Buffer Overflow Attacks

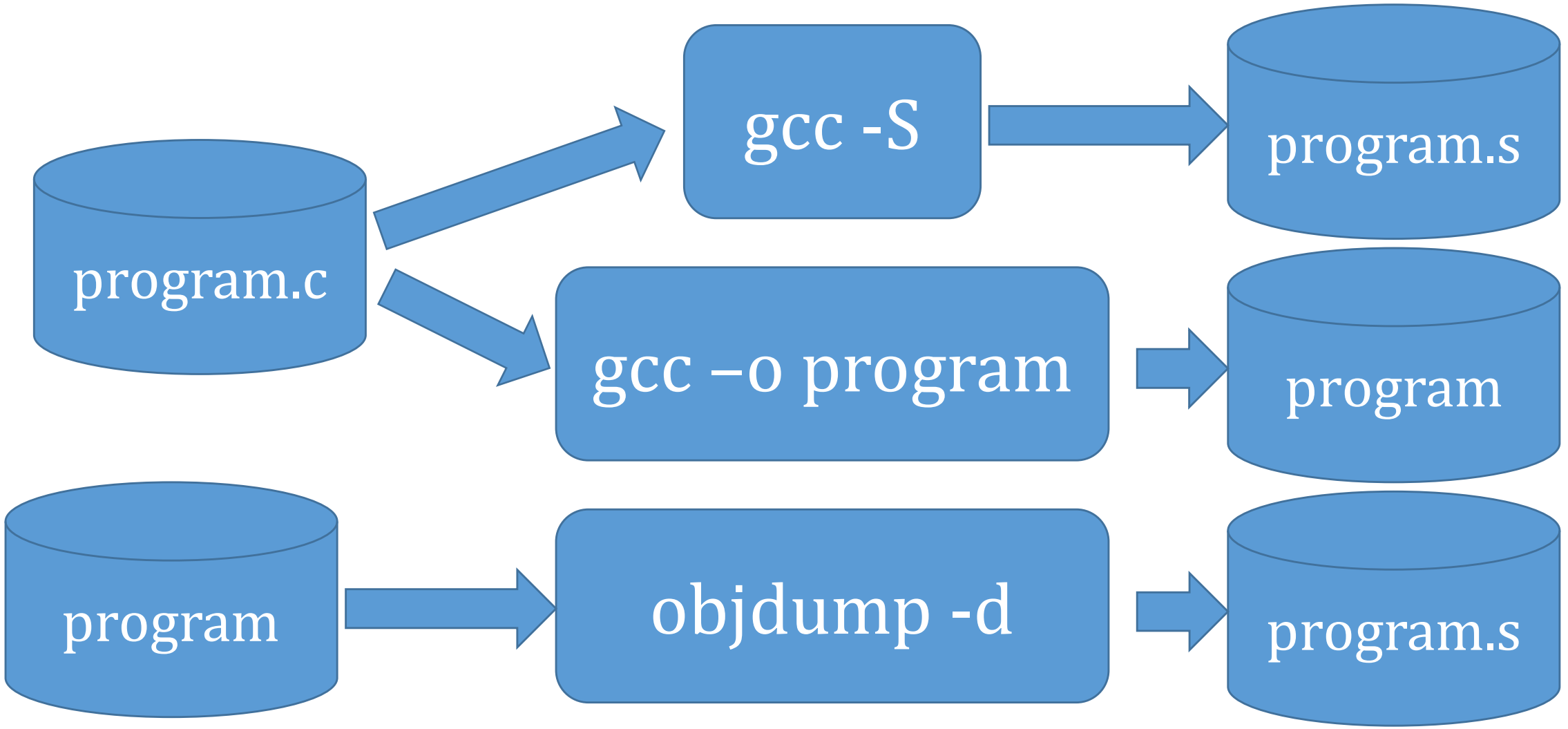
Computer Systems 3.10.3-4,

Hacking

- Roots in phone phreaking
- White Hat vs Gray Hat vs Black Hat
- Over 50% of Modern Software Development is Black Hat!

Tip the balance: Be a force for good… not evil!

