Miscellaneous Concepts

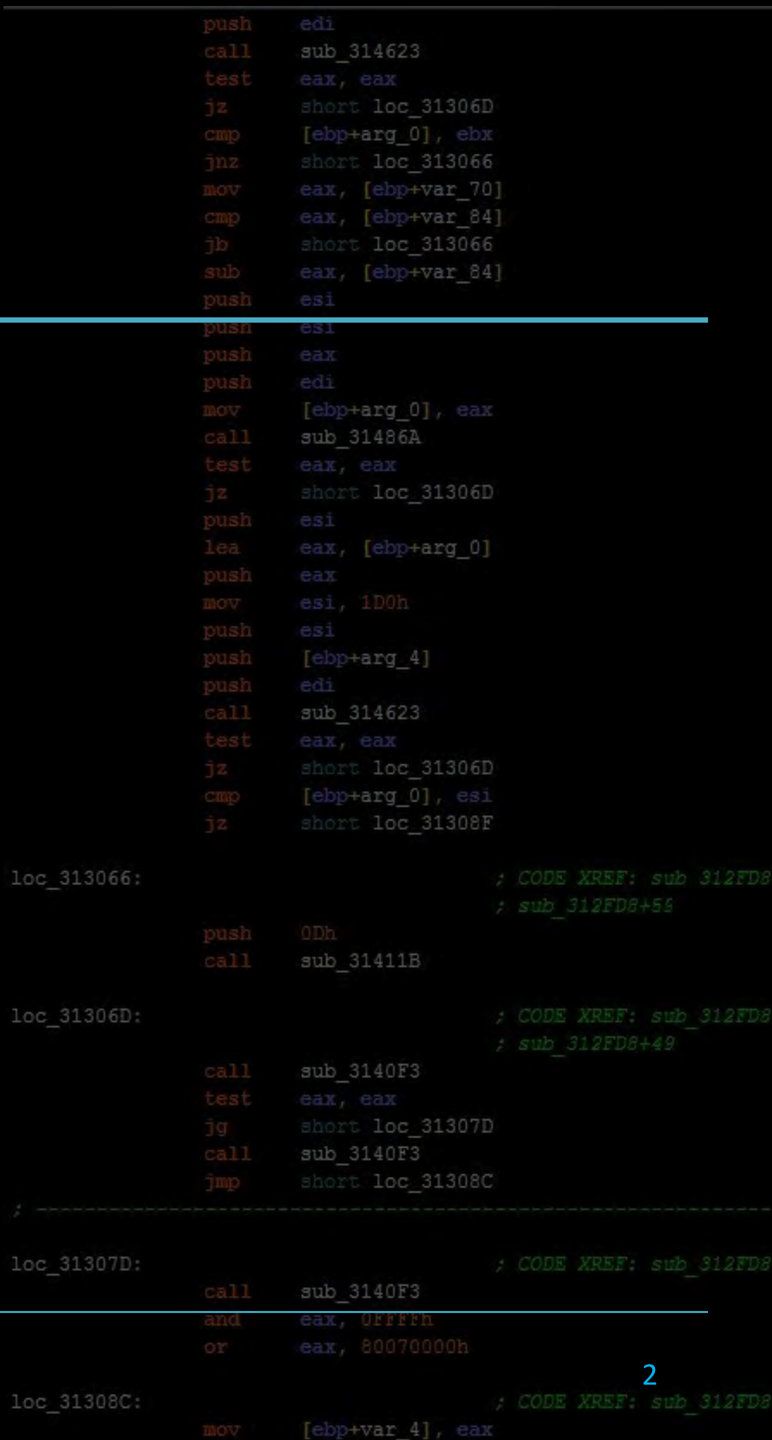
Modern Binary Exploitation

CSCI 4968 - Spring 2015

Austin Ralls

Lecture Overview

- Miscellaneous Concepts
  - Integers in C
  - Uninitialized data
  - Structs
  - File Descriptors
  - Stack Cookies



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Misc Concepts

- There’s a lot of smaller bits and pieces to this class that are important, but too small to warrant their own lectures

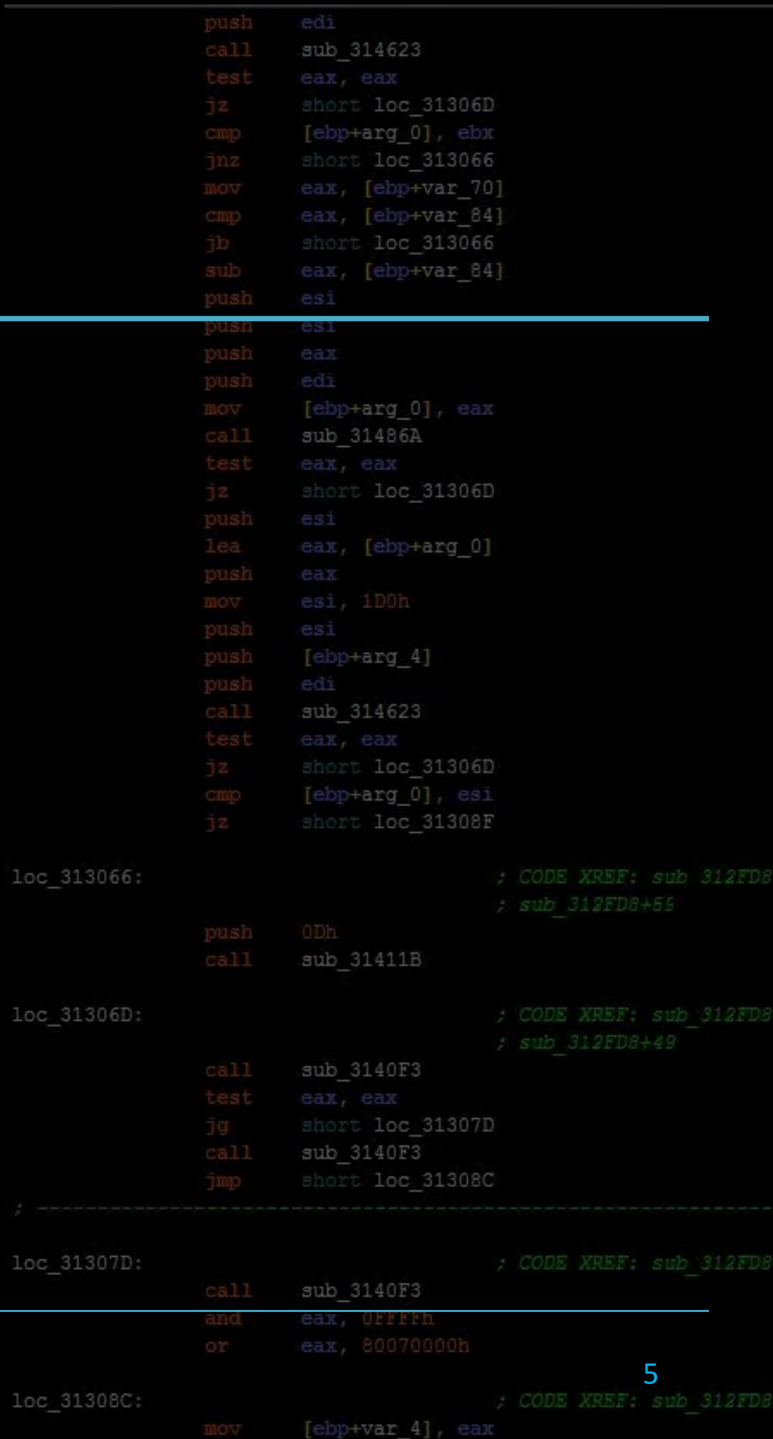
Misc Concepts

- There’s a lot of smaller bits and pieces to this class that are important, but too small to warrant their own lectures

- Also, this lecture should have come before spring break but got displaced till now

Lecture Overview

- Miscellaneous Concepts
  - Integers in C
  - Uninitialized data
  - Structs
  - File Descriptors
  - Stack Cookies



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Integers in C

- We haven't even mentioned signedness yet

```c
int var1 = 0;
unsigned int var2 = 0;
```

Integers in C

- We haven’t even mentioned signedness yet

```
int var1 = 0;
unsigned int var2 = 0;
```

What’s the difference between an int and an unsigned int?

Signed Integers

- A signed integer can be interpreted as positive or negative
- `int`
  - range: –2,147,483,648 to 2,147,483,647

Unsigned Integers

- An unsigned integer is only ever zero and up

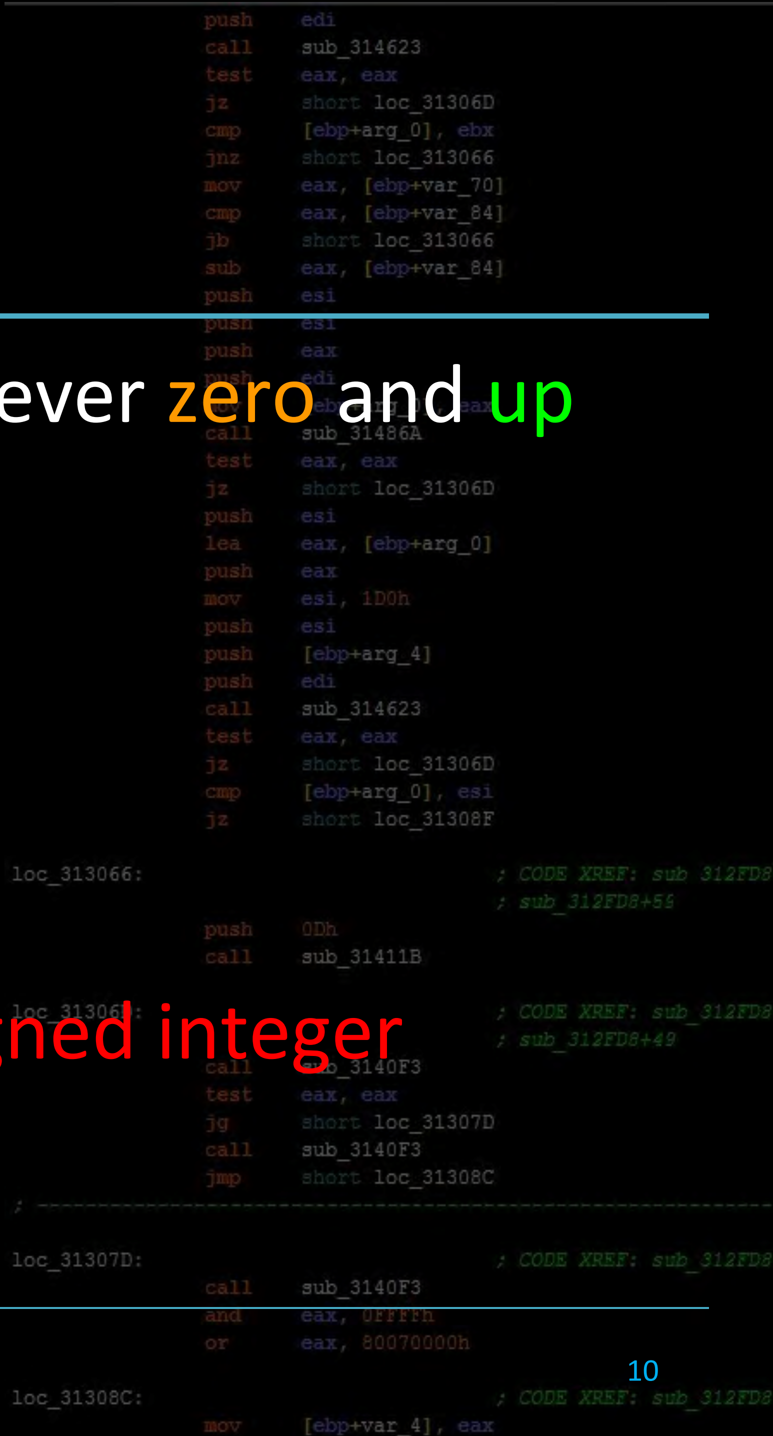
- `unsigned int`
  - range: 0 to 4,294,967,295

Unsigned Integers

- An unsigned integer is only ever zero and up

- `unsigned int`
  - range: 0 to 4,294,967,295

Twice the range of a signed integer



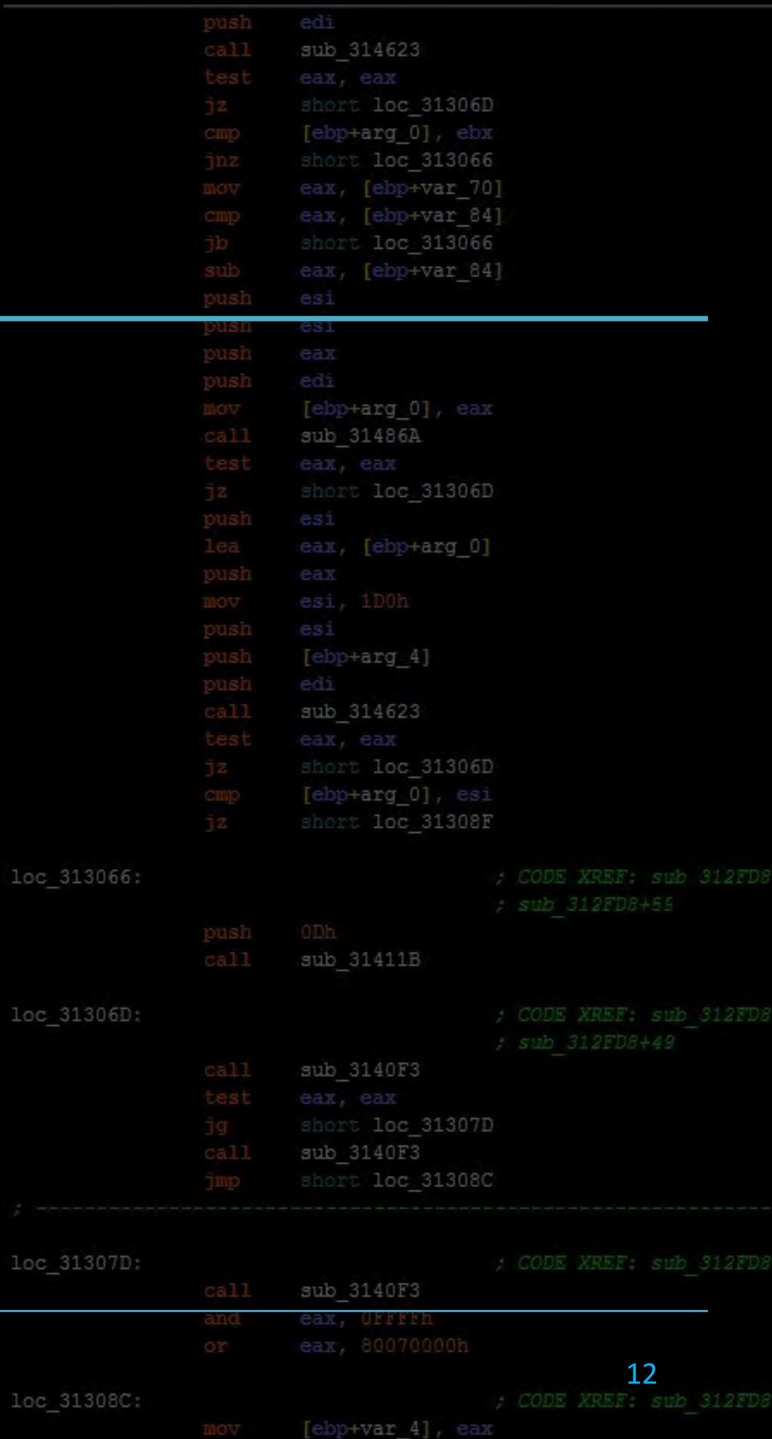
Signedness Naming

- The name signed or unsigned comes from whether or not the type can carry a sign (+/-)

