Sécurité des logiciels

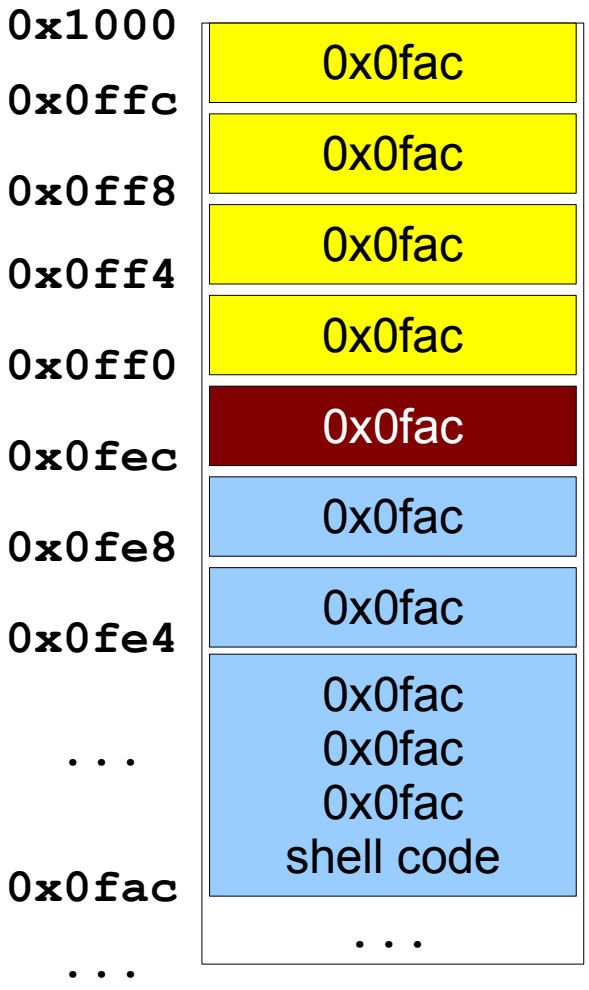
NX? Not on the stack? Still pwnd!

Samuel Thibault <samuel.thibault@u-bordeaux.fr>

Stack overflow exploit needs...

Stack overflow exploit needs

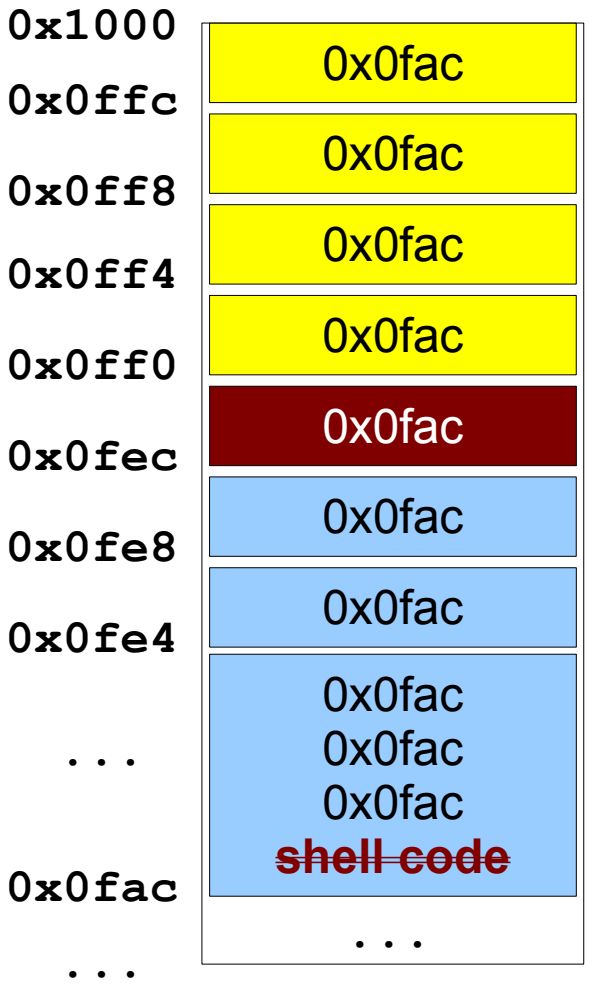
- Executable stack
- Buffer on the stack
- Known position



Stack overflow exploit needs...

Stack overflow exploit needs

- **Executable stack**
- Buffer on the stack
- Known position

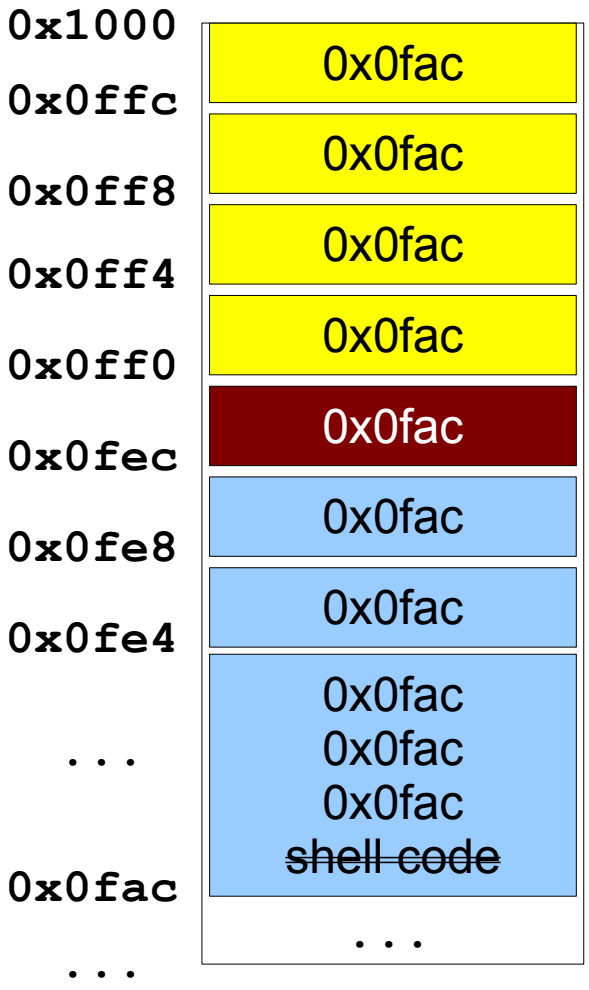


Not executable? Still pwnd!