Disassembly



Disclaimer – Buffer Overflow Attack

- DO NOT ABUSE!
- Ancient form of hacking
  - First documented in 1972
  - Used in 1988 “Morris Worm” – First internet virus
  - Used to hack Unix, Windows, Xbox, PS2, Wii
- Taught here as an example of what to watch out for!

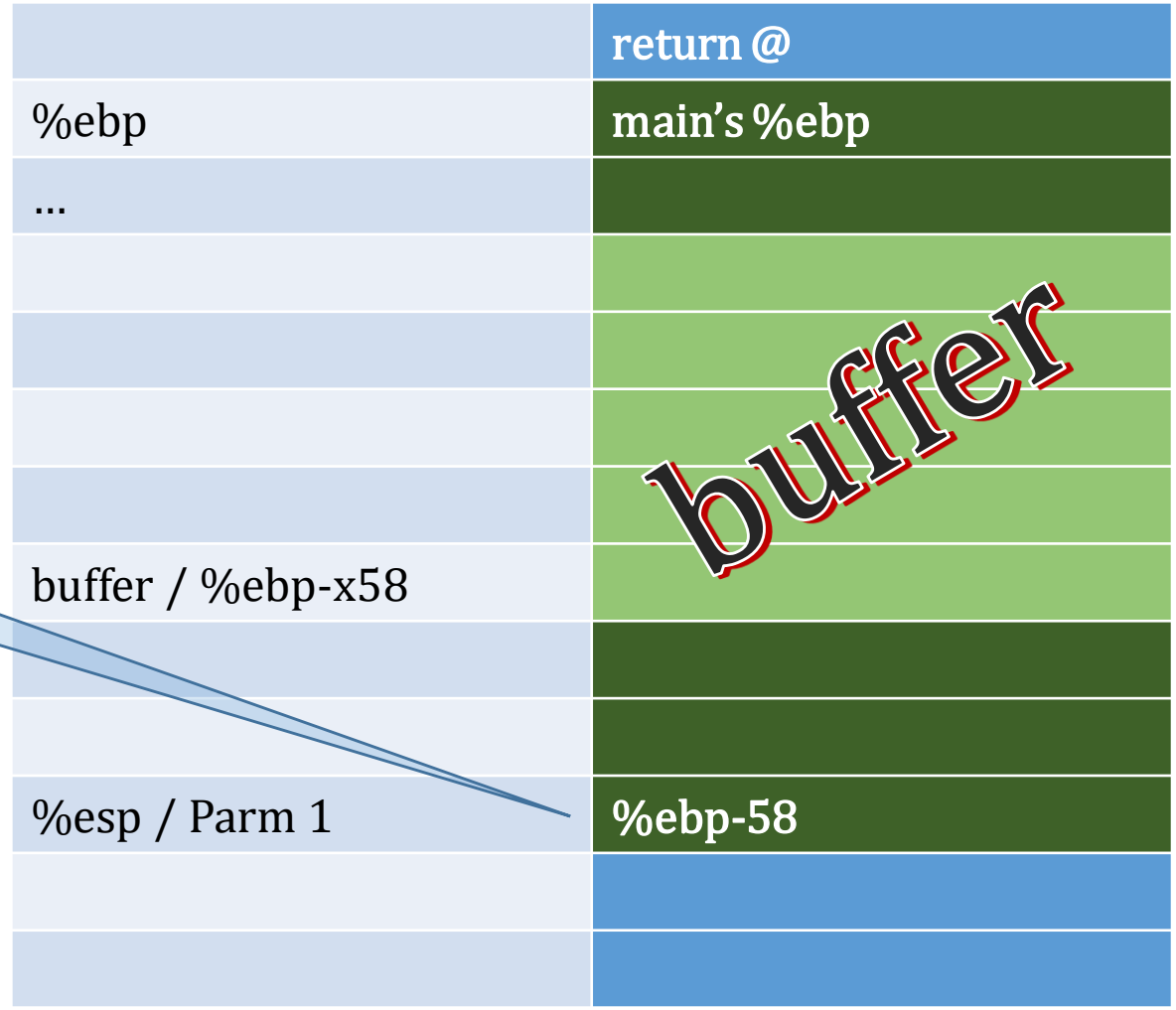
Example Vulnerable Code

```
char * getUserLine() {
   char buffer[80];
   static char retBuf[80];
   if (gets(buffer)) {
      strcpy(retBuf,buffer);
      return retBuf;
   }
   return NULL;
}
```

