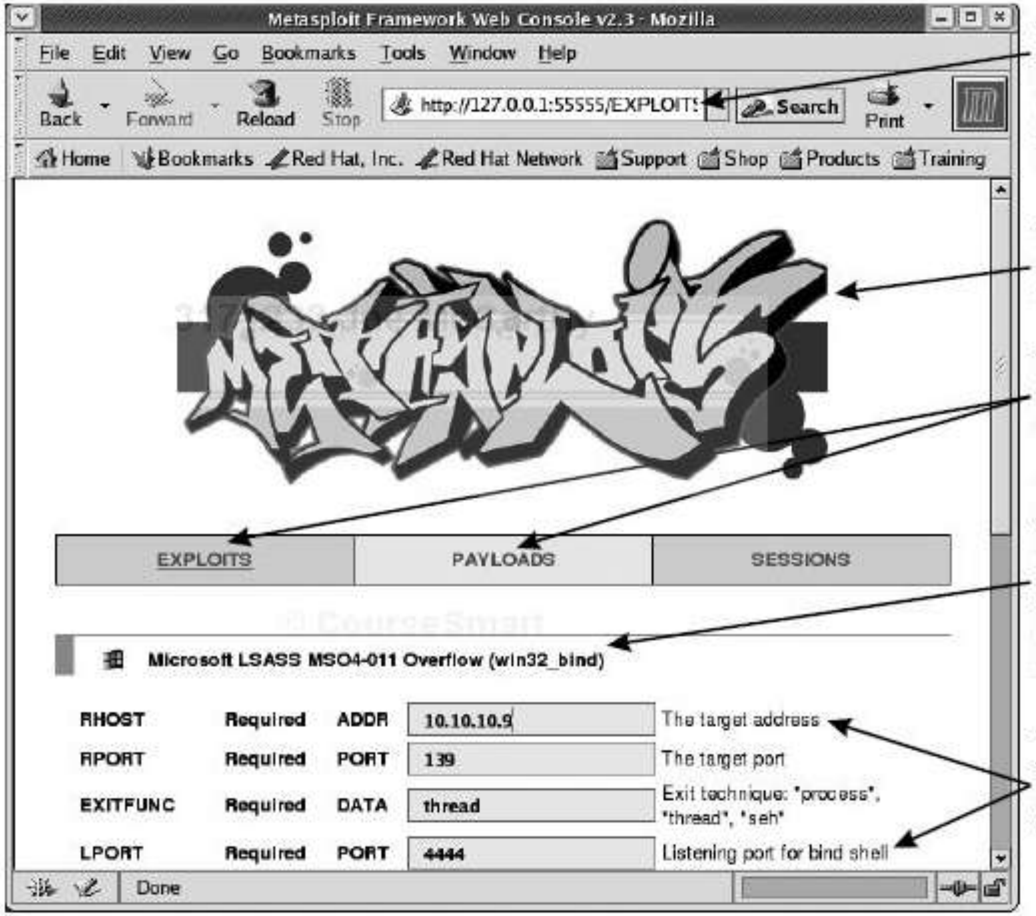
Figure 7.12 The components of Metasploit.



Sample Payloads

- Bind shell to current port
- Bind shell to arbitrary port
- Reverse shell
- Windows VNC Server DLL
- Reverse VNC DLL Inject
- Inject DLL into running application
- Create local admin user
- The Meterpreter (Metasploit Interpreter)

Metasploit - GUI



http://127.0.0.1:55555/EXPLOITS

Microsoft LSASS MS04-011 Overflow (win32_bind)

RHOST	Required	ADDR	10.10.10.9	The target address
RPORT	Required	PORT	139	The target port
EXITFUNC	Required	DATA	thread	Exit technique: "process", "thread", "seh"
LPORT	Required	PORT	4444	Listening port for bind shell

The attacker invokes the Metasploit Web interface and surfs there from a browser on the local machine to 127.0.0.1:55555.

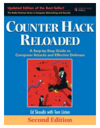
This text says "Metasploit", in a very nonstandard font.

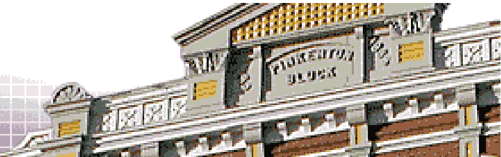
The attacker chooses from a list of exploits and payloads.

Here, we've chosen the LSASS buffer overflow exploit (LSASS MS04-011), with a Windows command shell bound to a TCP port (win32_bind).