How to execute stuff
without being executable?

Remember what our shell code was:
Basically
execve(“/bin/sh”, {“/bin/sh”, NULL}, {NULL});

Do we really need to write code for this?



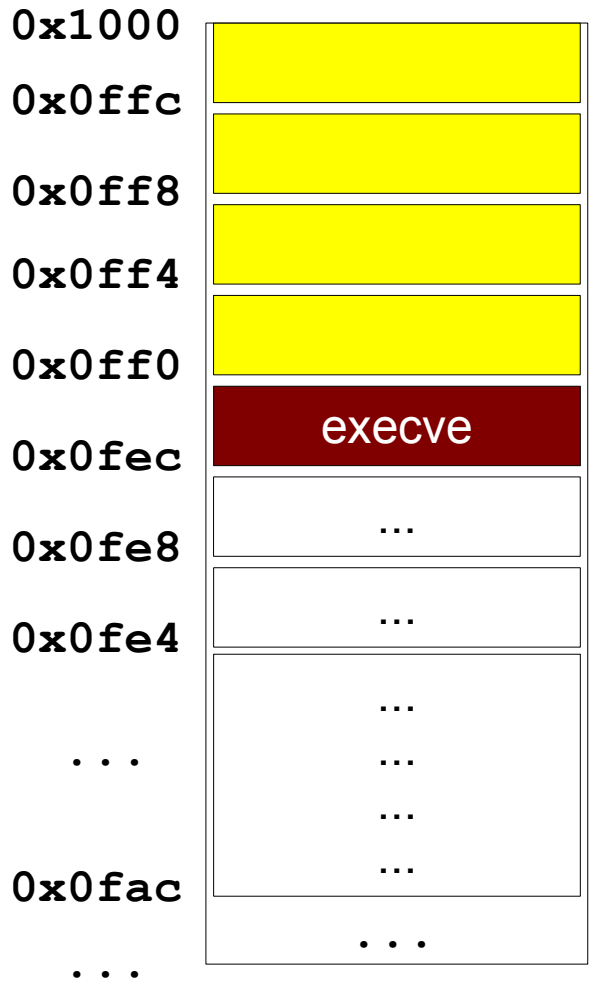
Not executable? Still pwnd!

How to execute stuff
without being executable?

Remember what our shell code was:
Basically
execve(“/bin/sh”, {“/bin/sh”, NULL}, {NULL});

Do we really need to write code for this?

What if we just ret to execve?



Not executable? Still pwnd!

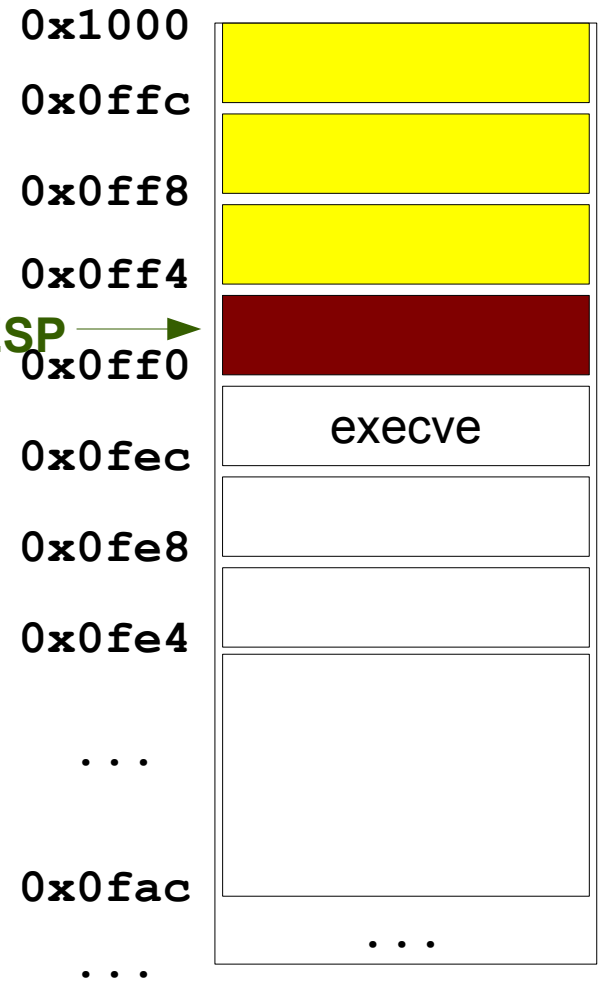
How to execute stuff
without being executable?

Remember what our shell code was:
Basically
execve(“/bin/sh”, {“/bin/sh”, NULL}, {NULL});

Do we really need to write code for this?

What if we just ret to execve?

Result of ret to execve



Not executable? Still pwnd!

How to execute stuff
without being executable?