- "gets" reads from stdin until it finds either an end-of-file or a newline (returns 0).
- "gets" copies whatever it reads into the argument (buffer).
- "gets" does not check to make sure result fits in space allocated.

getUserLine in X86 (with stack frame)

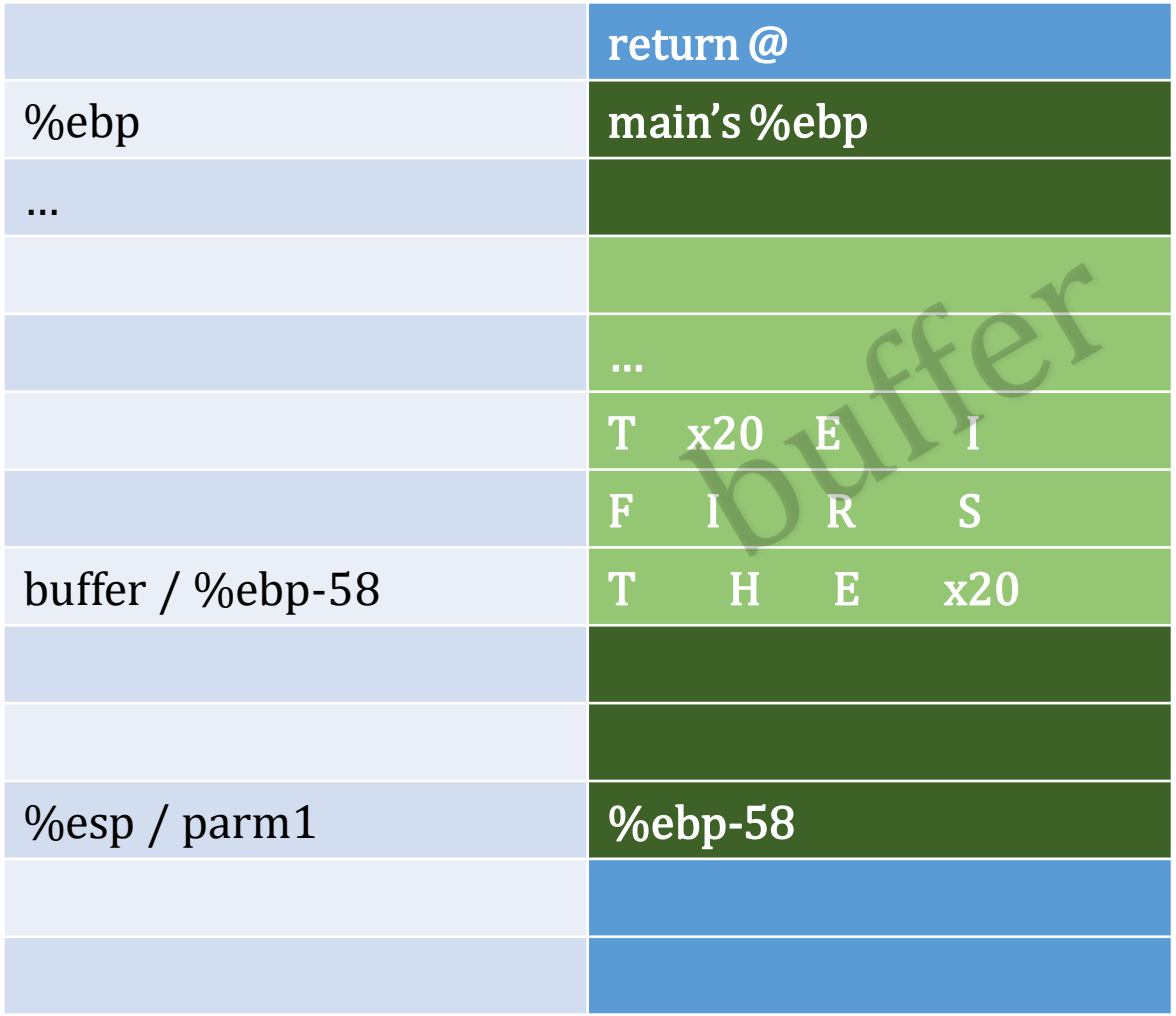
```
getUserLine:
    push   %ebp
    mov    %esp,%ebp
    sub    $0x68,%esp
    lea    -0x58(%ebp),%eax
    mov    %eax,(%esp)
    call   401210 <_gets>
    test   %eax,%eax
    ...
    move %ebp,%esp
    pop %ebp
    ret
```



gets functionality

gets reads a file that starts with…

"THE FIRST EIGHTY …"