Common Names

- Signed
  - `int`
  - `signed int`
  - `long`

- Unsigned
  - `uint`
  - `unsigned int`
  - `unsigned long`



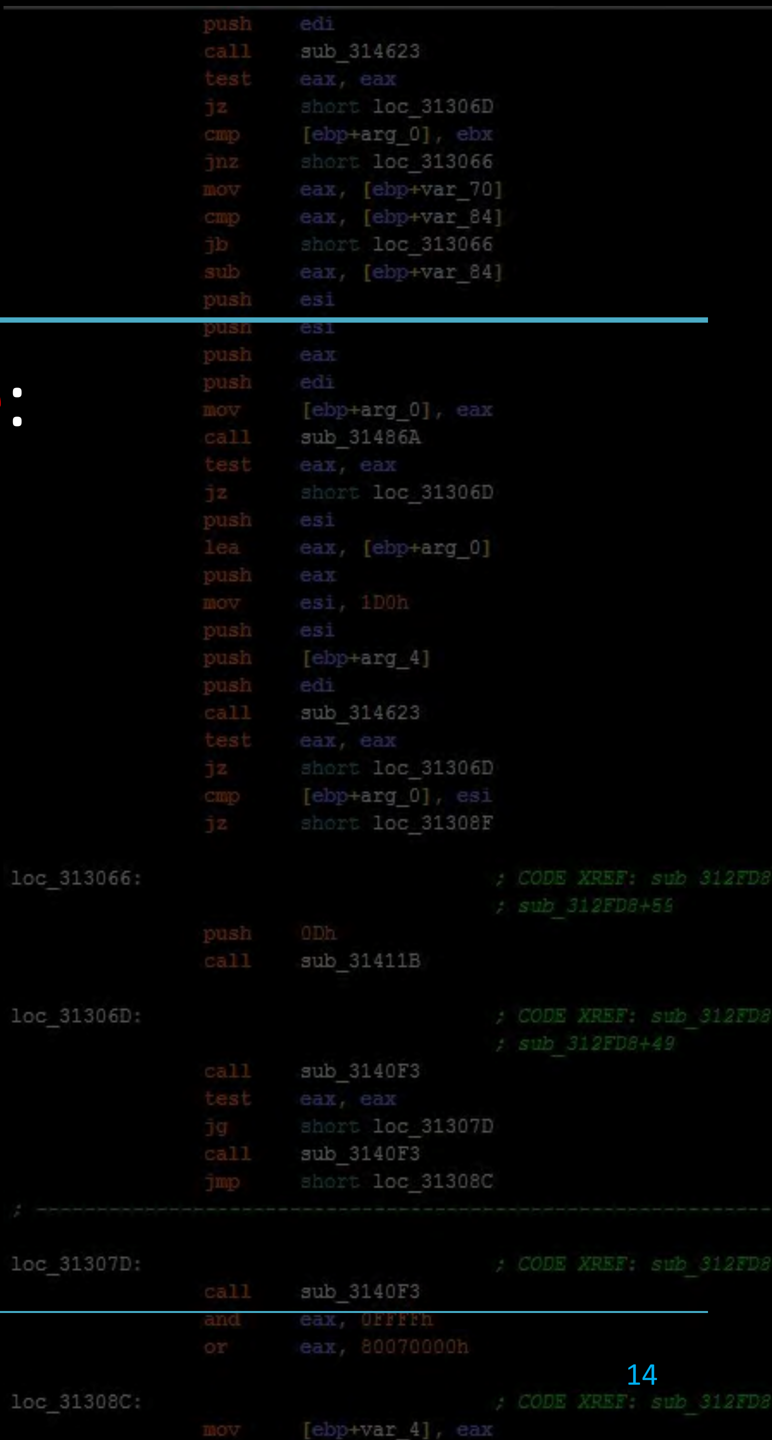
```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+55
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Visualizing Signedness

- A signed int uses the top bit to specify if it is a positive or negative number
  - `0x7FFFFFFF =  2147483647`
    - `01111111111111111111111111111111`
  - `0x80000000 = -2147483647`
    - `10000000000000000000000000000000`
  - `0xFFFFFFFF = -1`
    - `11111111111111111111111111111111`

Two’s Complement

- To make a number negative:
  - Invert all bits
  - Add 1



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

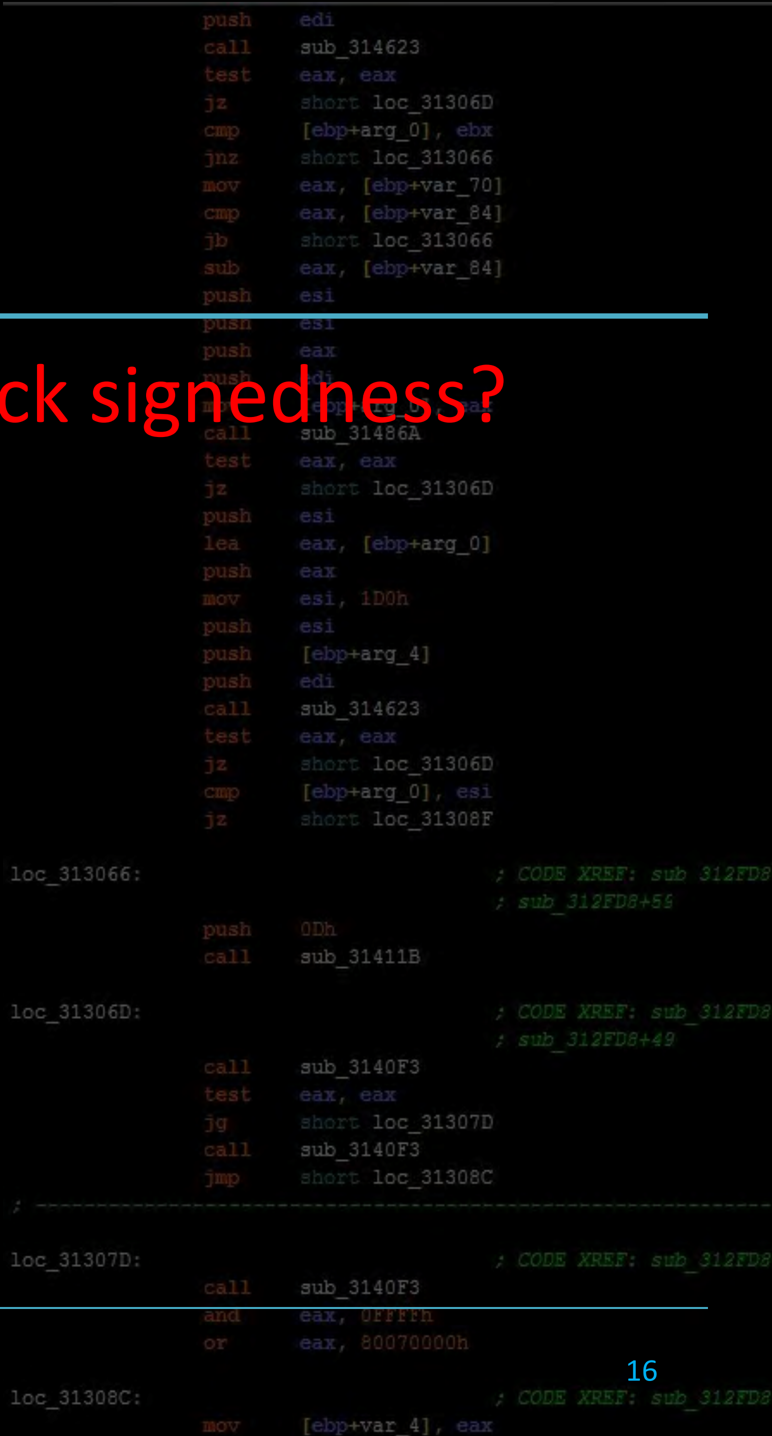
Two’s Complement

- To make a number negative:
  - Invert all bits
  - Add 1

```
eg: 0x00031337
  = 201527
  = 00000000000000110001001100110111
 ~= 11111111111111001110110011001000
 += 11111111111111001110110011001001
  = -201527 (0xFFFCECC9)
```

Tracking Signedness

- How does your program track signedness?



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Tracking Signedness

- How does your program track signedness?
  - Variable types are known at compile time, so signed instructions are compiled in to handle your variable

- You probably didn't realize this, but you can determine integer types at the assembly level

Signed instructions

- Some common signed instructions
  - IDIV - Signed divide
  - IMUL - Signed multiply
  - SAL - Shift left, preserve sign
  - SAR - Shift right, preserve sign
  - MOVSX - Move, sign extend
  - JL - Jump if less
  - JLE - Jump if less or equal
  - JG - Jump if greater
  - JGE - Jump if greater or equal

Unsigned instructions

- Some common unsigned instructions

– DIV	- Unsigned divide
– MUL	- Unsigned multiply
– SHL	- Shift left
– SHR	- Shift right
– MOVZX	- Move, zero extend
– JB	- Jump if below
– JBE	- Jump if below or equal
– JA	- Jump if above
– JAE	- Jump if above or equal

Minimum Size

- Minimum sizes

char	8 bits
short	16 bits
int	16 bits
long	32 bits
long long	64 bits

- These are MINIMUM sizes, can vary from system to system!

Fixed Sizes

- Fixed size format
  - int[# of bits]_t
  - uint[# of bits]_t

- eg int8_t, uint16_t, int32_t

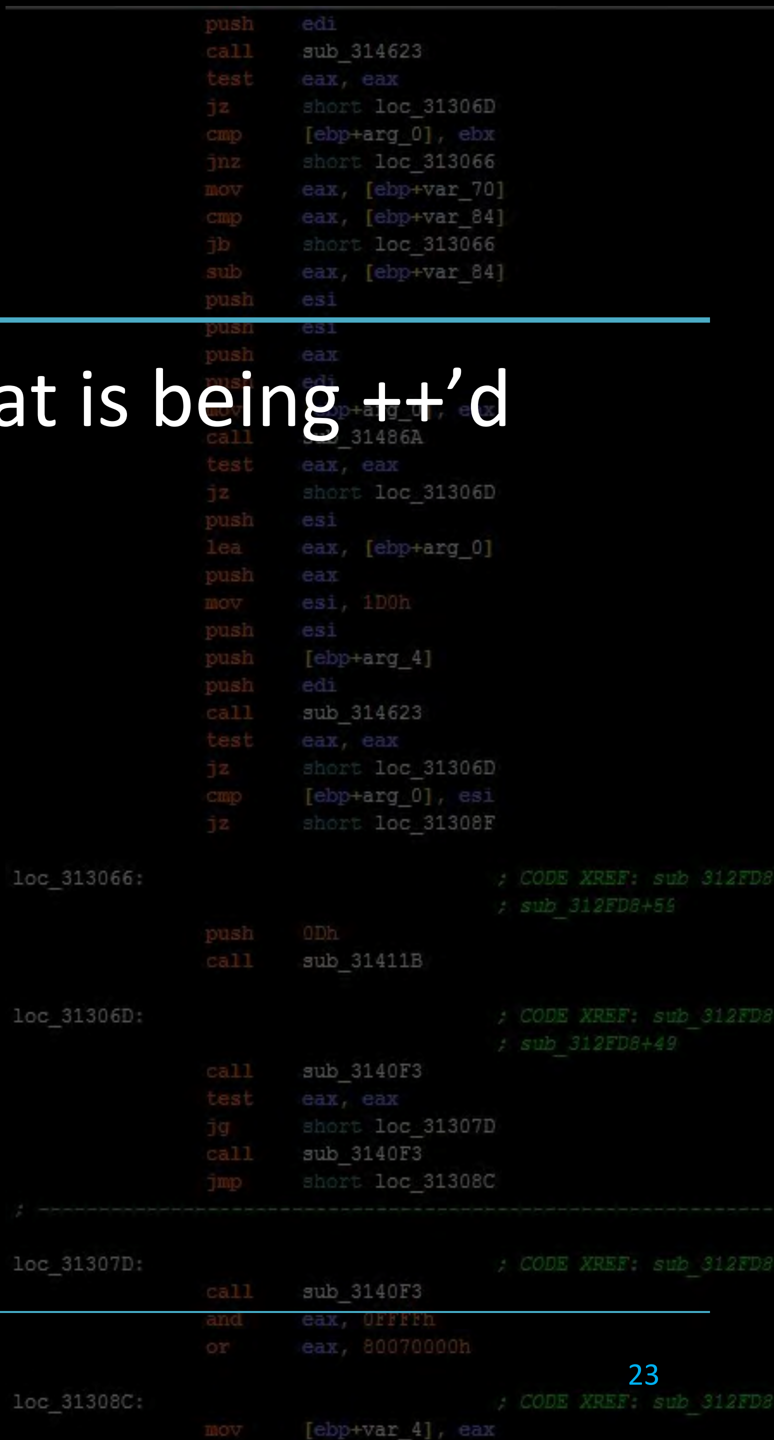
- Guaranteed size across systems
  - Defined in stdint.h
  - Also check out limits.h

Integer Overflows

- Imagine a simple uint8_t that is being ++'d
  - 0x00
  - 0x01
  - 0x02
  - ...

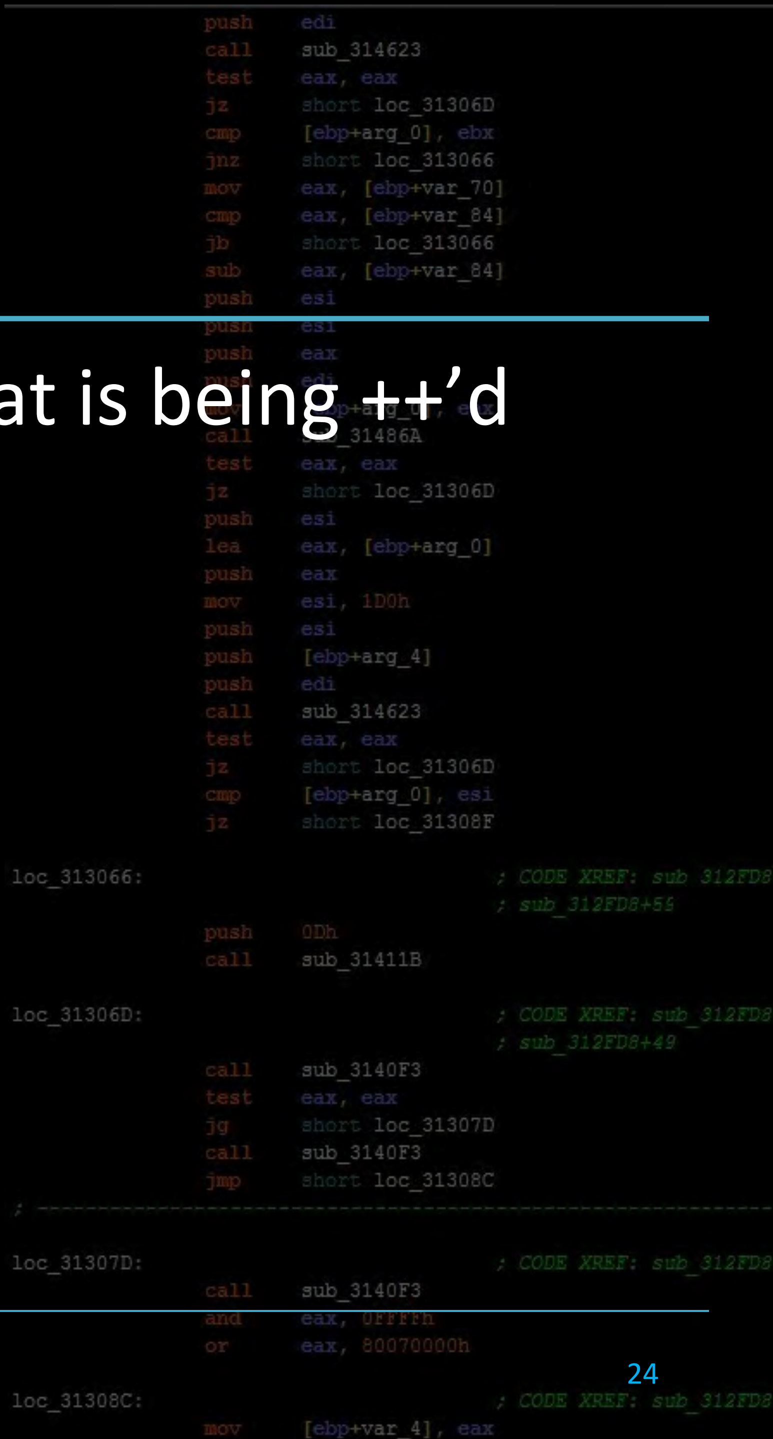
Integer Overflows

- Imagine a simple uint8_t that is being ++'d
  - `0x00`
  - `0x01`
  - `0x02`
  - `...`
  - `0xFE`
  - `0xFF`
  - `????`



```
            push    edi
            call    sub_314623
            test    eax, eax
            jz      short loc_31306D
            cmp     [ebp+arg_0], ebx
            jnz     short loc_313066
            mov     eax, [ebp+var_70]
            cmp     eax, [ebp+var_84]
            jb      short loc_313066
            sub     eax, [ebp+var_84]
            push    esi
            push    esi
            push    eax
            [illegible]
            [illegible]
            call    sub_31486A
            test    eax, eax
            jz      short loc_31306D
            push    esi
            lea     eax, [ebp+arg_0]
            push    eax
            mov     esi, 1D0h
            push    esi
            push    [ebp+arg_4]
            push    edi
            call    sub_314623
            test    eax, eax
            jz      short loc_31306D
            cmp     [ebp+arg_0], esi
            jz      short loc_31308F

loc_313066:                              ; CODE XREF: sub_312FD8
                                         ; sub_312FD8+59
            push    0Dh
            call    sub_31411B

loc_31306D:                              ; CODE XREF: sub_312FD8
                                         ; sub_312FD8+49
            call    sub_3140F3
            test    eax, eax
            jg      short loc_31307D
            call    sub_3140F3
            jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                              ; CODE XREF: sub_312FD8
            call    sub_3140F3
            and     eax, 0FFFFh
            or      eax, 80070000h