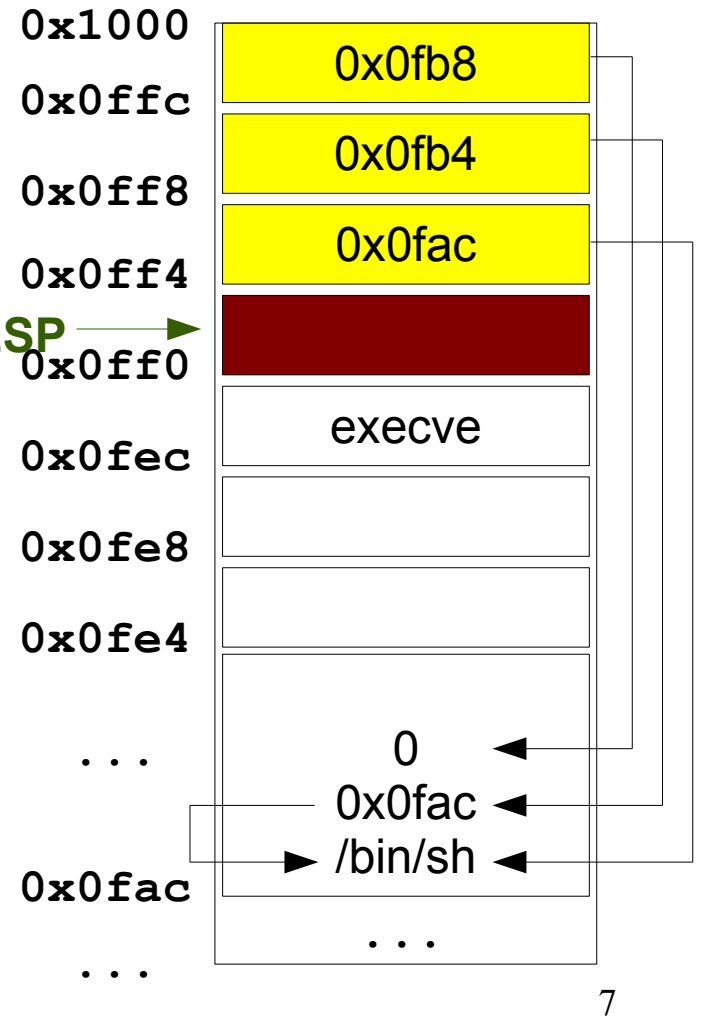
Remember what our shell code was:
Basically
execve(“/bin/sh”, {“/bin/sh”, NULL}, {NULL});

Do we really need to write code for this?

What if we just ret to execve?

Result of ret to execve

So we have to set parameters accordingly



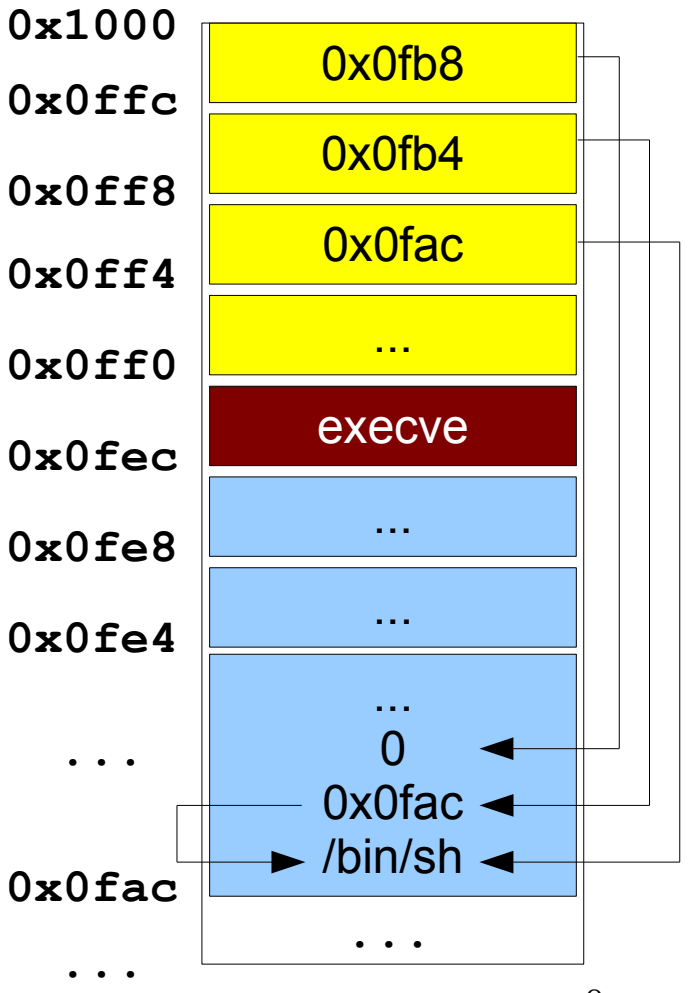
Not executable? Still pwnd!

How to execute stuff
without being executable?

Overflow **exactly** like this!

Ret-into-libc hack

What if we want to do more than just one system call?