Mixing Hex and ASCII

- We normally treat a file as a string of ASCII characters
- In fact, each ASCII character has a hex representation…

T	H	E		F	I	R	S	T		E	I	G	H	T	Y	…
5 4	4 8	4 5	2 0	4 6	4 9	5 2	5 3	5 4	2 0	4 5	4 9	4 7	4 8	5 4	5 9	…

- We can use the command "od –t x1z" to show both ASCII and hex

```
0000000 54 48 45 20 46 49 52 53 54 20 45 49 47 48 54 59  >THE FIRST EIGHTY<
0000020 2e 2e 2e 20 20 20 20 20 20 20 20 20 20 20 20 20  >...             <
0000040 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  >                 <
```

- We can write a program to put non-ASCII hex data in a file

Example of a file with ASCII and Hex

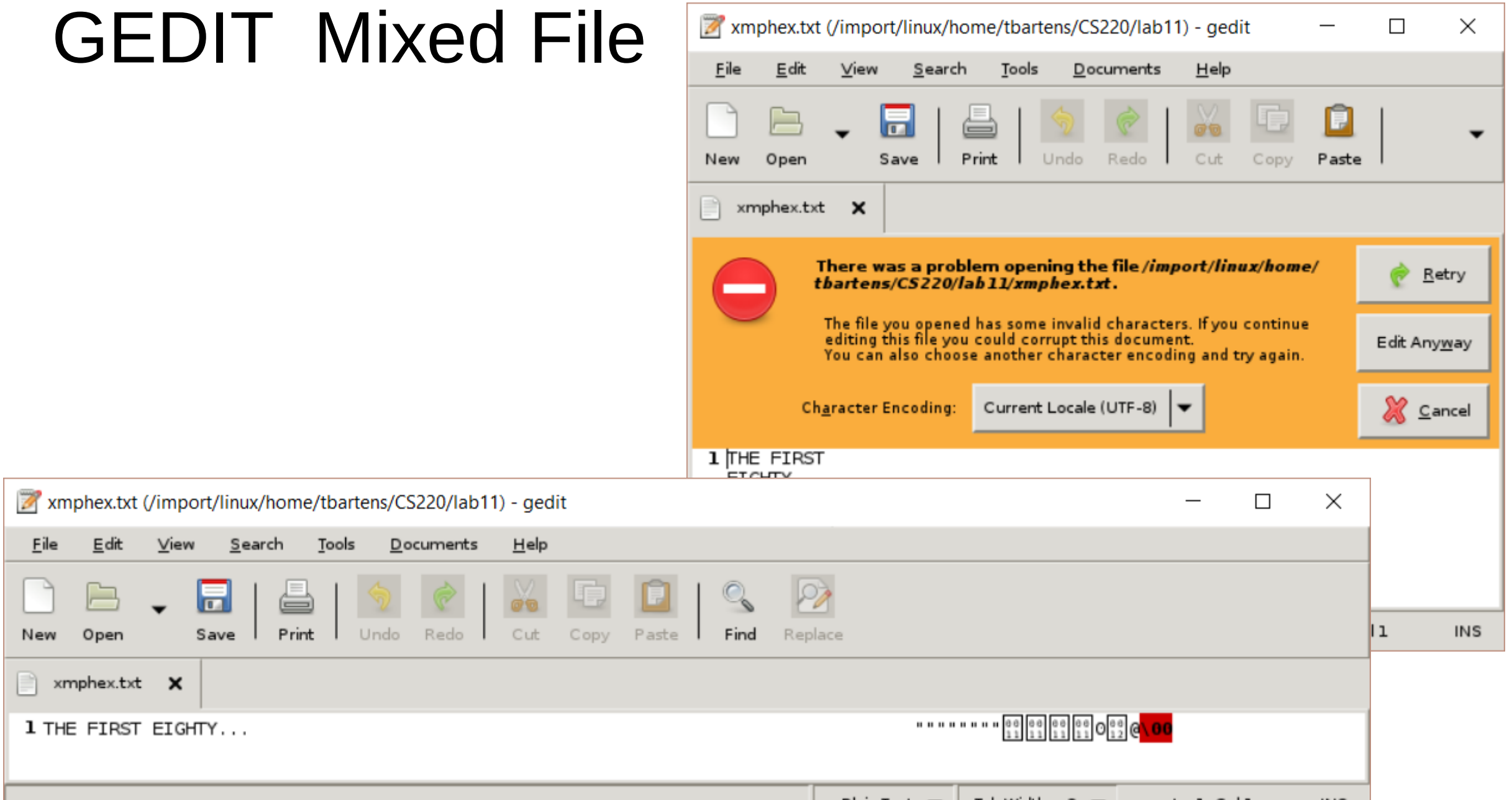
- ASCII Representation on terminal “cat file”…

```
THE FIRST EIGHTY...                                          """""""""0@
```