loc_31308C:                              ; CODE XREF: sub_312FD8
            mov     [ebp+var_4], eax
```

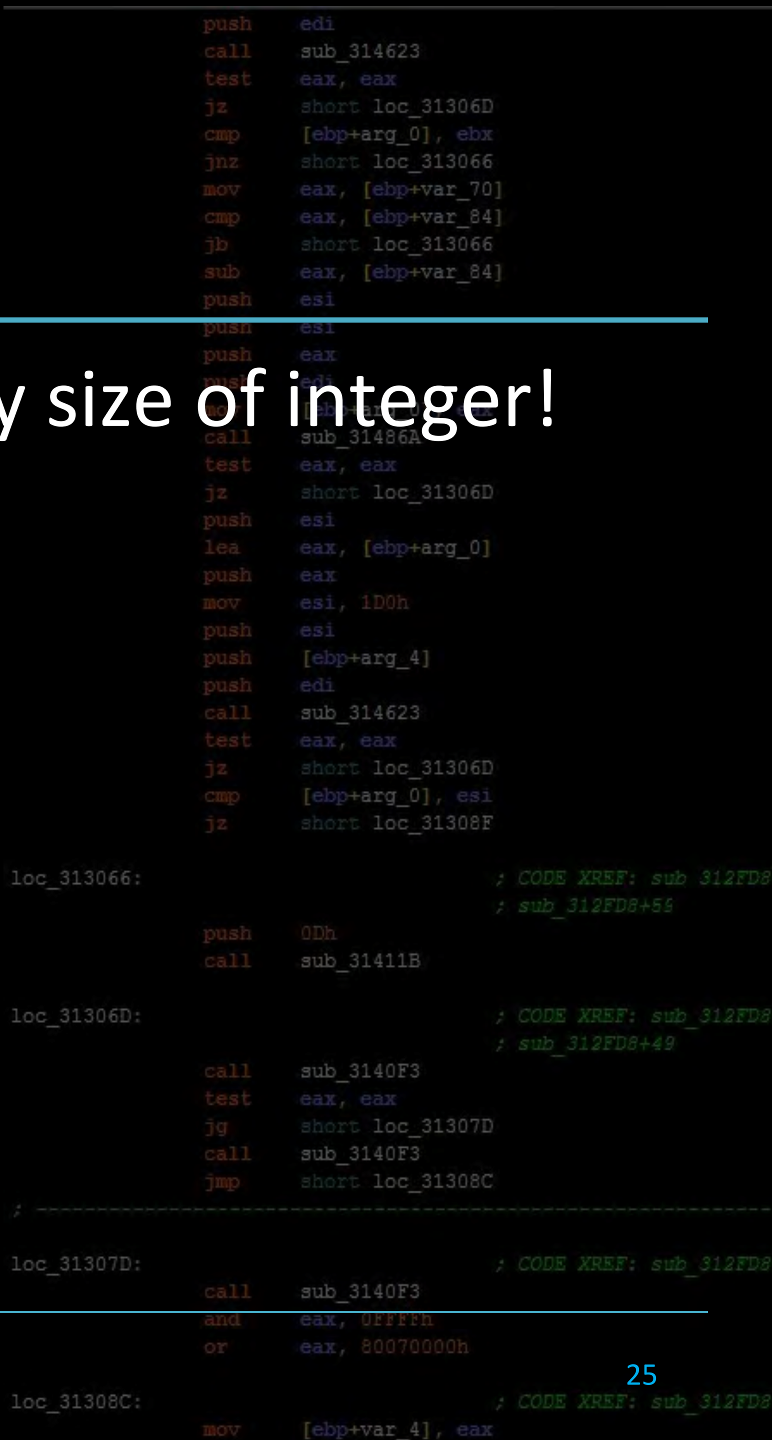
Integer Overflows



- Imagine a simple uint8_t that is being ++’d
  - 0x00
  - 0x01
  - 0x02
  - ...
  - 0xFE
  - 0xFF
  - 0x00 <-- overflows!
  - 0x01

Integer Overflows

- This obviously applies to any size of integer!
  - 0xFFFFFFFD
  - 0xFFFFFFFE
  - 0xFFFFFFFF
  - 0x00000000
  - 0x00000001
  - 0x00000002



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                [illegible]
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Integer Overflows

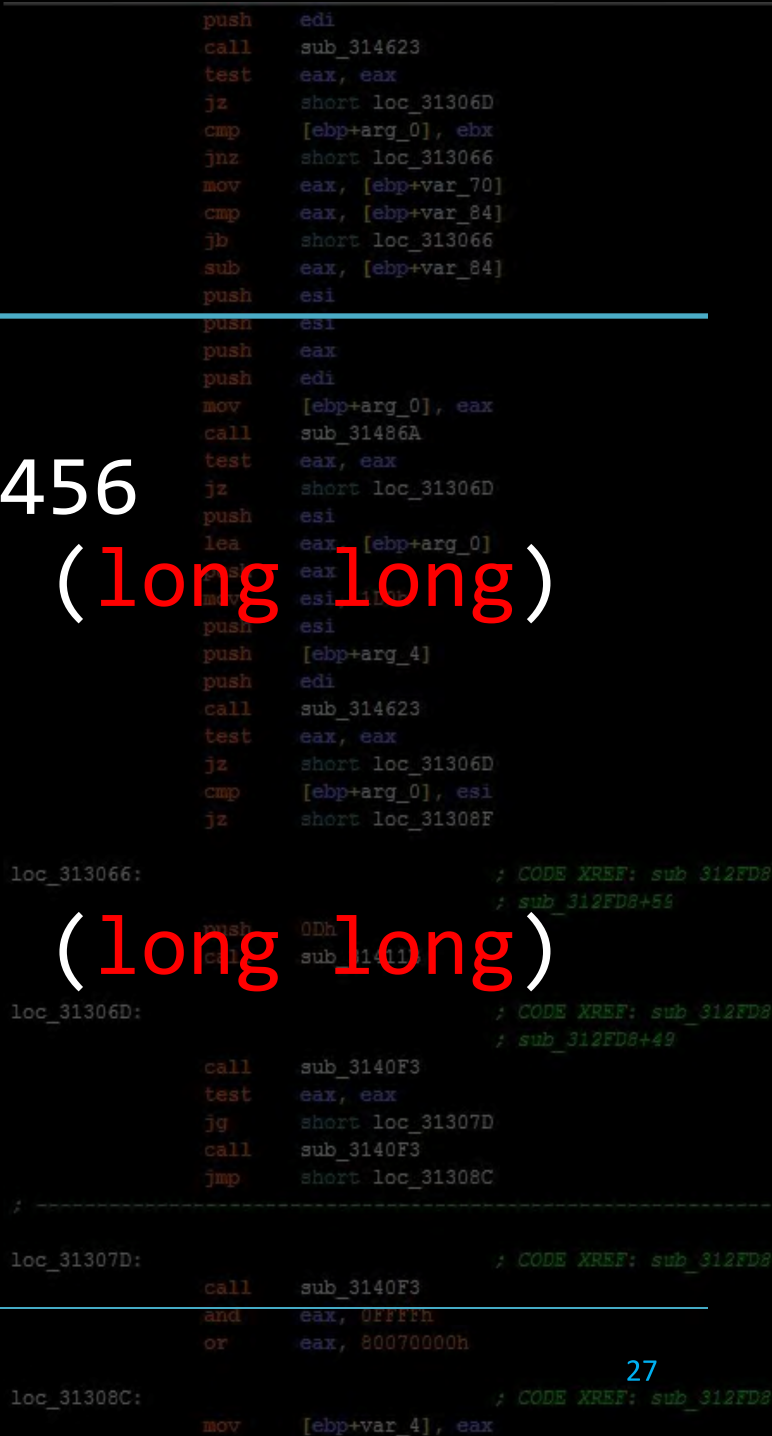
- Don’t forget multiplying!

```
0x00120000 * 0x00123456
= 0x00000147AE0C0000 (long long)
= 0xAE0C0000 (long)
```

Integer Overflows

- Don’t forget multiplying!

```
0x00120000 * 0x00123456
= 0x00000147AE0C0000 (long long)
= 0xAE0C0000 (long)
or
0x40000123 * 4
= 0x000000010000048C (long long)
= 0x0000048C (long)
```



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi,
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_3141

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

0-loop_exerpt.c

```
short int bytesRec = 0;
char buf[SOMEBIGNUM];

while(bytesRec < MAXGET)
  bytesRec += getFromInput(buf+bytesRec);
```

(https://www.owasp.org/index.php/Integer_overflow)

0-loop_exerpt.c Solution

```
short int bytesRec = 0;
char buf[SOMEBIGNUM];

while(bytesRec < MAXGET)
  bytesRec += getFromInput(buf+bytesRec);
```

If getFromInput() returns enough bytes to overflow bytesRec, the loop will continue and overflow buf

1-OpenSSH3.3_exerpt.c

```
nresp = packet_get_int();

if(nresp > 0)
{
  response = xmalloc(nresp*sizeof(char*));
  for (i = 0; i < nresp; i++)
    response[i] = packet_get_string(NULL);
}
```

(https://www.owasp.org/index.php/Integer_overflow)

1-OpenSSH3.3_exerpt.c Solution

```
nresp = packet_get_int();

if(nresp > 0)
{
  response = xmalloc(nresp*sizeof(char*));
...
```

nresp is a signed int, what happens when packet_get_int() returns INT_MAX/sizeof (char*)?

1-OpenSSH3.3_exerpt.c Solution

```
nresp = packet_get_int();

if(nresp > 0)
{
  response = xmalloc(nresp*sizeof(char*));
...
```

nresp is a signed int. what happens when packet_get_int() returns INT_MAX/sizeof (char*)?

Probably allocates a 0 size buffer

2-variable-length_exerpt.c

```c
char* processNext(char* strm) {
  char buf[512];
  short len = *(short*) strm;
  strm += sizeof(len);
  if (len <= 512) {
    memcpy(buf, strm, len);
    process(buf);
    return strm + len;
  } else {
    return -1;
  }
}
```

(https://www.owasp.org/index.php/Integer_overflow)

2-variable-length_exerpt.c Solution

```
char* processNext(char* strm) {
  char buf[512];
  short len = *(short*) strm;
  strm += sizeof(len);
  if (len <= 512) {
    memcpy(buf, strm, len);
...
```

len is signed short, any negative len will pass the if-statement

2-variable-length_exerpt.c - Solution

```
char* processNext(char* strm) {
  char buf[512];
  short len = *(short*) strm;
  strm += sizeof(len);
  if (len <= 512) {
    memcpy(buf, strm, len);
...
```

len is signed short, any negative len will pass the if-statement

memcpy takes an unsigned int

underflow -> large copy -> stack corruption

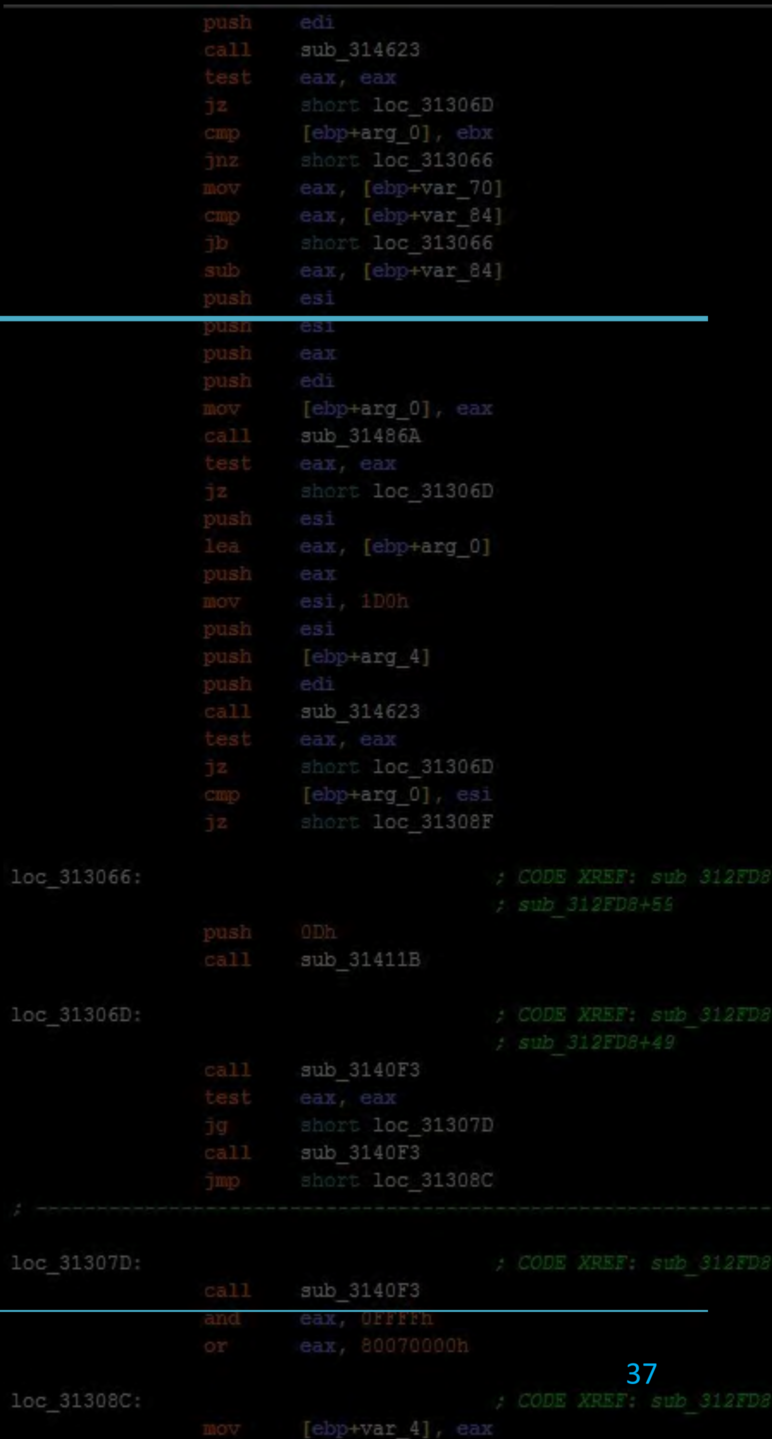
Integer Problems

- It’s very common to see modern bugs stem from integer confusion and misuse

- Know when to use signed/unsigned!

Lecture Overview

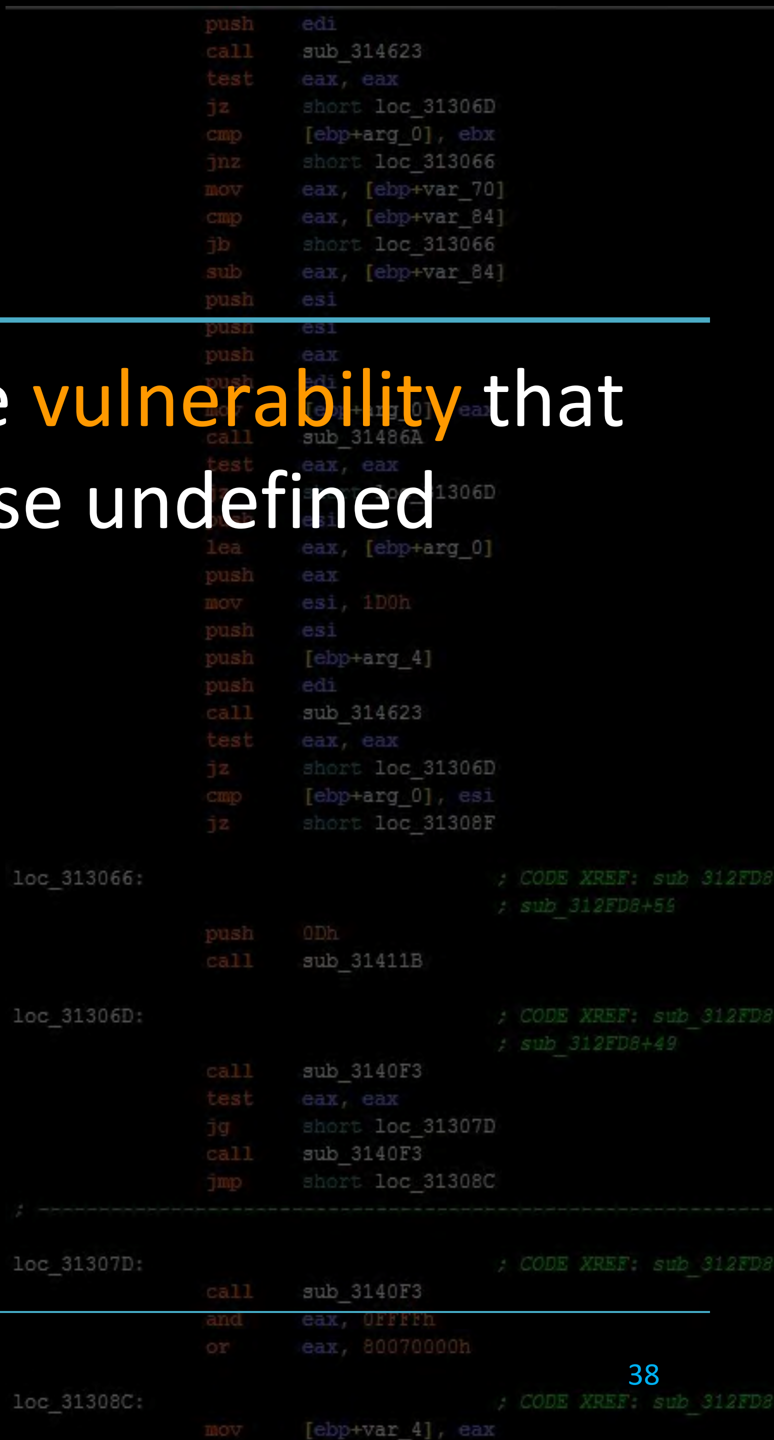
- Miscellaneous Concepts
  - Integers in C
  - Uninitialized data
  - Structs
  - File Descriptors
  - Stack Cookies



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Uninitialized Data

- Uninitialized data is a subtle vulnerability that can leak information or cause undefined behavior in an application

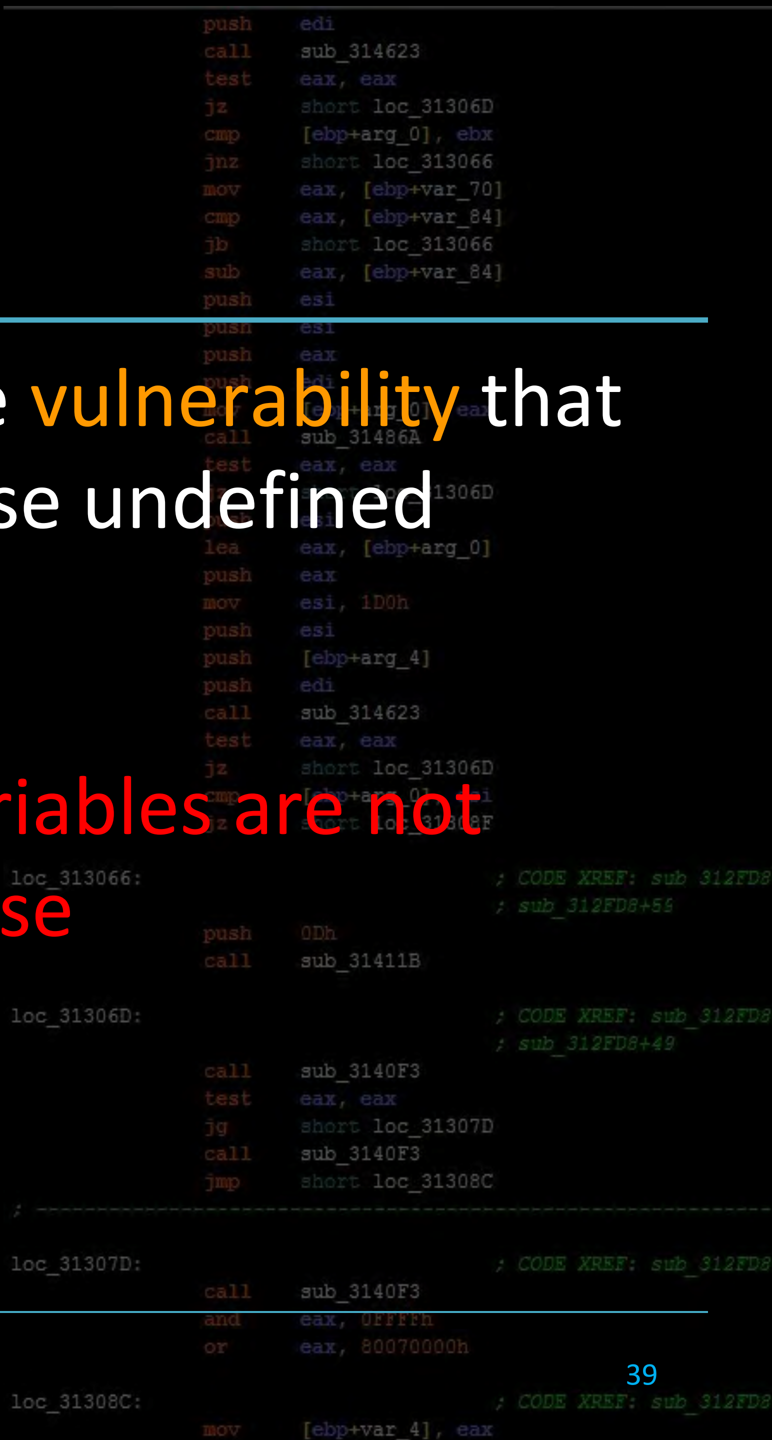