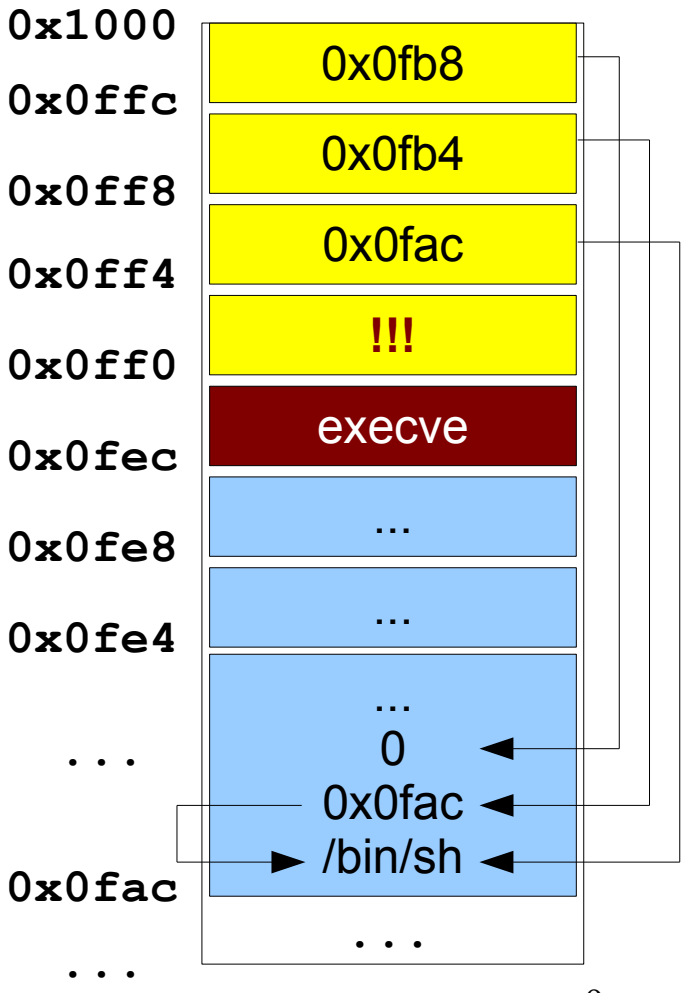
Not executable? Still pwnd!

How to execute stuff
without being executable?

Overflow **exactly** like this!

Ret-into-libc hack

What if we want to do more than just one system call?

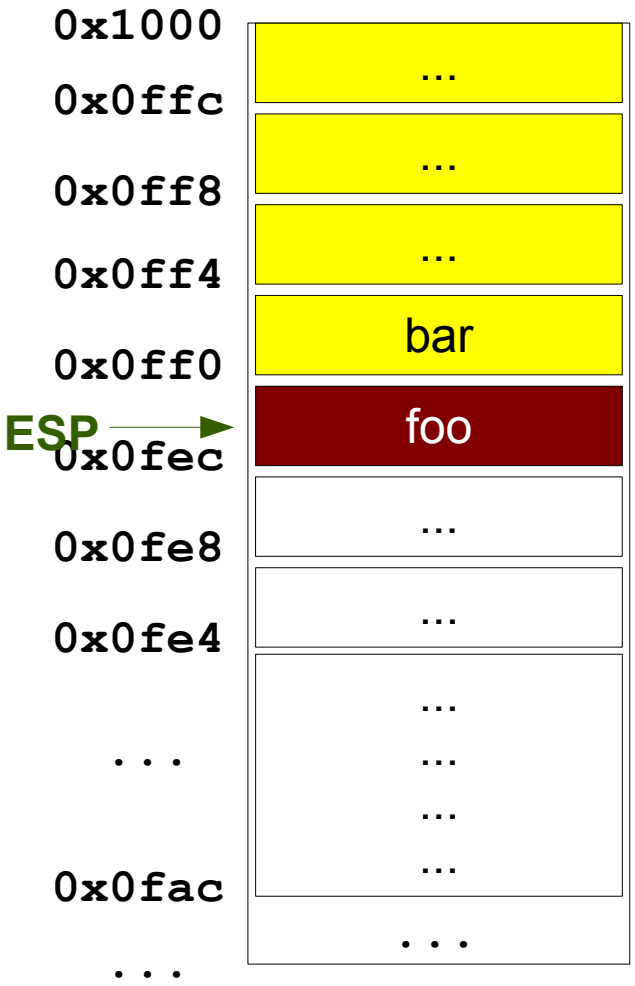


Not executable? Still pwnd!

How to execute stuff
without being executable?

A second chance...

First “return to” foo

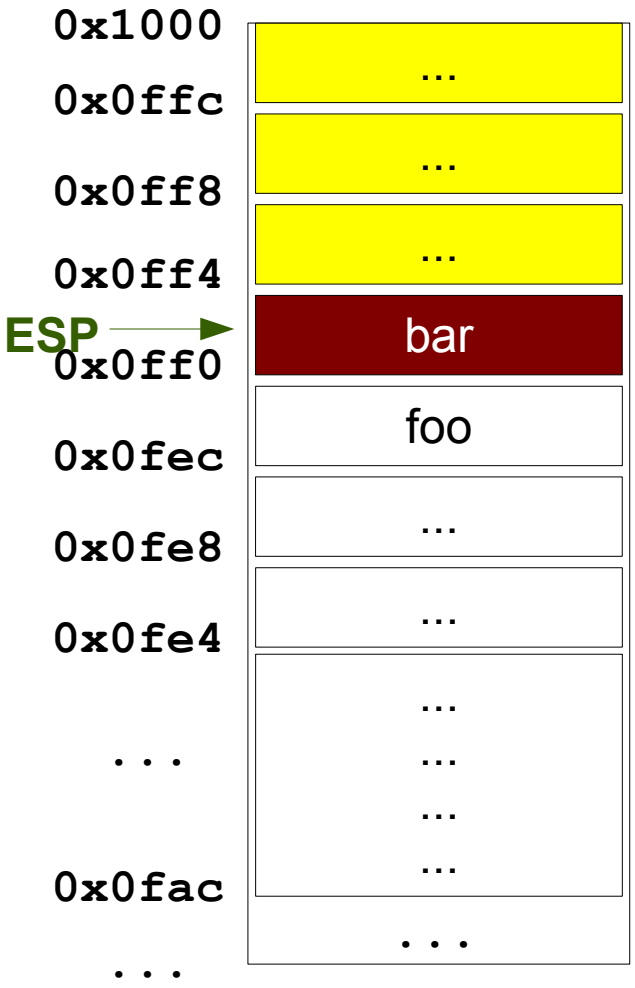


Not executable? Still pwnd!