- Mixed representation “od -t x1z xmphex.txt”

```
0000000 54 48 45 20 46 49 52 53 54 20 45 49 47 48 54 59  >THE FIRST EIGHTY<
0000020 2e 2e 2e 20 20 20 20 20 20 20 20 20 20 20 20 20  >...             <
0000040 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  >                <
*
0000120 22 22 22 22 22 22 22 22 11 11 11 11 30 12 40 00  >"""""""".....0.@.<
0000140 0a                                               >.<
```

GEDIT Mixed File



“String” in file (stdin) read by gets

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
T	H	E	20	F	I	R	S	T	20	E	I	G	H	T	Y	.	.	.	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96			
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	22	22	22	22	22	22	22	22	11	11	11	11	31	40	20	40				

Alignment Padding

main’s ebp

Return Address!

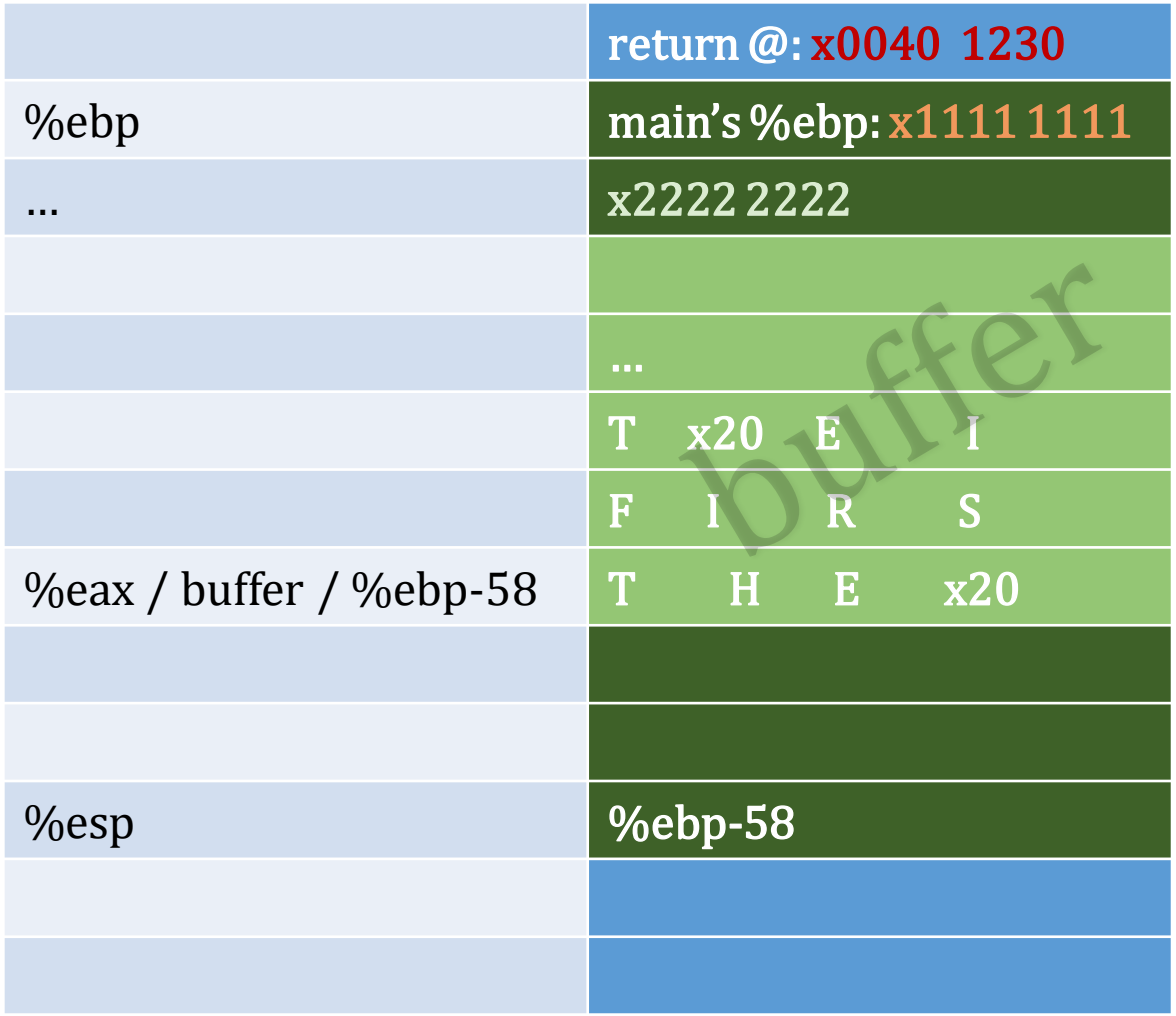
stack frame after gets returns

gets reads a file whose first line ends with…

0x2222 2222

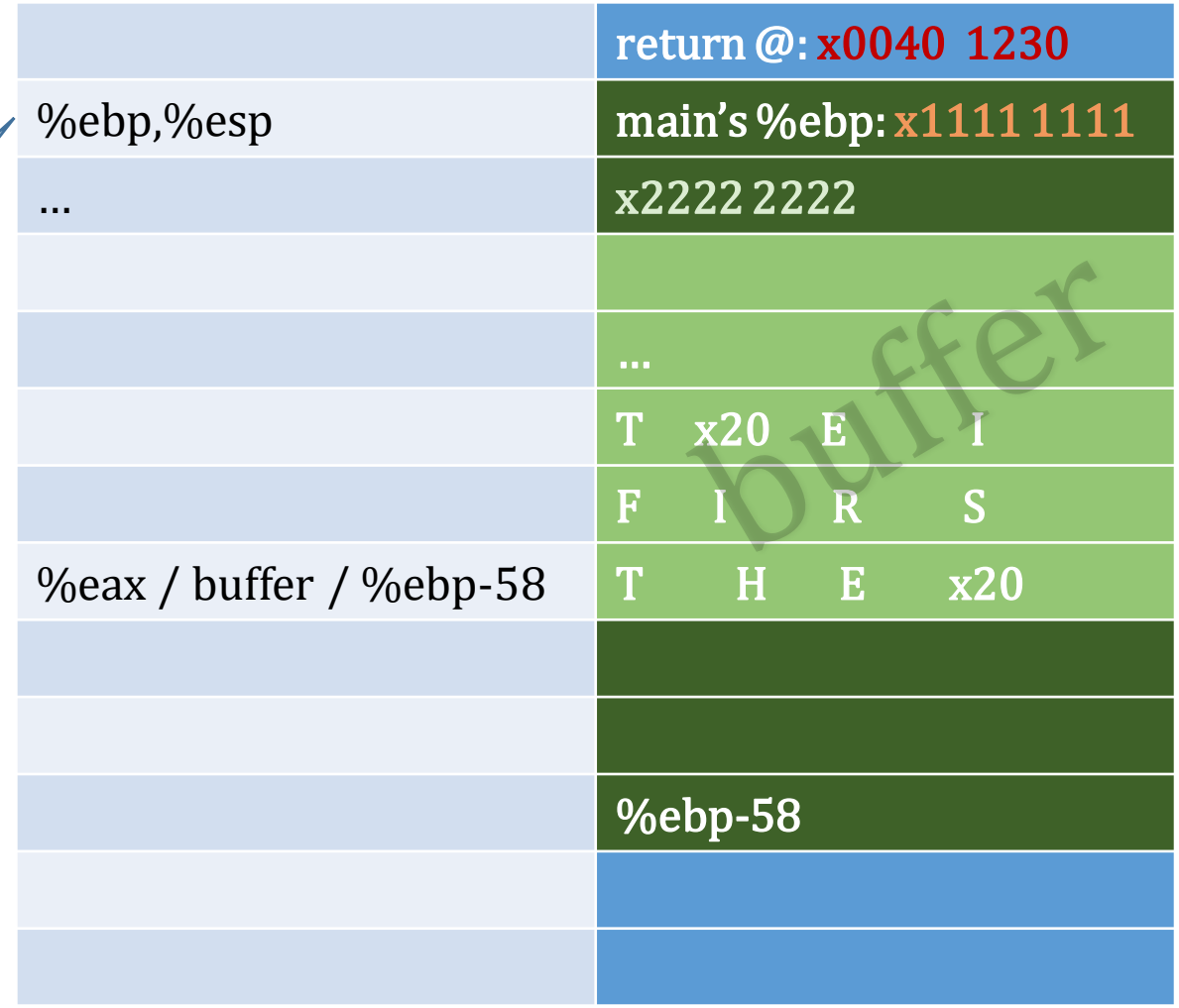
0x1111 1111

0x3012 4000 <- little endian



getUserLine in X86 (with stack)

```
getUserLine:
    push   %ebp
    mov    %esp,%ebp
    sub    $0x68,%esp
    lea    -0x58(%ebp),%eax
    mov    %eax,(%esp)
    call   401210 <_gets>
    test   %eax,%eax
    ...
    move %ebp,%esp
    pop %ebp
    ret
```