```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Uninitialized Data

- Uninitialized data is a subtle vulnerability that can leak information or cause undefined behavior in an application

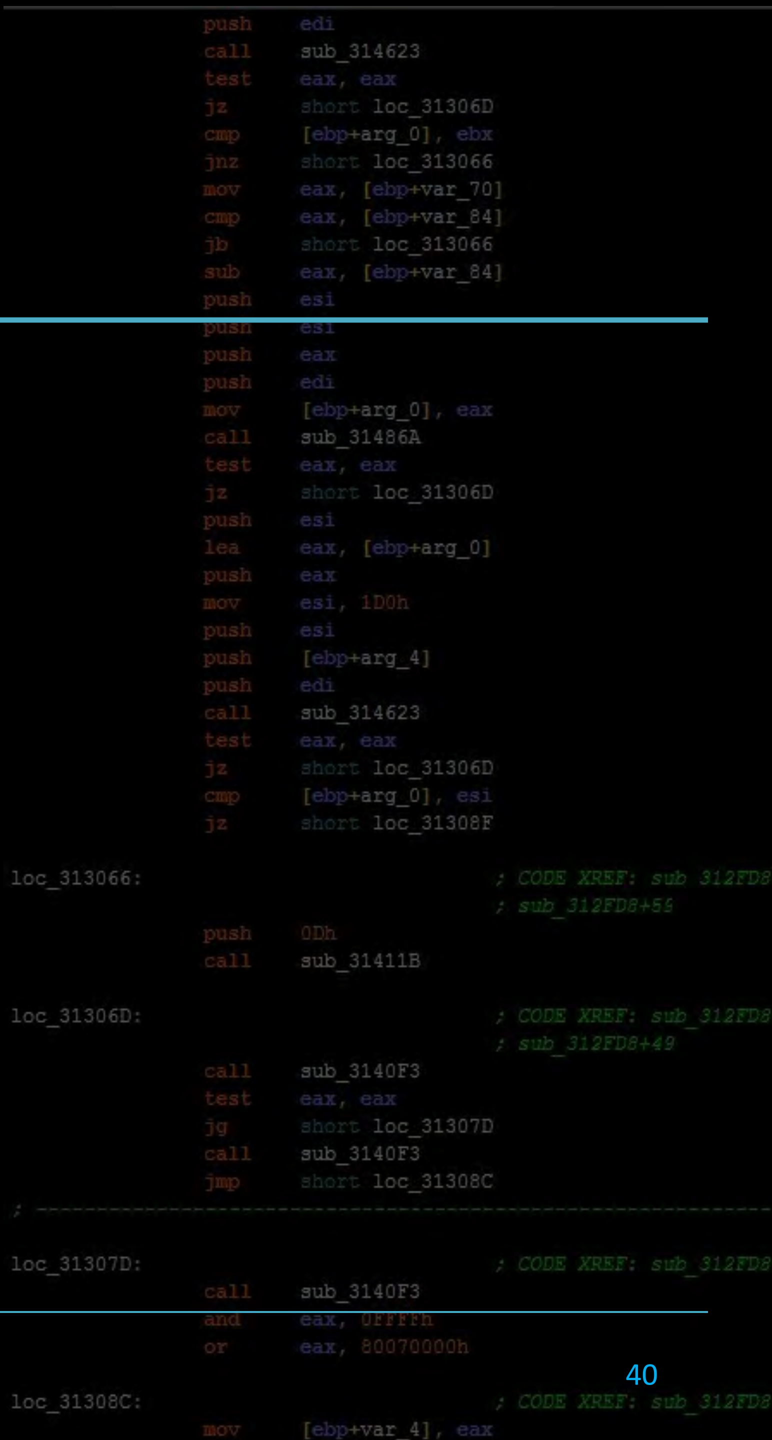
- The bug manifests when variables are not properly initialized before use



```
           push    edi
           call    sub_314623
           test    eax, eax
           jz      short loc_31306D
           cmp     [ebp+arg_0], ebx
           jnz     short loc_313066
           mov     eax, [ebp+var_70]
           cmp     eax, [ebp+var_84]
           jb      short loc_313066
           sub     eax, [ebp+var_84]
           push    esi
           push    esi
           push    eax
           push    edi
           mov     [ebp+arg_0], eax
           call    sub_31486A
           test    eax, eax
           jz      short loc_31306D
           push    esi
           lea     eax, [ebp+arg_0]
           push    eax
           mov     esi, 1D0h
           push    esi
           push    [ebp+arg_4]
           push    edi
           call    sub_314623
           test    eax, eax
           jz      short loc_31306D
           cmp     [ebp+arg_0], esi
           jz      short loc_31308F

loc_313066:                            ; CODE XREF: sub_312FD8
                                       ; sub_312FD8+59
           push    0Dh
           call    sub_31411B

loc_31306D:                            ; CODE XREF: sub_312FD8
                                       ; sub_312FD8+49
           call    sub_3140F3
           test    eax, eax
           jg      short loc_31307D
           call    sub_3140F3
           jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                            ; CODE XREF: sub_312FD8
           call    sub_3140F3
           and     eax, 0FFFFh
           or      eax, 80070000h

loc_31308C:                            ; CODE XREF: sub_312FD8
           mov     [ebp+var_4], eax
```

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = ‘A’;
    i++;
   }

   return 0;
}
```



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = ‘A’;
    i++;
   }

   return 0;
}
```

i is never initialized to anything

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = 'A';
    i++;
   }

   return 0;
}
```

i is never initialized to anything

So what is i?

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = ‘A’;
    i++;
   }

   return 0;
}
```

i is never initialized to anything

So what is i?

The variable will be whatever data happens to be left on the stack frame from a previous function call of any sort

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = ‘A’;
    i++;
   }

   return 0;
}
```

So can you exploit this function?

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = ‘A’;
    i++;
   }

   return 0;
}
```

So can you exploit this function?

Probably.

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = ‘A’;
    i++;
   }

   return 0;
}
```

So can you exploit this function?

Probably.

If you can control i, you can reliably write 20 A’s anywhere on the stack.

Spot the Bug

```
int do_work()
{
  int i;
  char buf[20];

  while(i < 20){
    buf[i] = 'A';
    i++;
   }

   return 0;
}
```

So can you exploit this function?

Probably.

If you can control i, you can reliably write 20 A's anywhere on the stack.

(do a partial overwrite or corrupt something more meaningful)

More Subtle

```
#include <stdio.h>
#include <stdlib.h>
void take_ptr( int *bptr ) {
    printf( “%lx”, *bptr );
}
int main( int argc, char **argv ) {
    int b;
    take_ptr( &b );
    printf( “%lx”, b );
}
```

(https://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Flake.pdf)

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

#-uninitialized_data.c

- on warzone
- http://www.exploit-db.com/docs/99.pdf

#-uninitialized_data.c - Solution

- char *err, *mesg;
- easy to exploit with ASLR off

Uninitialized Data

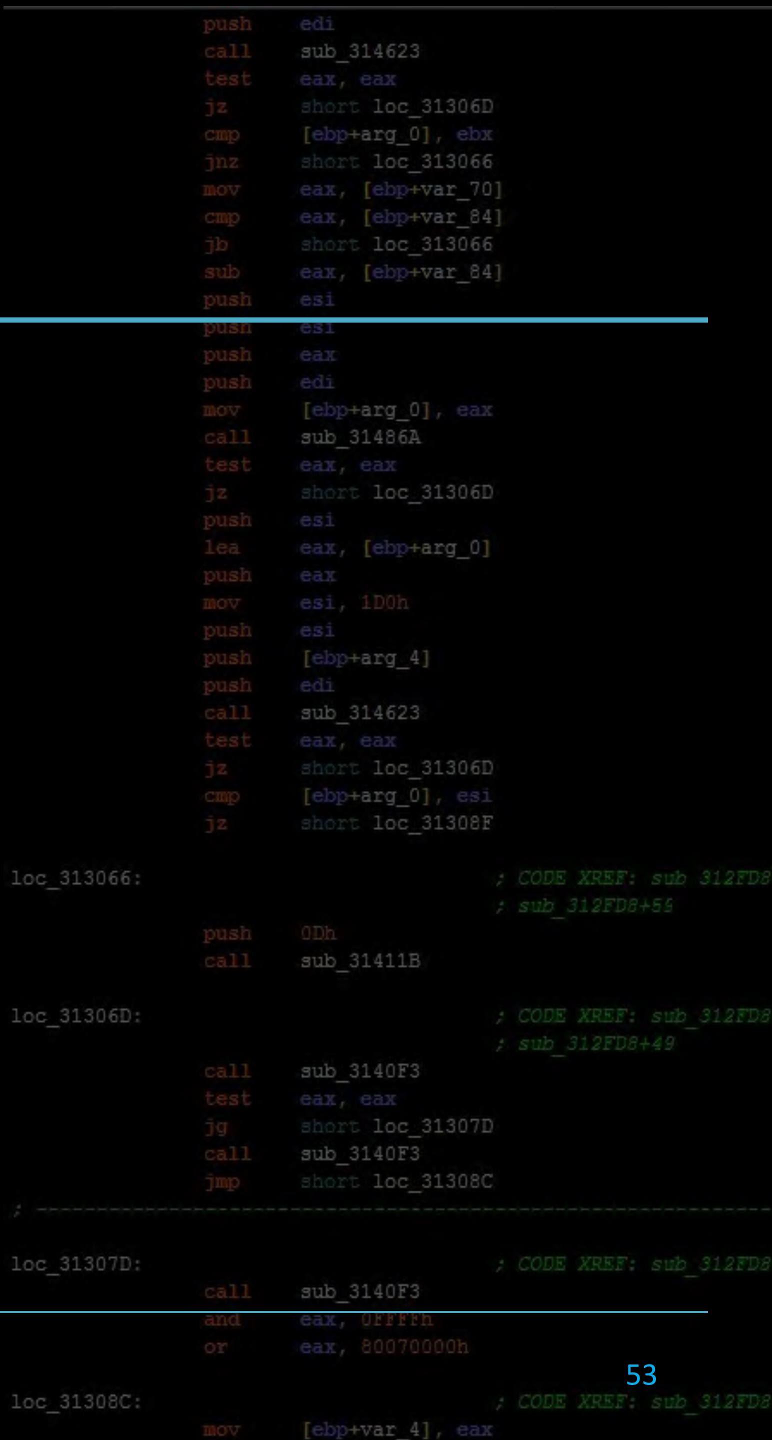
- Keep in mind this can happen on the heap too!
- There's no knowing what's going to be on the other end of the pointer you get back from something like malloc()

Uninitialized Data

- Pretty common in amateur development, smaller software projects, CTF problems
- Less common in industry as this is an easy issue to detect statically (in source and binary)

Lecture Overview

- Miscellaneous Concepts
  - Integers in C
  - Uninitialized data
  - Structs
  - File Descriptors
  - Stack Cookies



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

#-struct-heap2.c

```c
while(1) {
    if(fgets(line, sizeof(line), stdin) == NULL) break;
    if(strncmp(line, "auth ", 5) == 0) {
      auth = malloc(sizeof(auth));
      memset(auth, 0, sizeof(auth));
      if(strlen(line + 5) < 31)
        strcpy(auth->name, line + 5);
    }
    if(strncmp(line, "service", 6) == 0)
      service = strdup(line + 7);
    if(strncmp(line, "login", 5) == 0) {
      if(auth->auth)
        printf("you have logged in already!\n");
      else
        printf("please enter your password\n");
    }
  }
```

(https://exploit-exercises.com/protostar/heap2/)

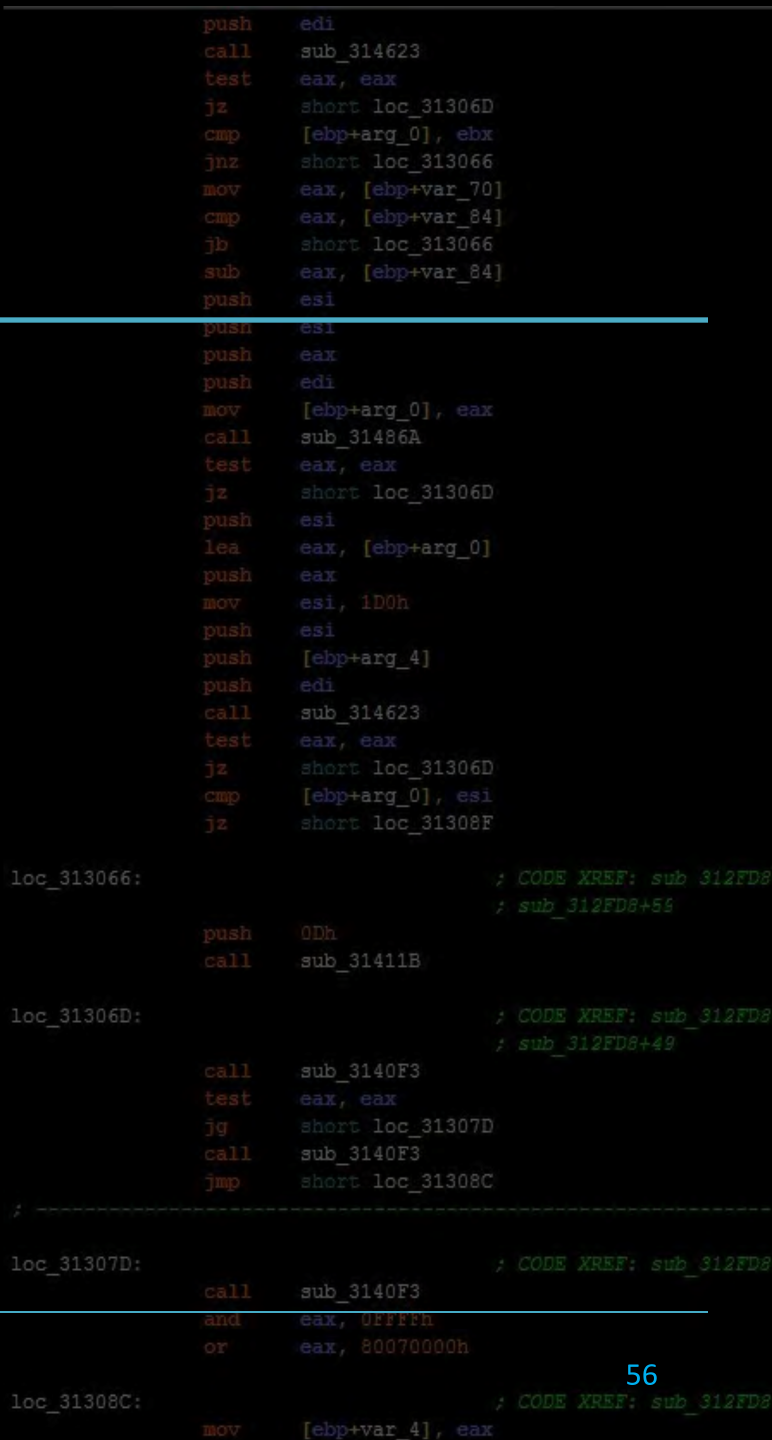
#-struct-heap2.c