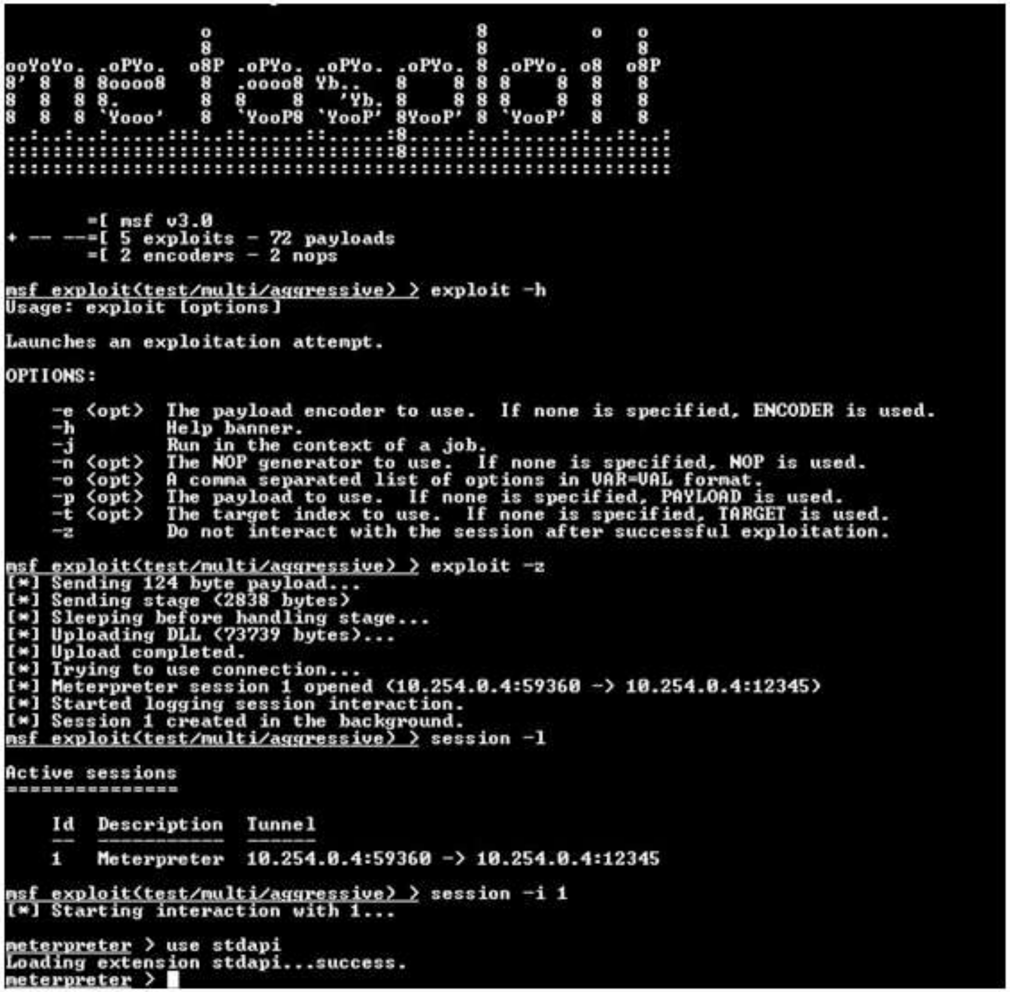
The attacker configures the target address, and a port to have a target shell listen on.

Figure 7.13 Metasploit's Web-based interface.





Metasploit – command line

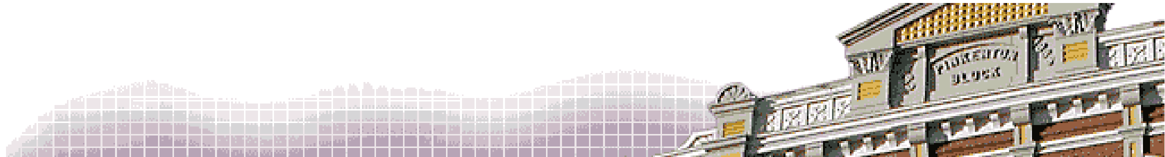


```
                 o                       8         o   o
                 8                       8             8
ooYoYo. .oPYo.  o8P .oPYo. .oPYo. .oPYo. 8 .oPYo. o8  o8P
8' 8  8 8oooo8   8  .oooo8 Yb..   8    8 8 8    8  8   8
8  8  8 8.       8  8    8   'Yb. 8    8 8 8    8  8   8
8  8  8 `Yooo'   8  `YooP8 `YooP' 8YooP' 8 `YooP'  8   8
..:..:..:.....::..::.....::.....:8.....:..:.....::..::..:
::::::::::::::::::::::::::::::::::8:::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::


       =[ msf v3.0
+ -- --=[ 5 exploits - 72 payloads
       =[ 2 encoders - 2 nops

msf exploit(test/multi/aggressive) > exploit -h
Usage: exploit [options]

Launches an exploitation attempt.

OPTIONS:

    -e <opt>  The payload encoder to use.  If none is specified, ENCODER is used.
    -h        Help banner.
    -j        Run in the context of a job.
    -n <opt>  The NOP generator to use.  If none is specified, NOP is used.
    -o <opt>  A comma separated list of options in VAR=VAL format.
    -p <opt>  The payload to use.  If none is specified, PAYLOAD is used.
    -t <opt>  The target index to use.  If none is specified, TARGET is used.
    -z        Do not interact with the session after successful exploitation.

msf exploit(test/multi/aggressive) > exploit -z
[*] Sending 124 byte payload...
[*] Sending stage (2838 bytes)
[*] Sleeping before handling stage...
[*] Uploading DLL (73739 bytes)...
[*] Upload completed.
[*] Trying to use connection...
[*] Meterpreter session 1 opened (10.254.0.4:59360 -> 10.254.0.4:12345)
[*] Started logging session interaction.
[*] Session 1 created in the background.
msf exploit(test/multi/aggressive) > session -l

Active sessions
===============

    Id  Description  Tunnel
    --  -----------  ------
    1   Meterpreter  10.254.0.4:59360 -> 10.254.0.4:12345

msf exploit(test/multi/aggressive) > session -i 1
[*] Starting interaction with 1...

meterpreter > use stdapi
Loading extension stdapi...success.
meterpreter >
```