How to execute stuff
without being executable?

A second chance...

First “return to” foo
Now in foo, then “return to” bar



Not executable? Still pwnd!

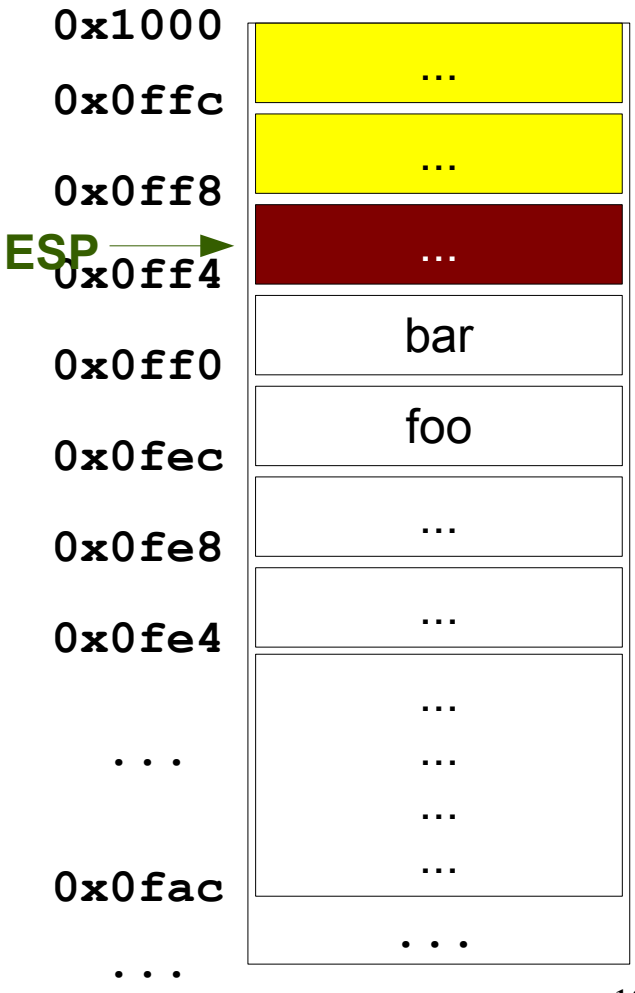
How to execute stuff
without being executable?

A second chance...

First “return to” foo
Now in foo, then “return to” bar
Now in bar

Err, but then bar’s parameters need to be almost the same as foo’s...

But bar does not have to be a **proper function!**



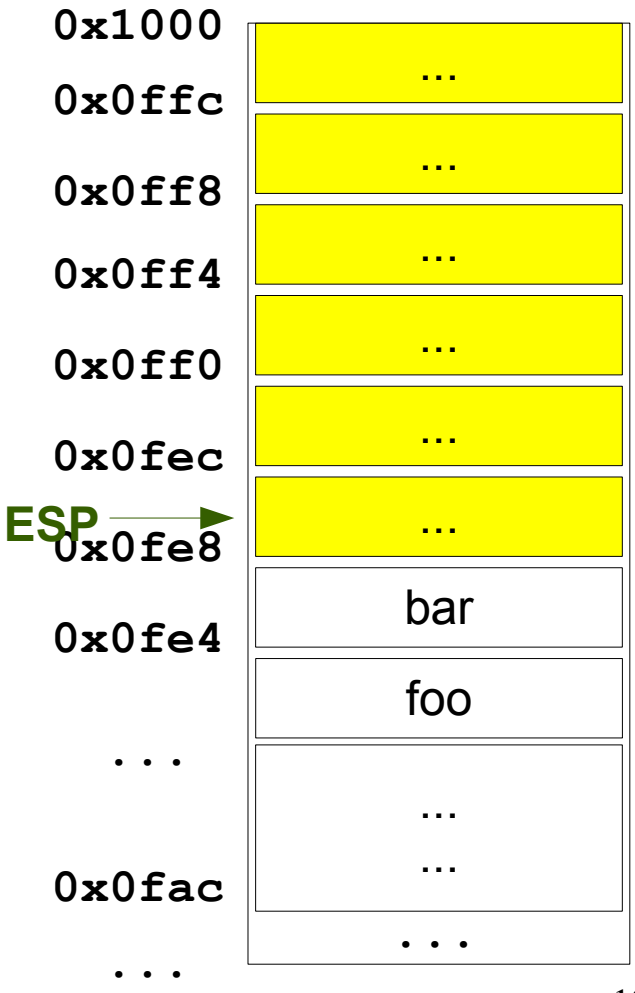
Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack

A lot of glibc functions end with cleaning the stack, e.g.:

```
bar:
addl $12,%esp
ret
```