```
...
        if(strncmp(line, "auth ", 5) == 0) {
            auth = malloc(sizeof(auth));
            memset(auth, 0, sizeof(auth));
            if(strlen(line + 5) < 31)
                strcpy(auth->name, line + 5);
        }
        if(strncmp(line, "service", 6) == 0)
            service = strdup(line + 7);
...
```

- `sizeof(auth)` doesn’t return size of struct
- In this case, it returns 16
- `service` makes new buffer over `name`/`auth`
- Use `sizeof(struct auth)`

Lecture Overview

- Miscellaneous Concepts
  - Integers in C
  - Uninitialized data
  - Structs
  - File Descriptors
  - Stack Cookies



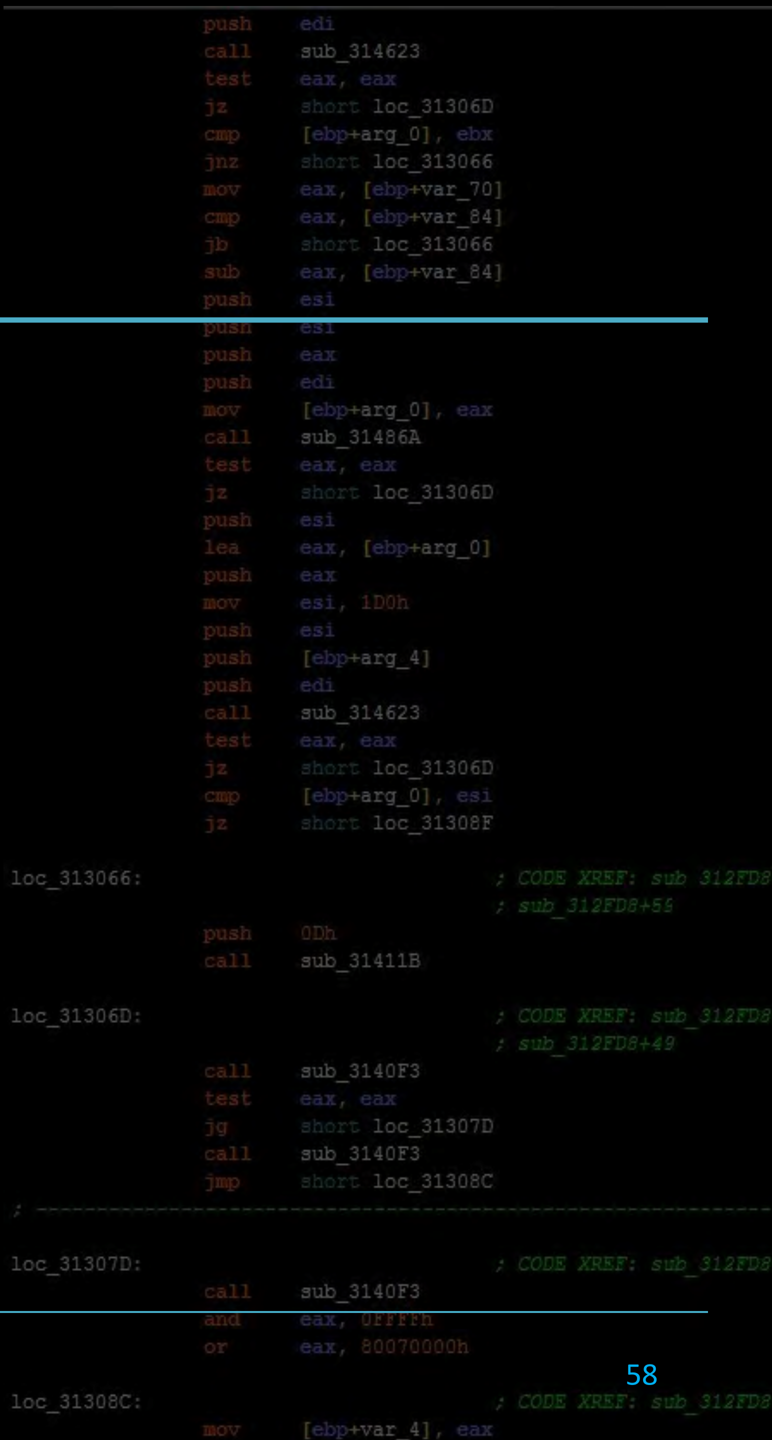
```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

File Descriptors

- In Linux, everything is a file
- When opening a file, it gets a number
- You use some frequently
  - 0 STDIN
  - 1 STDOUT
  - 2 STDERR
- open returns a file descriptor

#-fd.c

- On the warzone



```
                 push    edi
                 call    sub_314623
                 test    eax, eax
                 jz      short loc_31306D
                 cmp     [ebp+arg_0], ebx
                 jnz     short loc_313066
                 mov     eax, [ebp+var_70]
                 cmp     eax, [ebp+var_84]
                 jb      short loc_313066
                 sub     eax, [ebp+var_84]
                 push    esi
                 push    esi
                 push    eax
                 push    edi
                 mov     [ebp+arg_0], eax
                 call    sub_31486A
                 test    eax, eax
                 jz      short loc_31306D
                 push    esi
                 lea     eax, [ebp+arg_0]
                 push    eax
                 mov     esi, 1D0h
                 push    esi
                 push    [ebp+arg_4]
                 push    edi
                 call    sub_314623
                 test    eax, eax
                 jz      short loc_31306D
                 cmp     [ebp+arg_0], esi
                 jz      short loc_31308F

loc_313066:                              ; CODE XREF: sub_312FD8
                                         ; sub_312FD8+59
                 push    0Dh
                 call    sub_31411B

loc_31306D:                              ; CODE XREF: sub_312FD8
                                         ; sub_312FD8+49
                 call    sub_3140F3
                 test    eax, eax
                 jg      short loc_31307D
                 call    sub_3140F3
                 jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                              ; CODE XREF: sub_312FD8
                 call    sub_3140F3
                 and     eax, 0FFFFh
                 or      eax, 80070000h

loc_31308C:                              ; CODE XREF: sub_312FD8
                 mov     [ebp+var_4], eax
```

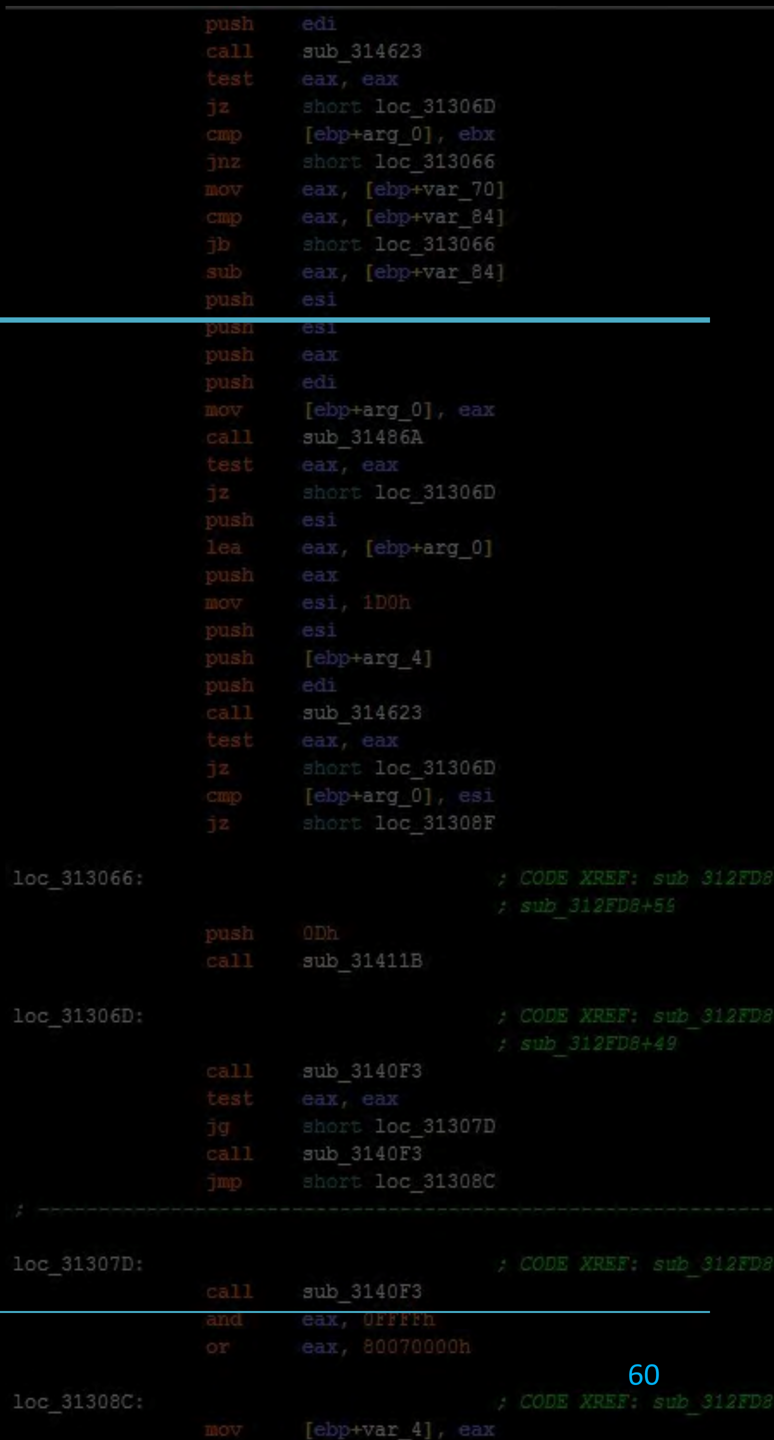
#-fd.c - Solution

- file descriptors from parent processes are inherited by children

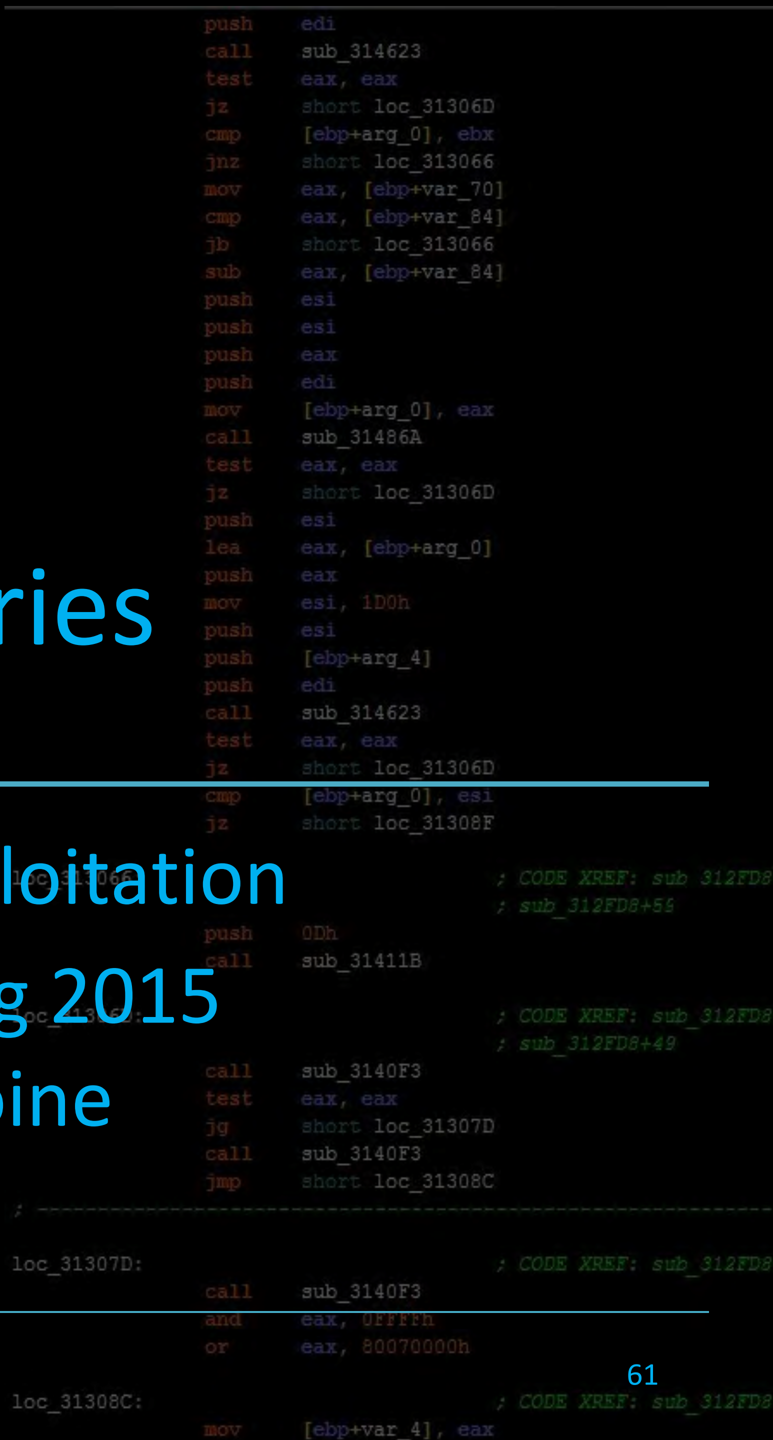
- fd to password file wasn’t closed

Lecture Overview

- Miscellaneous Concepts
  - Integers in C
  - Uninitialized data
  - Structs
  - File Descriptors
  - Stack Cookies



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```



```
        push    edi
        call    sub_314623
        test    eax, eax
        jz      short loc_31306D
        cmp     [ebp+arg_0], ebx
        jnz     short loc_313066
        mov     eax, [ebp+var_70]
        cmp     eax, [ebp+var_84]
        jb      short loc_313066
        sub     eax, [ebp+var_84]
        push    esi
        push    esi
        push    eax
        push    edi
        mov     [ebp+arg_0], eax
        call    sub_31486A
        test    eax, eax
        jz      short loc_31306D
        push    esi
        lea     eax, [ebp+arg_0]
        push    eax
        mov     esi, 1D0h
        push    esi
        push    [ebp+arg_4]
        push    edi
        call    sub_314623
        test    eax, eax
        jz      short loc_31306D
        cmp     [ebp+arg_0], esi
        jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
        push    0Dh
        call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
        call    sub_3140F3
        test    eax, eax
        jg      short loc_31307D
        call    sub_3140F3
        jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
        call    sub_3140F3
        and     eax, 0FFFFh
        or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
        mov     [ebp+var_4], eax
```

Stack Canaries

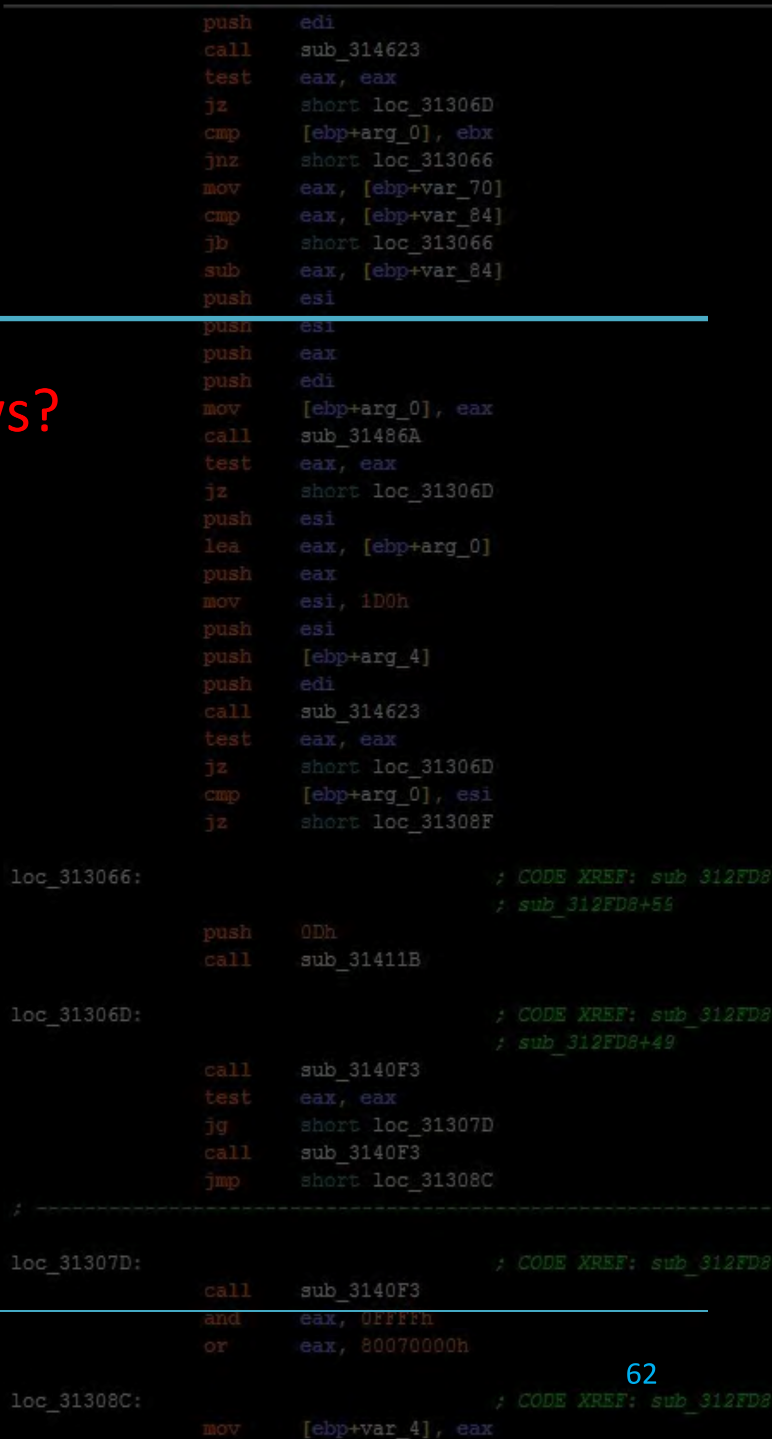
Modern Binary Exploitation

CSCI 4968 - Spring 2015

Sophia D'Antoine

Lecture Overview

1. How do we protect against overflows?
2. Different Types
3. Guarding the Stack
4. Ways to Leak Information
5. When All Else Fails