Pros & Cons of Exploit Frameworks

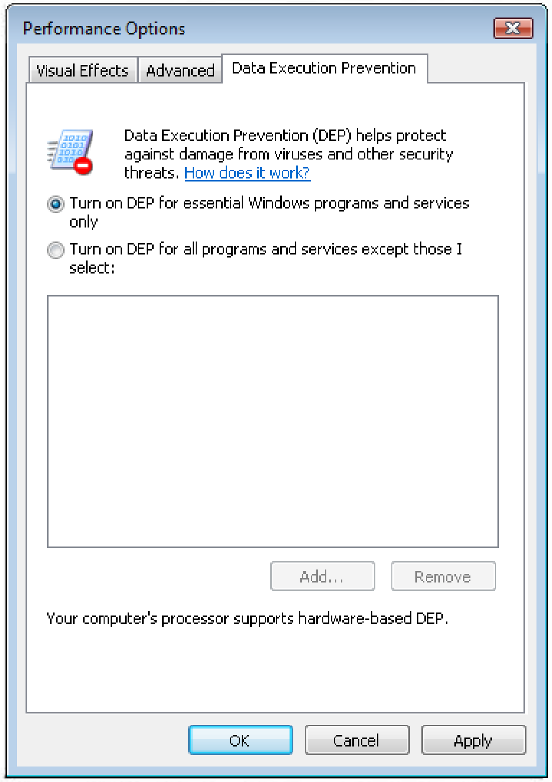
Pros & Cons of Exploit Frameworks

- Advantages for Attackers
  - Reduced time
  - Increased quality
- Advantages for Defenders
  - Increased accuracy of security assessments
    - Vulnerability scans yield many false positives (30-50%)
    - Scan, then sploit to find “real” problems
  - Verify IDS / IPS functionality
    - Malfunctions, misconfiguration, pre-emptive attacks
  - Improving management awareness
    - “Please don’t steal this file!”



Defenses against Buffer Overflow Attacks

- Safer programming
  - StackGuard, Stack Shield
- Security reviews
  - ITS4 (“It’s the Software, Stupid - Security Scanner”)
  - RATS (Rough Auditing Tool for Security)
  - Flawfinder



Data Execution Prevention (DEP) helps protect against damage from viruses and other security threats. How does it work?

● Turn on DEP for essential Windows programs and services only

○ Turn on DEP for all programs and services except those I select:

Your computer's processor supports hardware-based DEP.

Figure 7.14 Windows XP Service Pack 2 and Windows 2003 Service Pack 1

For more information

- "Smashing the Stack for Fun and Profit"
  - Aleph One, aleph1@underground.org
  - http://www.phrack.org/issues.html?id=14&issue=49
- Common Vulnerabilities & Exposures
  - http://cve.mitre.org/cve/
  - Total CVEs: 45,149
  - Stack-based overflow vulnerabilities
    - 1200+:
      IE, Safari, Firefox, Opera, RealPlayer, QuickTime, WMP, WinAmp, DB2, Excel, Access, Word, PowerPoint, OpenOffice, Eudora, Acrobat, Reader, JDK, JRE, Norton, McAfee, eTrust, RAZR
  - Heap-based overflow vulnerabilities
    - 900+:
      IE, Opera, Firefox, Thunderbird, Apache, VB, ColdFusion, Skype, PHP, Oracle, PostgreSQL, AIM, Windows Live Messenger, WordPerfect, Outlook Express, PageMaker, PowerPoint, Excel, Netscape, McAfee, DirectX, Shockwave, Subversion, QuickTime, Norton, Sophos, Kaspersky, RSA SecurID, PuTTY, iTunes, RealPlayer, WinAmp, OpenOffice, JRE, Facebook Photo Uploader ActiveX, Blackberry