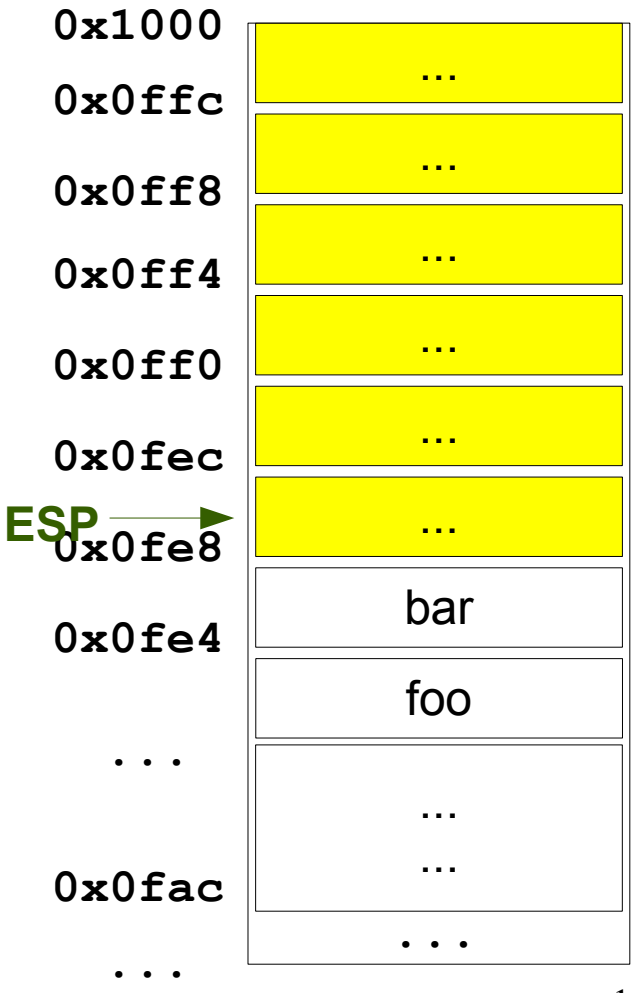
Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack

A lot of glibc functions end with cleaning the stack, e.g.:

```
bar:
addl $12,%esp
ret
```



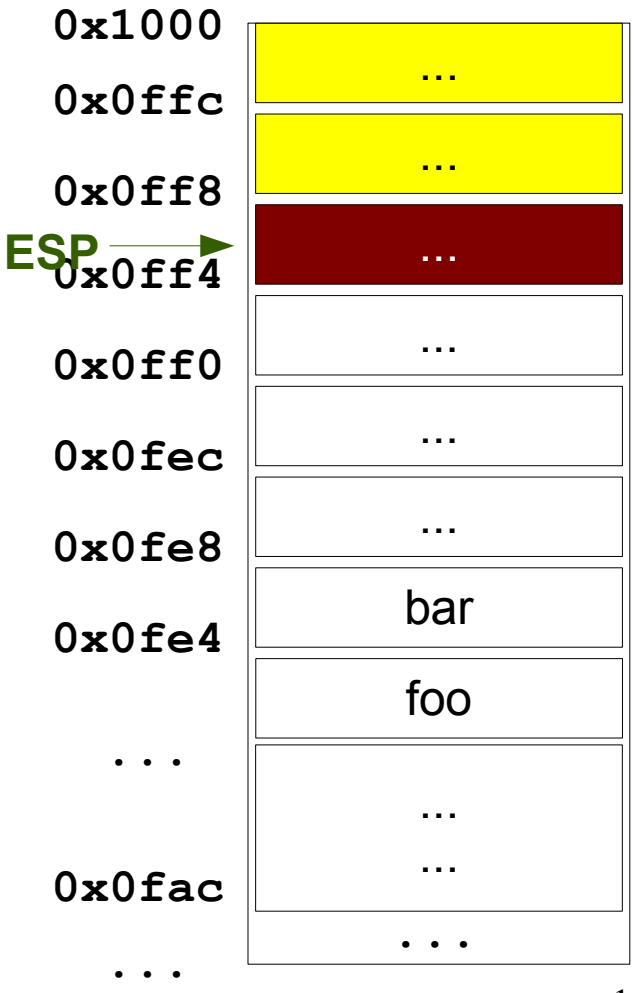
Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack

A lot of glibc functions end with cleaning the stack, e.g.:

```
bar:
addl $12,%esp
ret
```



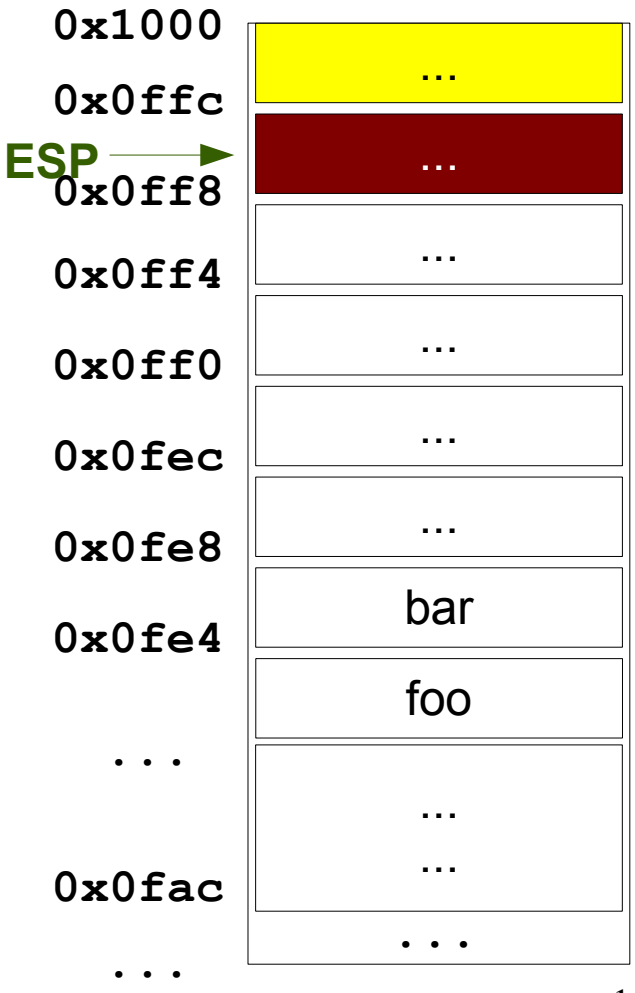
Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack