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Overflow Protections

Before the Overflow (program and compiler)

- program well	strcpy v strncpy v strlcpy
- validate input	ASCII
- static/ dynamic analysis	LLVM or SAT Solvers

After the Overflow (OS level)

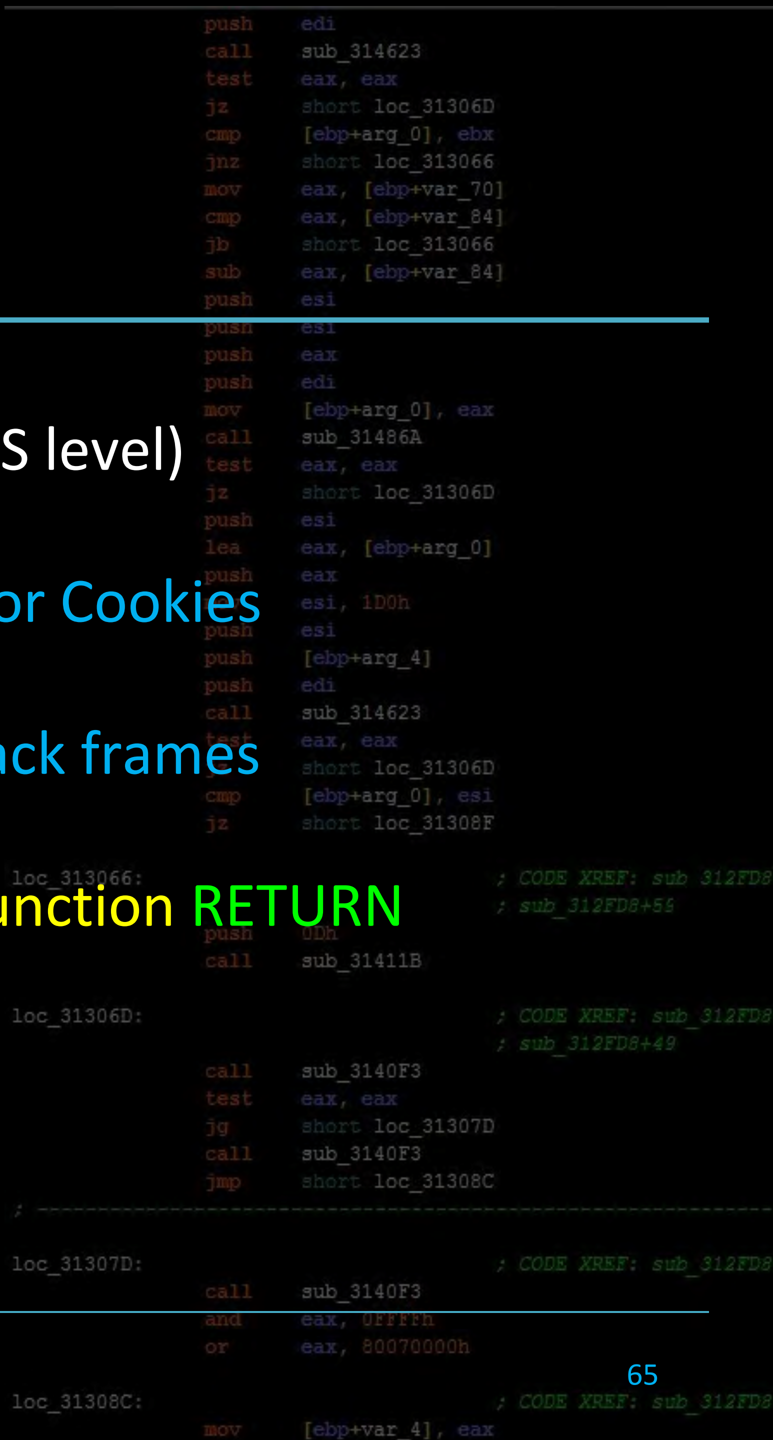
- intercept function calls	Link Libsafe
- turn off execution	NX bit / DEP
- randomize the addresses	ASLR

Avaya Labs ⇒ http://directory.fsf.org/wiki/Libsafe

BUT WAIT
THERE'S MORE!

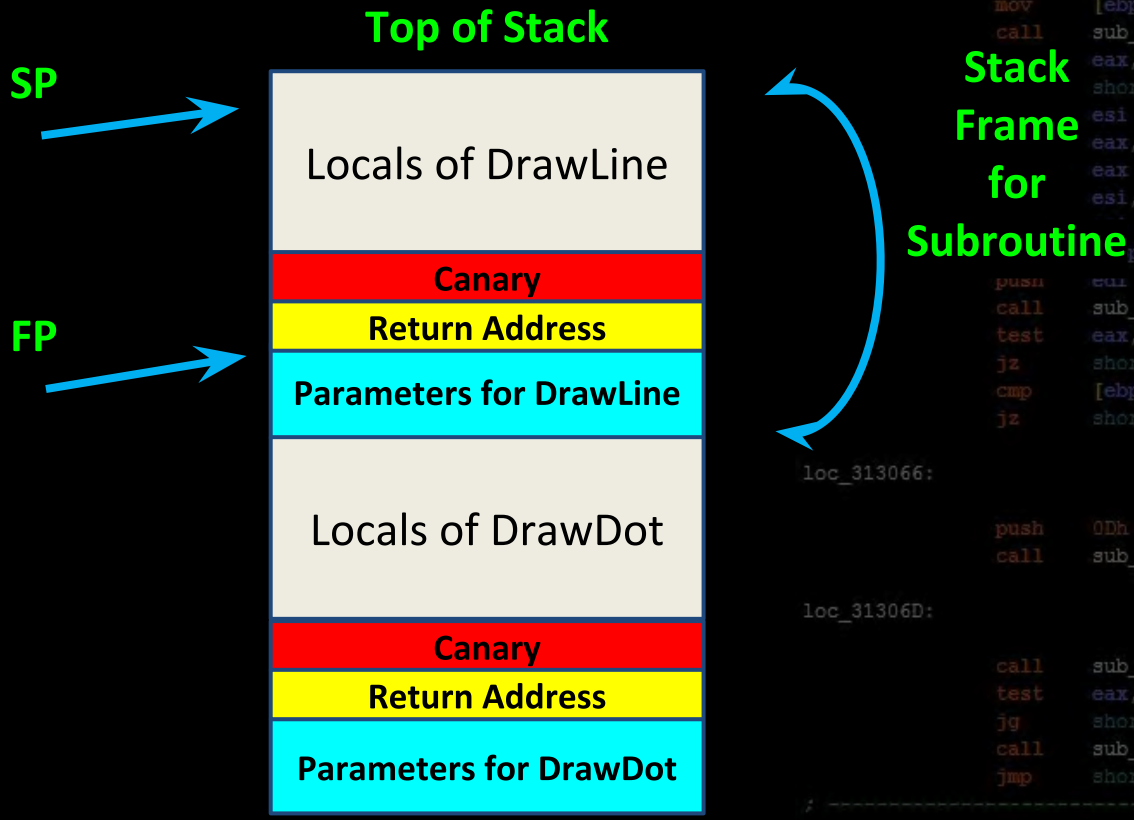
.... Stack Canaries!

+ After the Overflow (Compiler and OS level)
    - sometimes called Stack Guards or Cookies
    - embed random “canaries” in stack frames
    - verify their integrity PRIOR to Function RETURN



```
            push    edi
            call    sub_314623
            test    eax, eax
            jz      short loc_31306D
            cmp     [ebp+arg_0], ebx
            jnz     short loc_313066
            mov     eax, [ebp+var_70]
            cmp     eax, [ebp+var_84]
            jb      short loc_313066
            sub     eax, [ebp+var_84]
            push    esi
            push    esi
            push    eax
            push    edi
            mov     [ebp+arg_0], eax
            call    sub_31486A
            test    eax, eax
            jz      short loc_31306D
            push    esi
            lea     eax, [ebp+arg_0]
            push    eax
                    esi, 1D0h
            push    esi
            push    [ebp+arg_4]
            push    edi
            call    sub_314623
                    eax, eax
                    short loc_31306D
            cmp     [ebp+arg_0], esi
            jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
            push    0Dh
            call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
            call    sub_3140F3
            test    eax, eax
            jg      short loc_31307D
            call    sub_3140F3
            jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
            call    sub_3140F3
            and     eax, 0FFFFh
            or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
            mov     [ebp+var_4], eax
```

.... Stack Canaries!



…. Stack Canaries!

+ What is a canary?
  - its a random integer
  - pushed onto stack after certain triggers are pushed
  - popped off stack and checked before the trigger is read from
  - valued saved as global variable padded by unmapped pages

.... Stack Canaries!

+ Drawbacks
    1. adds overhead (huge cache footprint)
    2. only defends against stack overflows
    3. NULL canaries can potentially be abused
    4. Random canaries can potentially be learned
        a. format string vulns
        b. information leak

Lecture Overview

1. How do we protect against overflows?
2. Different Types
3. Guarding the Stack
4. Ways to Leak Information
5. When All Else Fails

Terminator Canaries

The Canary = 0 (null), newline, linefeed, EOF, -1

- targets string functions
  - they will stop copying at the terminator
- attackers cannot use string functions as the attack vector
- ignores rest of program security

Terminator Canaries

How to Defeat This:

- if input is treated as binary data and not text
- overwrite the canary with its known value, passing the canary check code
    - control information with mismatched values
    - executed soon before the return instruction

Terminator Canaries

Seems like a bad idea, who would use this…..

GCC

“If a random generator can't be used, the protector switches the guard to the terminator canary.”

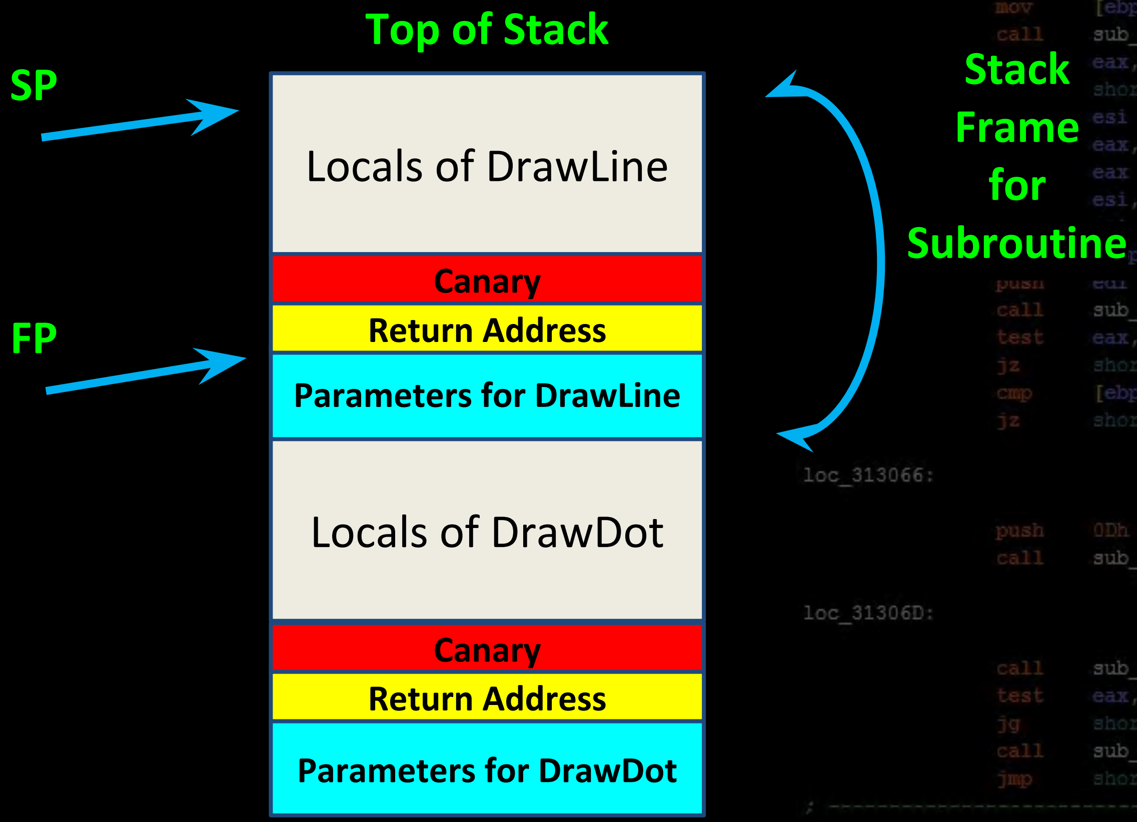
Randomized Canaries

Most popular (GCC uses them)

- random number chosen at program startup
  - attacker must be dynamic
- inserts into every stack frame
- trigger: return addresses
- Some possibilities
  - NULL canaries

- gcc on a typical 32-bit machine
  is ⇒ 4 byte canary

Safeguarding the Return Addresses



Randomized Canaries

GCC

```
-fstack-protector-all
-fstack-protector

    +  char array of 8 bytes or
       more declared on the stack
    +  --param=ssp-buffer-size=N

-fstack-protector-strong

    +  declaration of type or length of local
       arrays
    +  local var addresses or local register
       variables
```

Randomized Canaries

Random XOR Canaries

- They are still random!
- XOR-ed with all or part of the control data
  - if altered, the canary value is immediately invalidated
  - same vulnerabilities as random canaries in reading off stack

http://lwn.net/1999/1111/a/stackguard.html

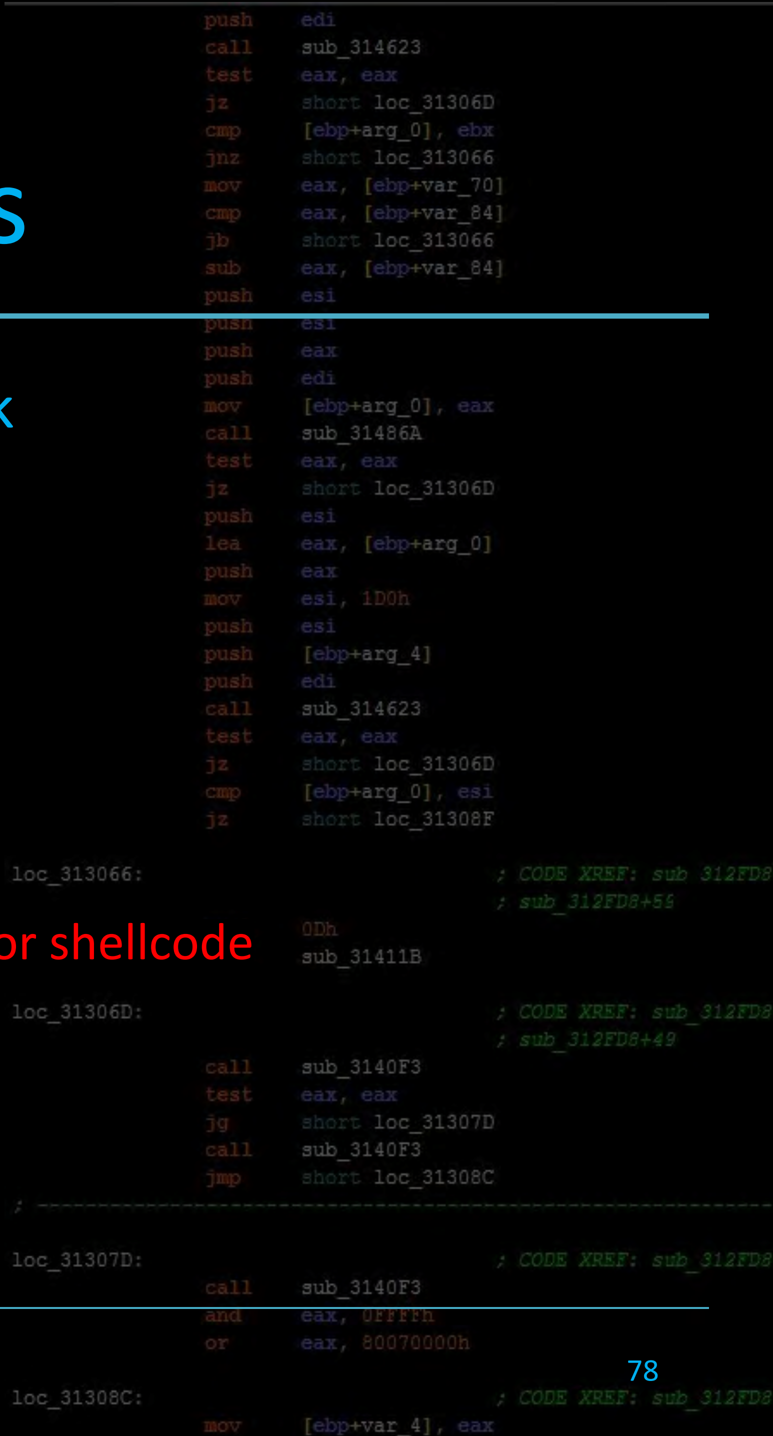
http://deployingradius.com/pscan/stackguard.txt

Random XOR Canaries

- To Bypass: read value from the stack
  - get the canary value
  - the control data
  - the algorithm

⇒ RE the XOR-ed canary

⇒ spoof custom canary for shellcode



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                                     ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+55
                        0Dh
                        sub_31411B

loc_31306D:                                     ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                                     ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                                     ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Random XOR Canaries

- Benefits
  - same protection as basic random canaries
  - defends against specific attacks involving control data or retur

    value changes without overflowing the canary (invalidates it)
    - XOR's the canary with the return address
  - protect against overflowing buffers in structures
    - attacker tries to make pointer point at control data

Random XOR Canaries

- Downsides
  - + more overhead means more security
    - # of canaries (StackGuard uses 128 static) & complexity of algorithm
  - + only protects control data from being altered

    IF the attacker is overwriting pointers
  - + still allows overwrite of data and the pointers themselves
    - function pointers can be victimized

      ⇒ overflow into them and call to execute shellcode

Random (XOR) Canaries

Moral of the Story:

everything relies on good crypto

For Both Random and Random XOR, the main security element relies on good random number generation. Pseudorandom sequences can be learned. Cryptographic PRNG can be found.