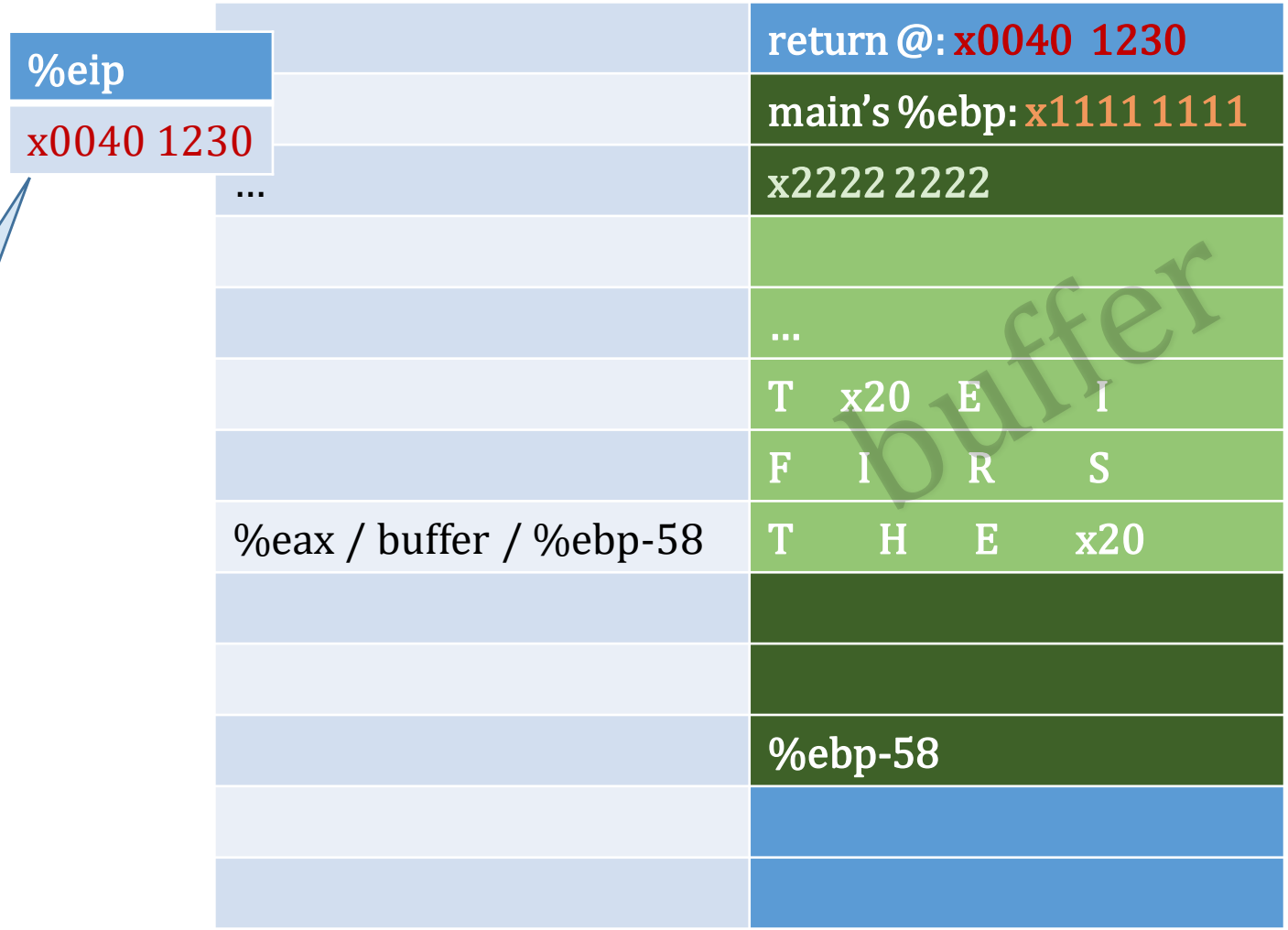


getUserLine in X86 (with stack)

```
getUserLine:
    push   %ebp
    mov    %esp,%ebp
    sub    $0x68,%esp
    lea    -0x58(%ebp),%eax
    mov    %eax,(%esp)
    call   401210 <_gets>
    test   %eax,%eax
    ...
    move %ebp,%esp
    pop %ebp
    ret
```

getUserLine in X86 (with stack)

```
getUserLine:
    push   %ebp
    mov    %esp,%ebp
    sub    $0x68,%esp
    lea    -0x58(%ebp),%eax
    mov    %eax,(%esp)
    call   401210 <_gets>
    test   %eax,%eax
    ...
    move %ebp,%esp
    pop %ebp
    ret
```



What's at 0x0040 1230?

Your evil code!

Problems with Buffer Overflow Attack

- Need to mix ASCII with hexadecimal in input file
- String can't contain "newline" (0x10)
- Need to know offset from buffer to top of stack / return @
- Overwrites caller's return address
- %ebp has been overwritten – lost top of caller's stack frame
- Need to know address of pirate routine to "return" to

- Can write a program to do that
- Basic restriction
- Get that from objdump –d
- Can get from objdump –d
- %esp points to bottom of caller's frame, and we can find its size from objdump -d
- Harder, but not impossible

Preventing Buffer Overflow w/ "Guard"

```
char * getUserLine() {
   char buffer[80];
   int guard=0xFEDCBA98;
   static char retBuf[80];
   if (gets(buffer)) {
      assert(guard==0xFEDCBA98);
      strcpy(retBuf,buffer);
      return retBuf;
   }
   return NULL;
}
```

- guard goes in stack frame after (on top of) buffer
- If buffer overflow occurs, guard will be modified
- If buffer overflow occurs, assert will fail
- Unless hacker did objdump and put 0xFEDCBA98 in his hacked file

“String” in file (stdin) read by gets

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
T	H	E	20	F	I	R	S	T	20	E	I	G	H	T	Y	.	.	.	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20

50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96			
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	98	BA	DC	FE	22	22	22	22	11	11	11	11	31	40	20	00				

Guard

Alignment Padding

main’s ebp

Return Address!

Preventing Buffer Overflow Attacks

```
char * getUserLine() {
   char buffer[80];
   static char retBuf[80];
   if (fgets(buffer,sizeof(buffer),stdin)) {
      strcpy(retBuf,buffer);
      return retBuf;
   }
   return NULL;
}
```

- "fgets" reads from any file (third parameter)
- "fgets" checks size (second parameter)
- "fgets" does not allow buffer overwrite