Lecture Overview

1. How do we protect against overflows?
2. Different Types
3. Guarding the Stack
4. Ways to Leak Information
5. When All Else Fails

Guarding the Stack

StackGuard - Used in patches of GCC

- started in 1998 as static canaries
- original prototype written in a few days by an intern
  - promptly patched into GCC
  - first canary was hardcoded

Guarding the Stack

StackGuard - The first canary

0xDEADBEEF

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Guarding the Stack

StackGuard

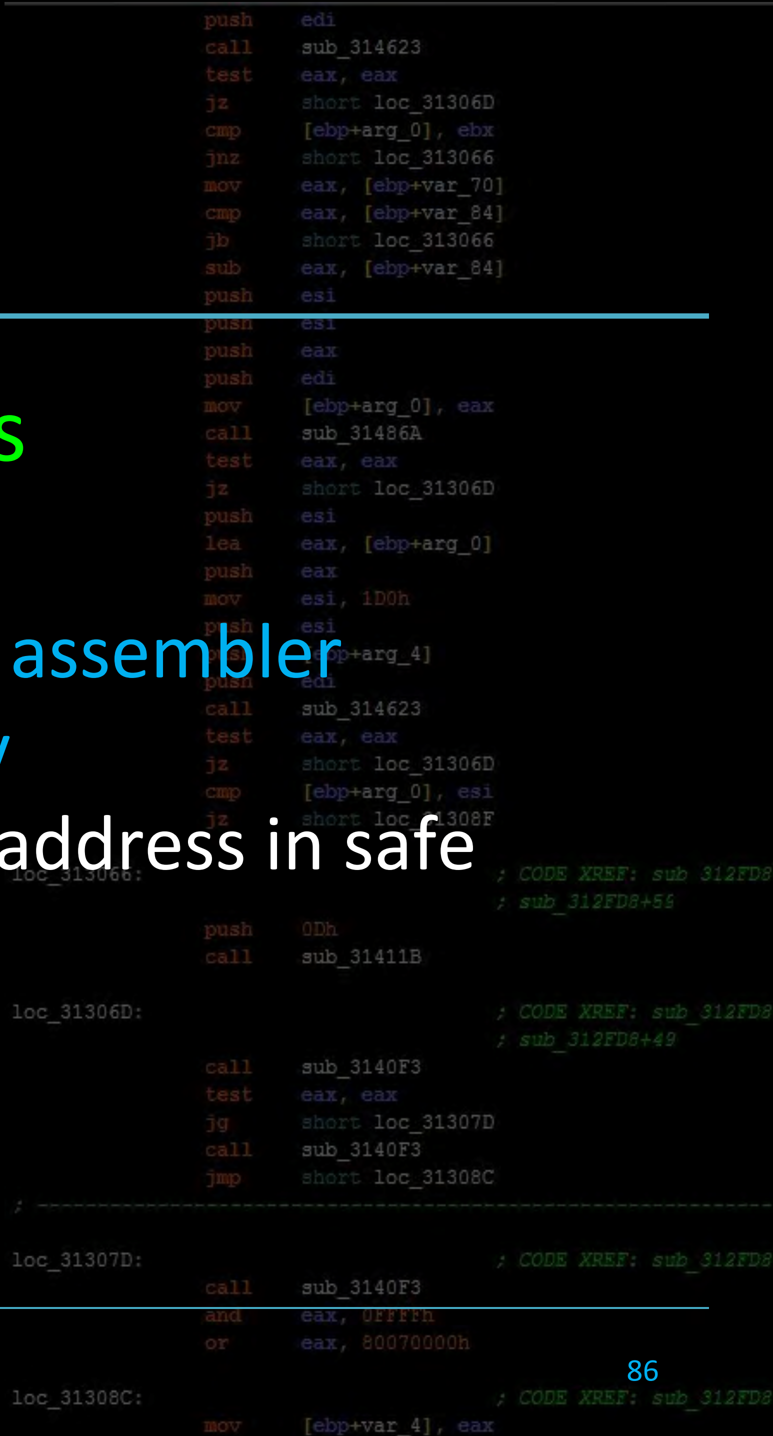
- terminator canary
  - CR, LF, 00, -1
- single random canary
  - using /dev/random
- single XOR random canary
  - xor-ed return address

```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Guarding the Stack

StackGuard - Extra Benefits

- implemented as modified assembler
- single XOR random canary
  - stores the valid return address in safe memory



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F
loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Guarding the Stack

Modding StackGuard - PointGuard

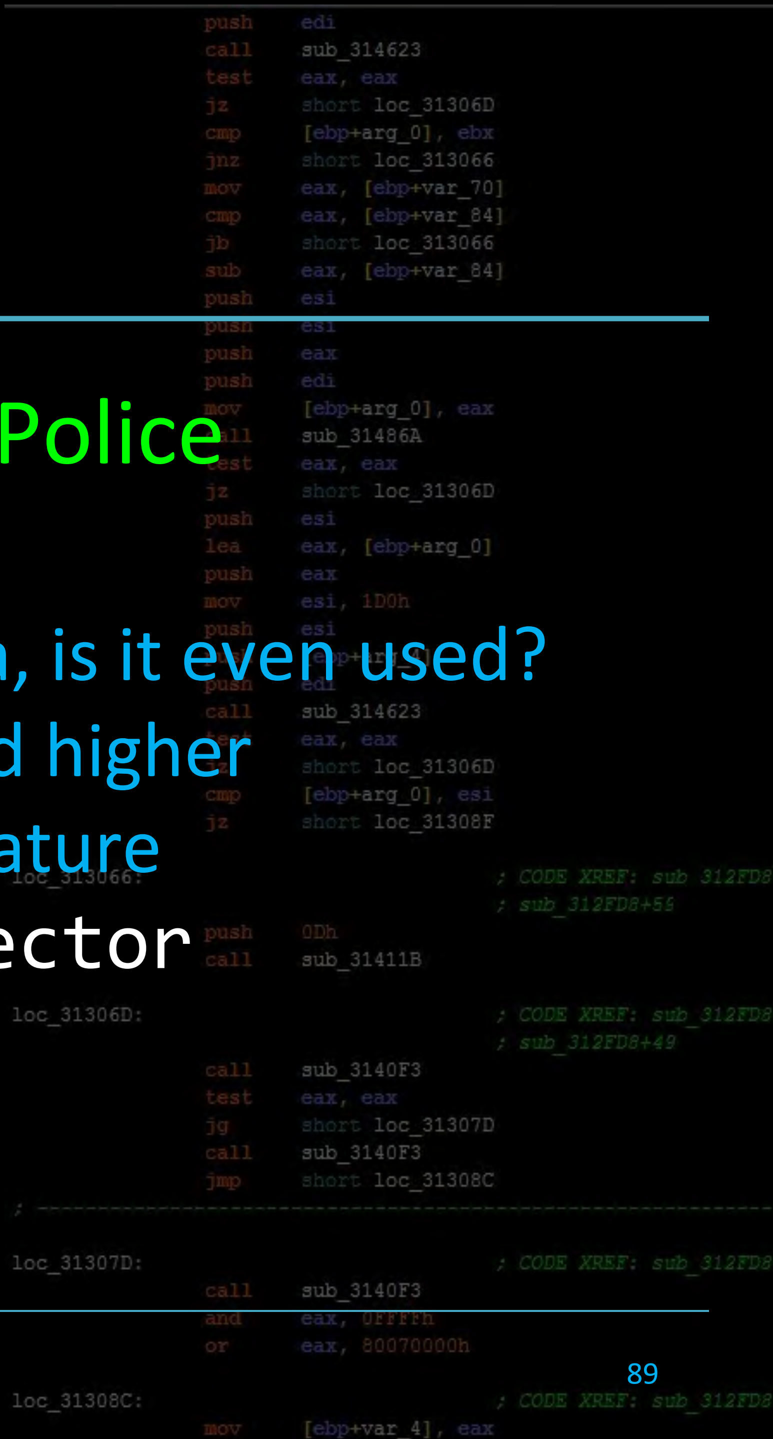
- does everything StackGuard does but is newer and slower
- allows canaries to be added to different data items,
  - automatically: FP and longjump buffers
  - requires users to specify which data items they think will be exploited

Guarding the Stack

Modding StackGuard - ProPolice

- also at the compiler level (a patch to GCC)
- does everything StackGuard does
- enhancements:
  - variable sorting

    $\Rightarrow$ buffers sorted to top of local variables, means they can’t overflow important values



Guarding the Stack

Modding StackGuard - ProPolice

...so this sounds like a good idea, is it even used?

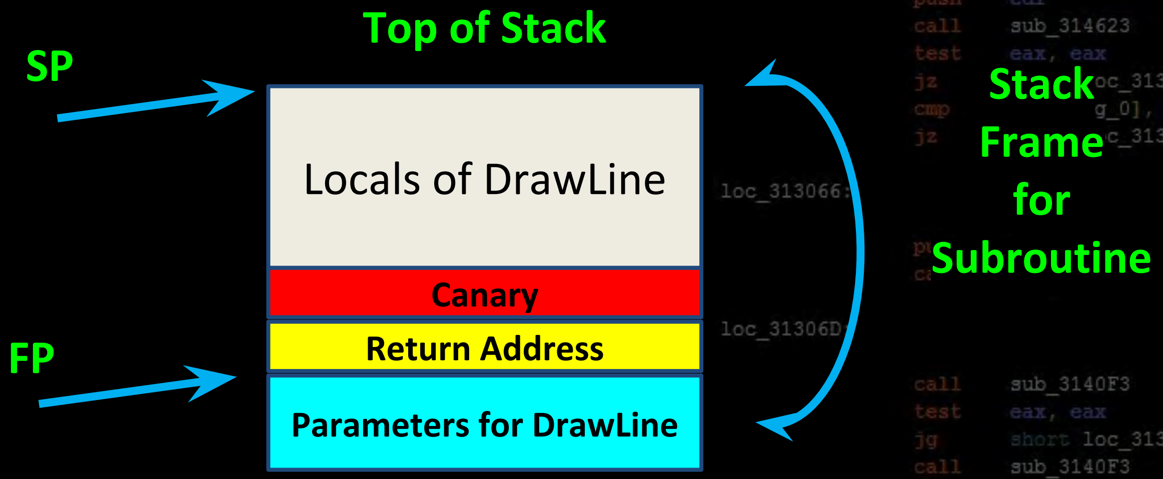
- Visual Studios 2003 and higher
- GCC uses it with the feature

  `-fstack_protector`

Guarding the Stack

Modding StackGuard - ProPolice

{ GCC 3.4.1 `–fstack_protector` }

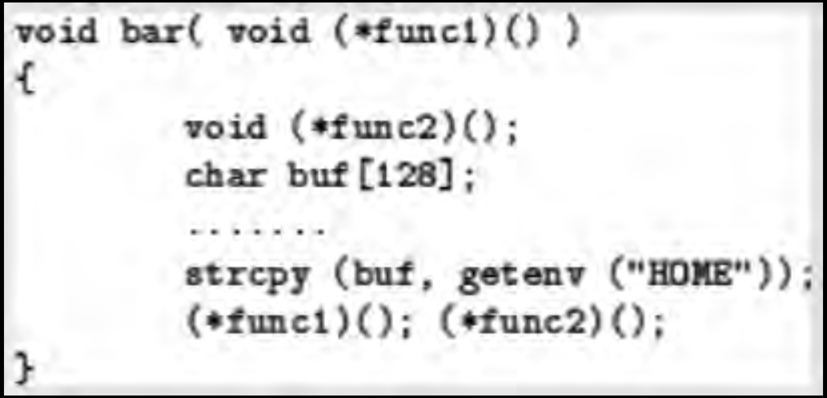


Guarding the Stack

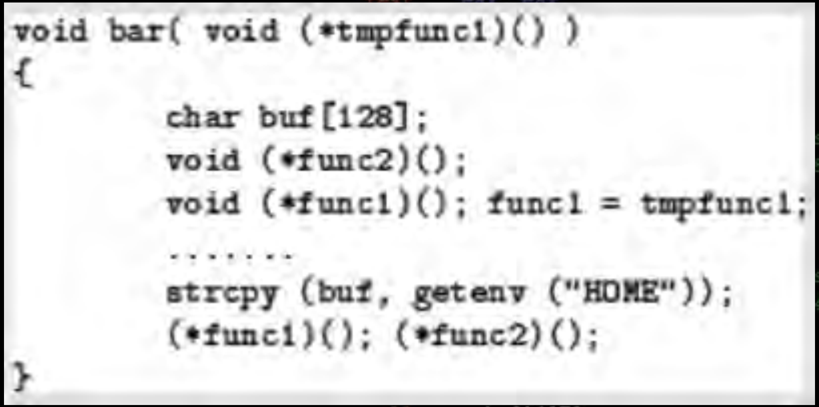
Modding StackGuard - ProPolice

{ From ProPolice Documentation }

Before → After



```
void bar( void (*func1)() )
{
        void (*func2)();
        char buf[128];
        .......
        strcpy (buf, getenv ("HOME"));
        (*func1)(); (*func2)();
}
```

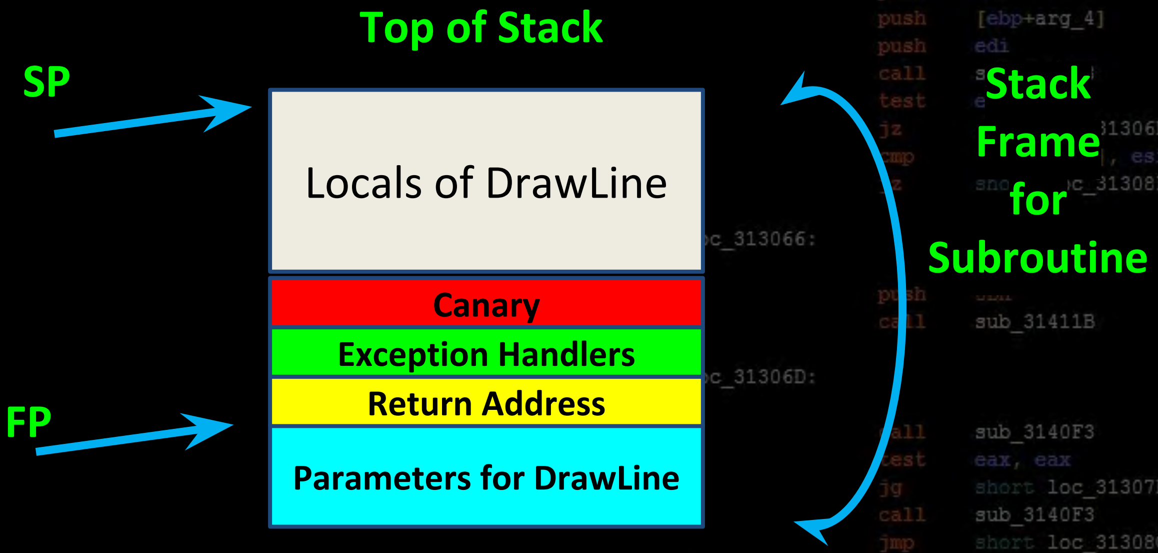


```
void bar( void (*tmpfunc1)() )
{
        char buf[128];
        void (*func2)();
        void (*func1)(); func1 = tmpfunc1;
        .......
        strcpy (buf, getenv ("HOME"));
        (*func1)(); (*func2)();
}
```

Guarding the Stack

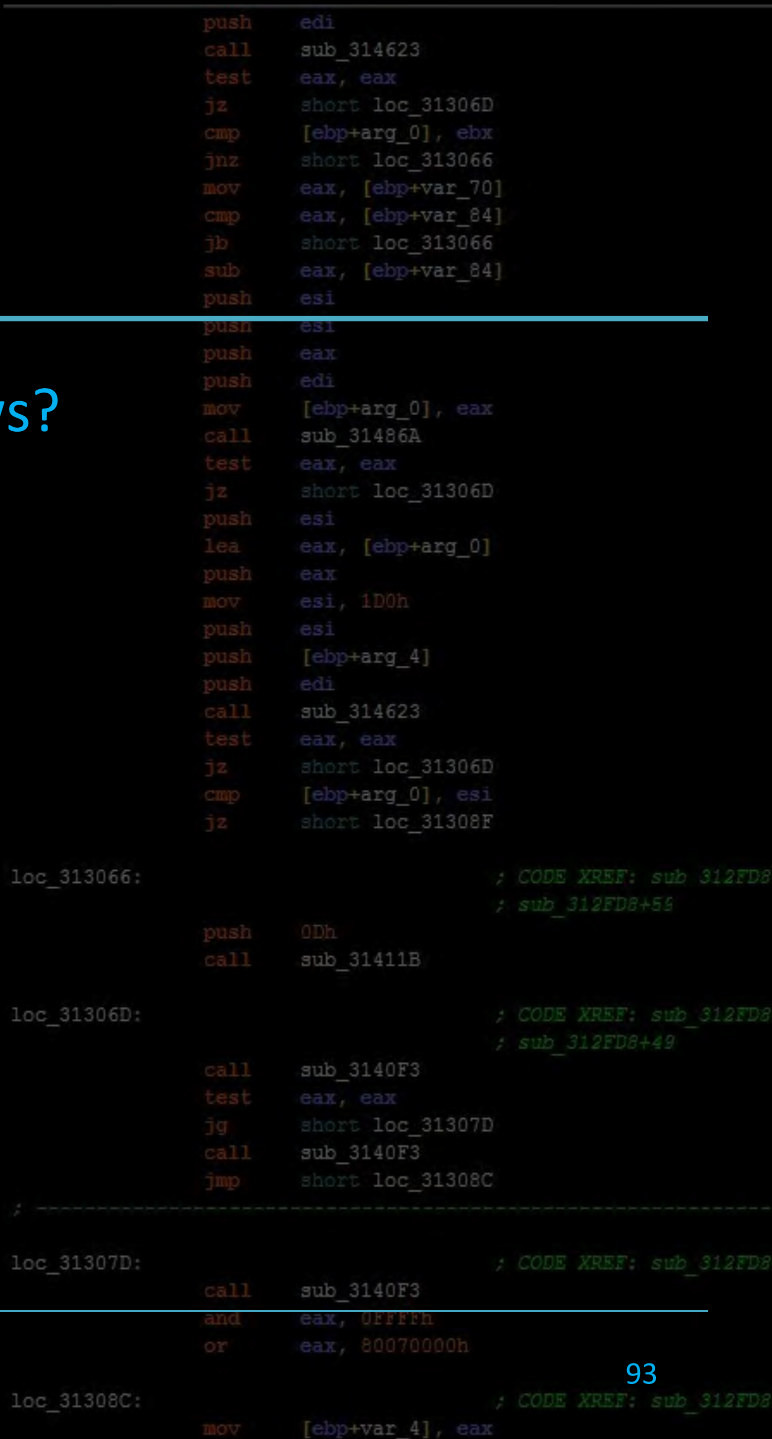
Modding StackGuard - ProPolice

{ MS Visual Studio 2003+ /GS }



Lecture Overview

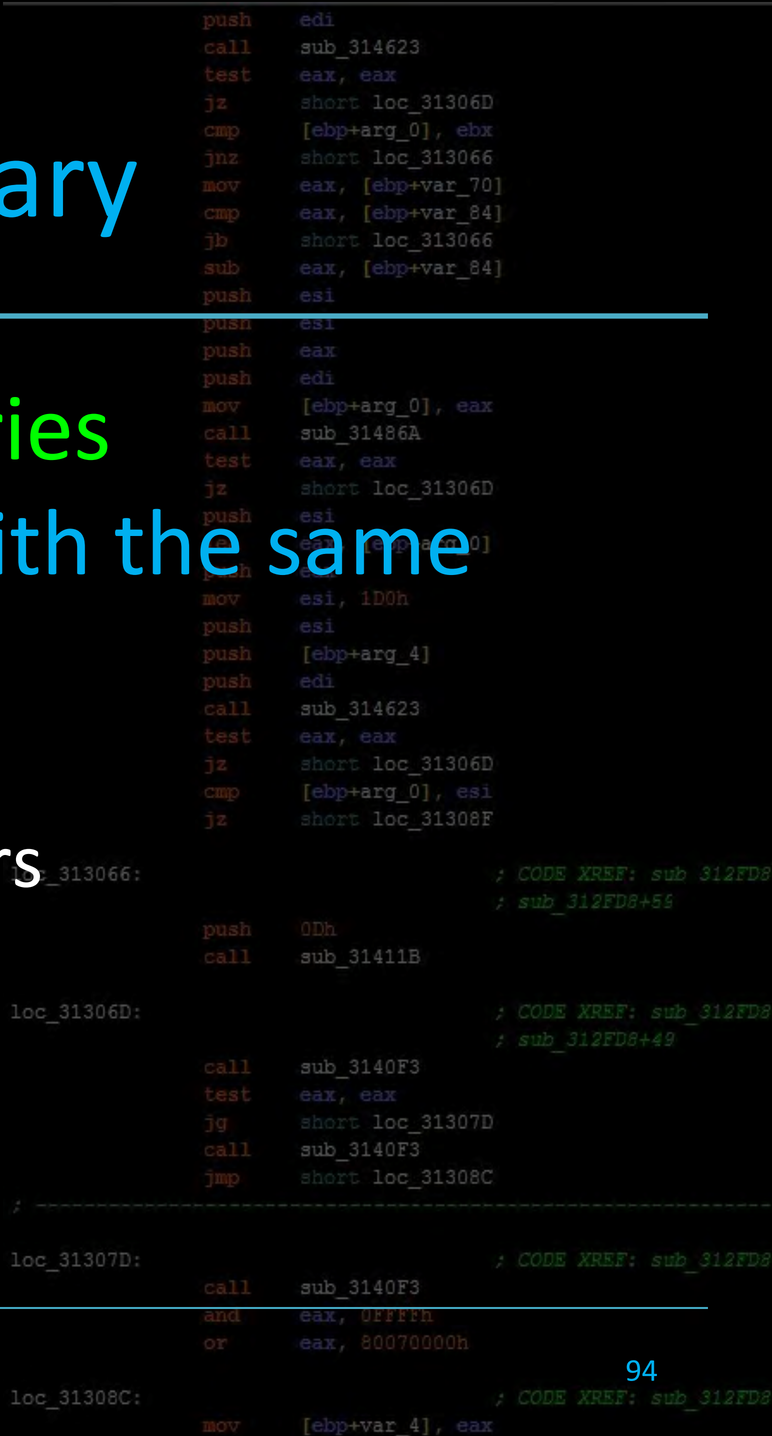
1. How do we protect against overflows?
2. Different Types
3. Guarding the Stack
4. Ways to Leak Information
5. When All Else Fails



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                             ; CODE XREF: sub_312FD8
                                        ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                             ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                             ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Ways to Leak the Canary

- Focus on Random Canaries
- Overwrite the Canary with the same value
  - brute force
  - learnable random numbers
  - unprotected data type
  - reading off of the stack



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                [illegible] eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                              ; CODE XREF: sub_312FD8
                                         ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                              ; CODE XREF: sub_312FD8
                                         ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                              ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                              ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Ways to Leak the Canary

- Brute Force
  - cool example attack: http://vagmour.eu/persistence-1/
  - requires same canary for each thread so can’t call execve()
  - overwrite canary byte by byte

Ways to Leak the Canary

- Learnable Random Numbers
  - GS calculate the canary 2007 http://uninformed.org/?v=7&a=2&t=sumry
  - Android PRNG example 2014 (IBM): https://www.usenix.org/system/files/conference/woot14/woot14-kaplan.pdf
  - bad crypto for random generator
  - if /dev/random is not found, sometimes pseudo-random generators are used

Ways to Leak the Canary

- Unprotected Data Item
  - usually if it isn't a string buffer, there will not be a canary

Ways to Leak the Canary

- Reading Off of the Stack
  - buffer overflow
    - overwrite null terminator

      ⇒ read past the end of array
    - format string vulnerabilities

      http://www.exploit-monday.com/2011/06/leveraging-format-string.html

Ways to Leak the Canary

- Reading Off of the Stack
  - information leaks /memory leaks (out of scope)
    - more complicated attack
    - useful against the stack reordering done by StackGuard/ ProPolice
    - pointers dangling /writing or reading after free
    - http://phrack.org/issues/56/5.html

Terminator Canaries

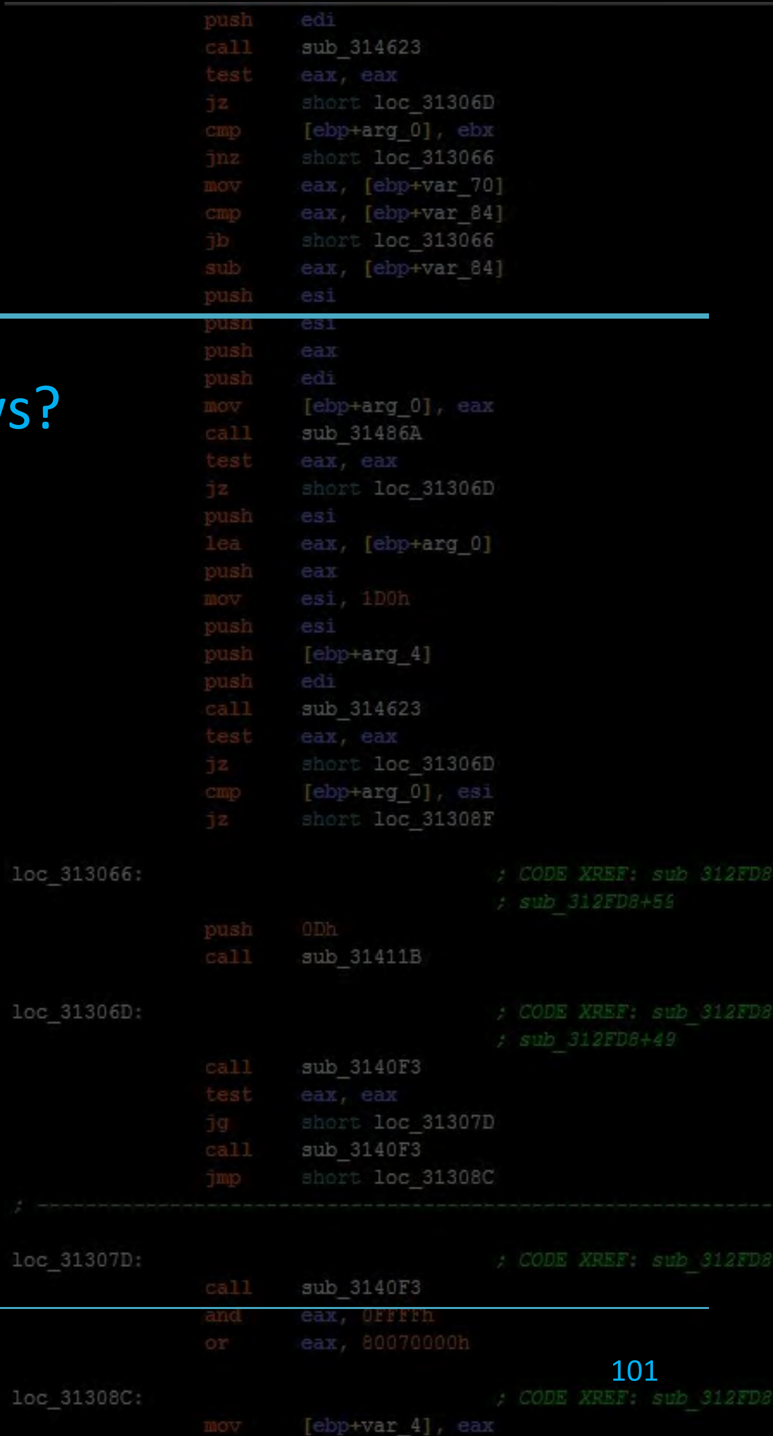
Exercise

Terminator Canary Bypass

ssh lecture@warzone.rpis.ec

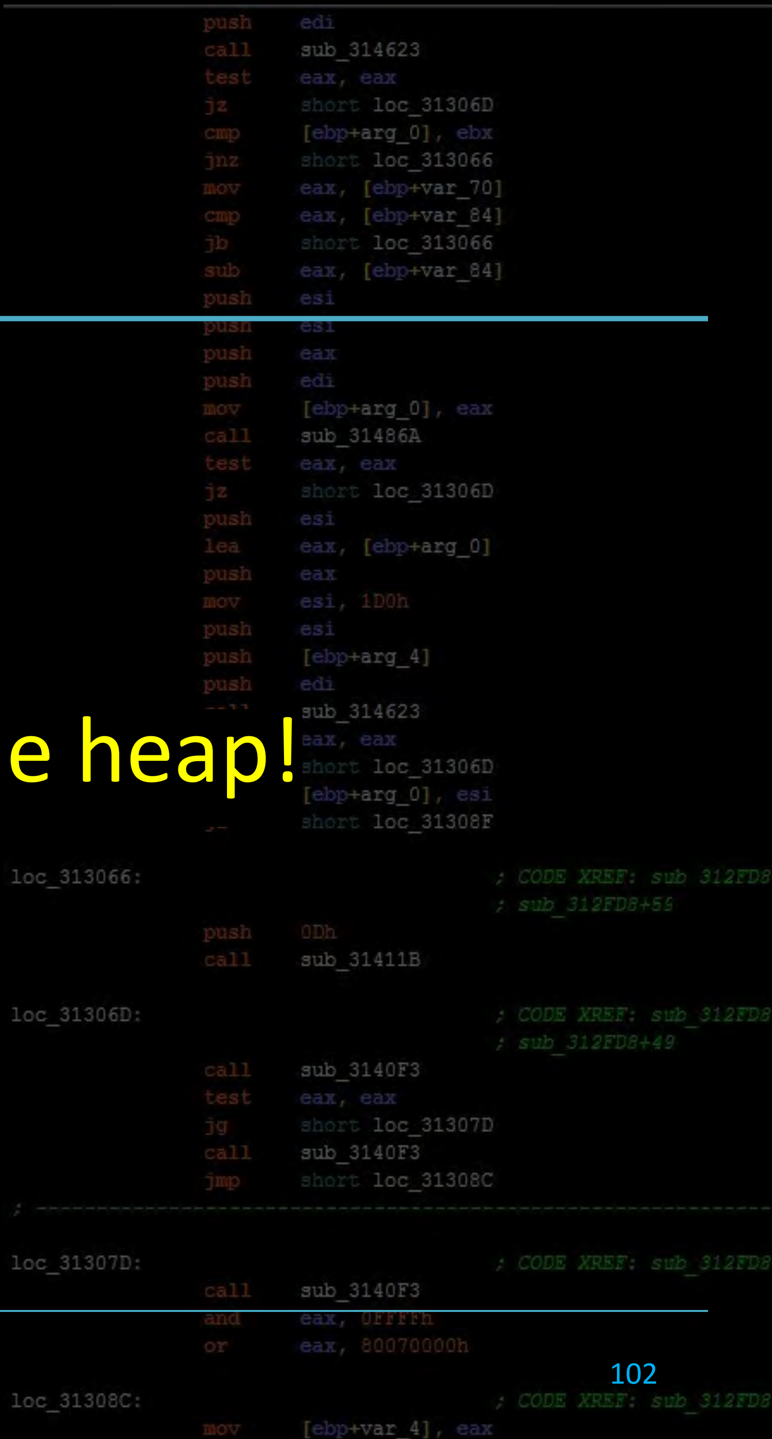
Lecture Overview

1. How do we protect against overflows?
2. Different Types
3. Guarding the Stack
4. Ways to Leak Information
5. When All Else Fails



```
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], ebx
                jnz     short loc_313066
                mov     eax, [ebp+var_70]
                cmp     eax, [ebp+var_84]
                jb      short loc_313066
                sub     eax, [ebp+var_84]
                push    esi
                push    esi
                push    eax
                push    edi
                mov     [ebp+arg_0], eax
                call    sub_31486A
                test    eax, eax
                jz      short loc_31306D
                push    esi
                lea     eax, [ebp+arg_0]
                push    eax
                mov     esi, 1D0h
                push    esi
                push    [ebp+arg_4]
                push    edi
                call    sub_314623
                test    eax, eax
                jz      short loc_31306D
                cmp     [ebp+arg_0], esi
                jz      short loc_31308F

loc_313066:                                     ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
                push    0Dh
                call    sub_31411B

loc_31306D:                                     ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
                call    sub_3140F3
                test    eax, eax
                jg      short loc_31307D
                call    sub_3140F3
                jmp     short loc_31308C
; ---------------------------------------------------------------------------

loc_31307D:                                     ; CODE XREF: sub_312FD8
                call    sub_3140F3
                and     eax, 0FFFFh
                or      eax, 80070000h

loc_31308C:                                     ; CODE XREF: sub_312FD8
                mov     [ebp+var_4], eax
```

Remember...



```
        push    edi
        call    sub_314623
        test    eax, eax
        jz      short loc_31306D
        cmp     [ebp+arg_0], ebx
        jnz     short loc_313066
        mov     eax, [ebp+var_70]
        cmp     eax, [ebp+var_84]
        jb      short loc_313066
        sub     eax, [ebp+var_84]
        push    esi
        push    esi
        push    eax
        push    edi
        mov     [ebp+arg_0], eax
        call    sub_31486A
        test    eax, eax
        jz      short loc_31306D
        push    esi
        lea     eax, [ebp+arg_0]
        push    eax
        mov     esi, 1D0h
        push    esi
        push    [ebp+arg_4]
        push    edi
                sub_314623
                eax, eax
                short loc_31306D
                [ebp+arg_0], esi
                short loc_31308F

loc_313066:                                     ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+59
        push    0Dh
        call    sub_31411B

loc_31306D:                                     ; CODE XREF: sub_312FD8
                                                ; sub_312FD8+49
        call    sub_3140F3
        test    eax, eax
        jg      short loc_31307D
        call    sub_3140F3
        jmp     short loc_31308C
; ---------------------------------------------------------------

loc_31307D:                                     ; CODE XREF: sub_312FD8
        call    sub_3140F3
        and     eax, 0FFFFh
        or      eax, 80070000h

loc_31308C:                                     ; CODE XREF: sub_312FD8
        mov     [ebp+var_4], eax
```

No canaries on the heap!