A lot of glibc functions end with cleaning the stack, e.g.:

```
bar:
addl $12,%esp
ret
```

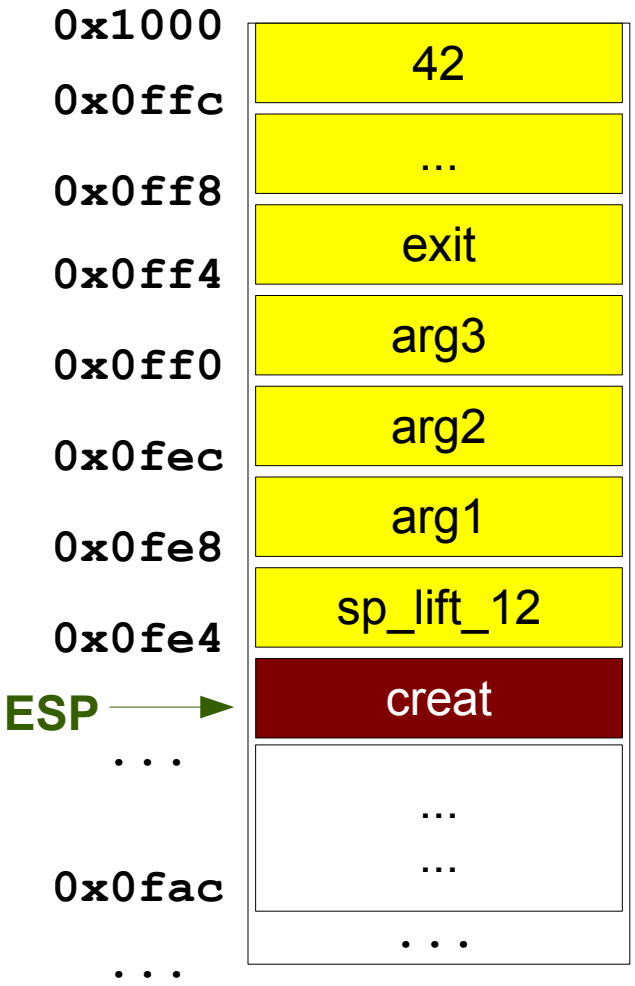


Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack, complete story:

First ret-into-libc

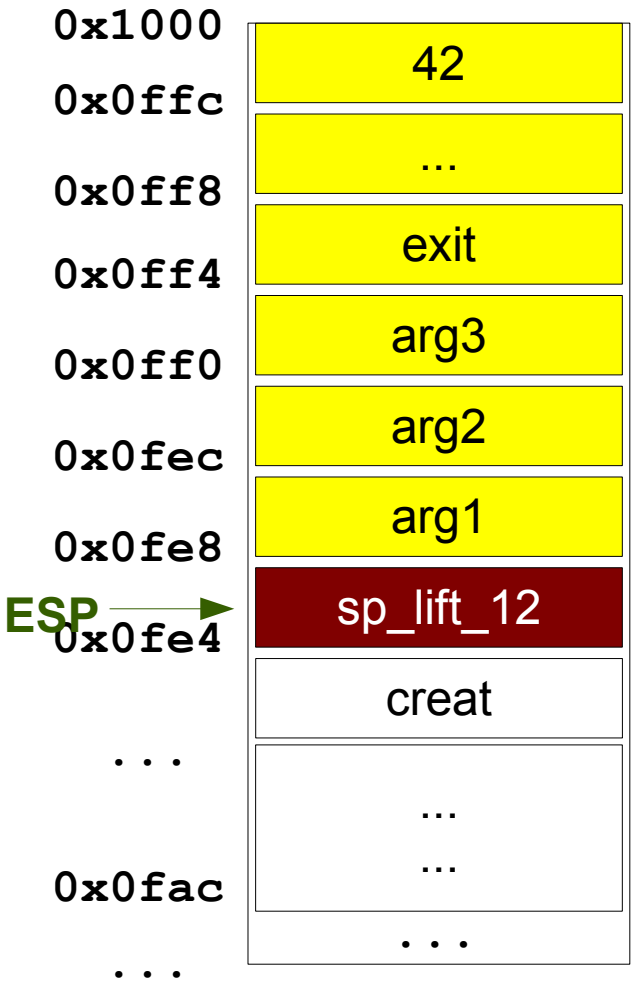


Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack, complete story:

First ret-into-libc
Now in creat, ..., returns to sp_lift_12

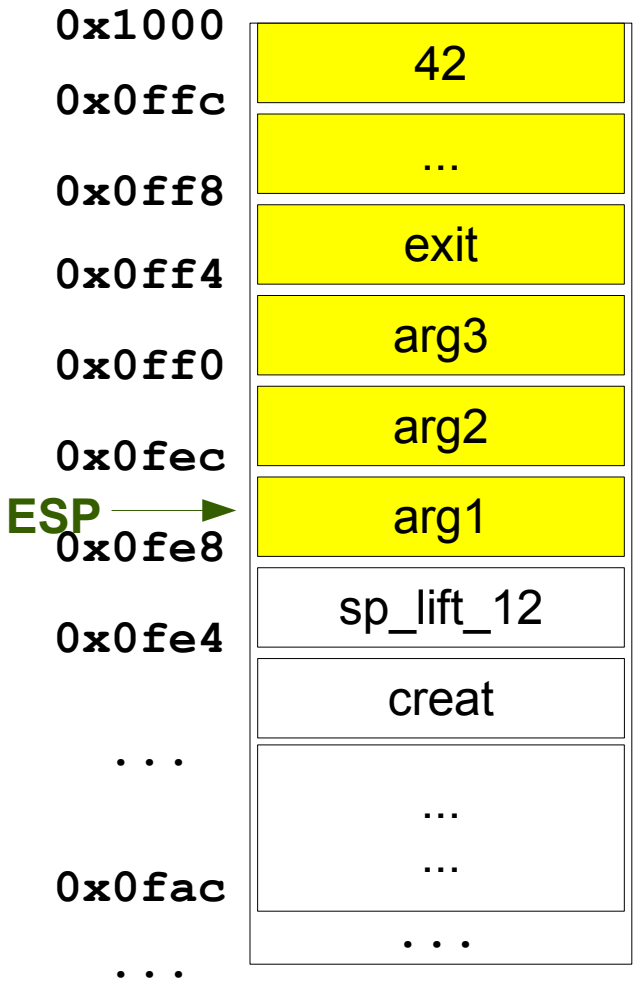


Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack, complete story:

First ret-into-libc
Now in creat, ..., returns to sp_lift_12
Now in sp_lift, cleans stack

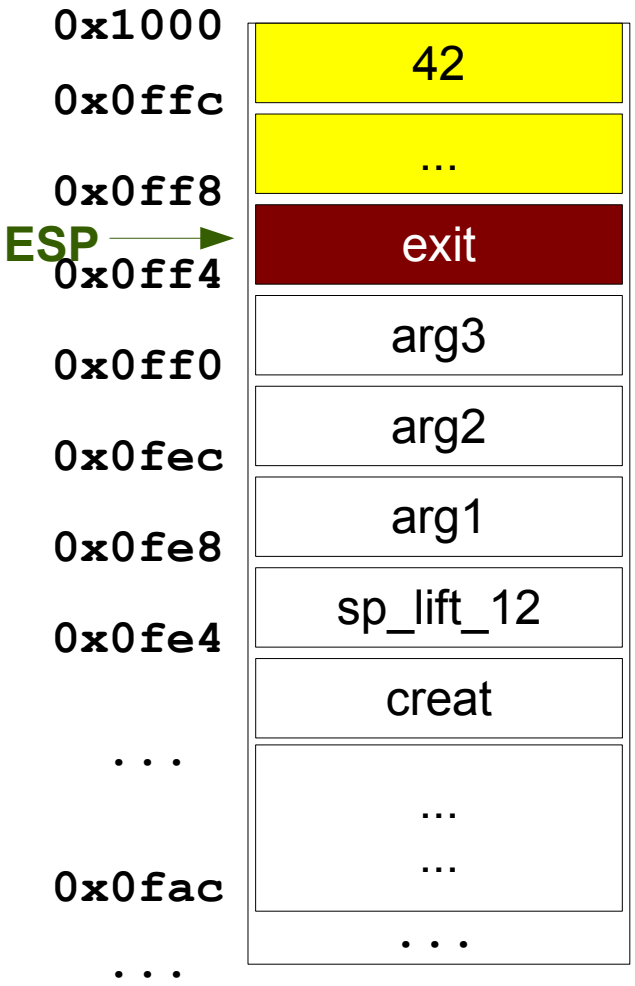


Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack, complete story:

First ret-into-libc
Now in creat, ..., returns to sp_lift_12
Now in sp_lift, cleans stack
Stack cleaned, returns to exit

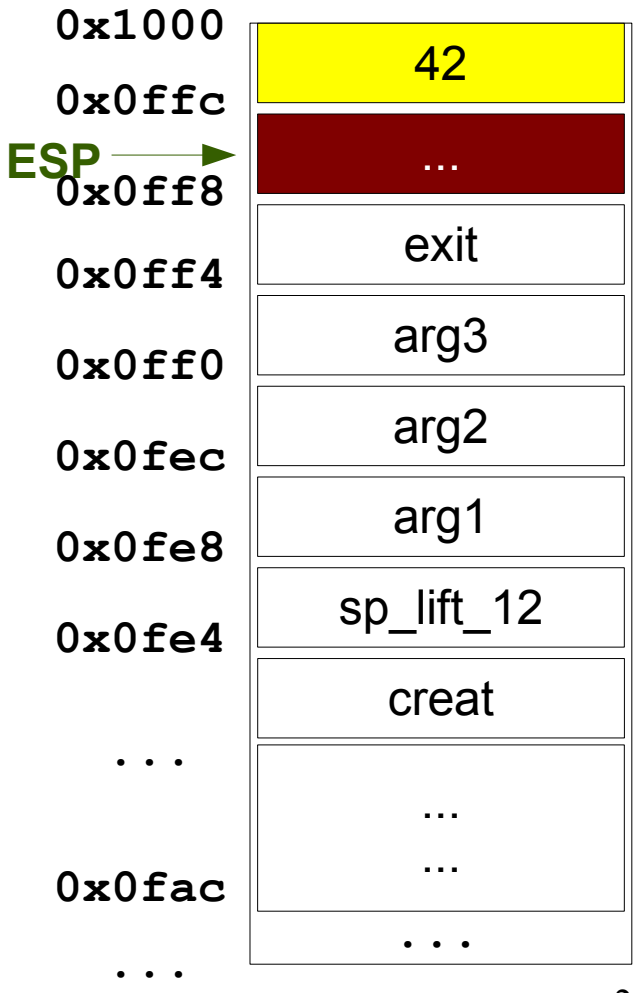


Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack, complete story:

First ret-into-libc
Now in creat, ..., returns to sp_lift_12
Now in sp_lift, cleans stack
Stack cleaned, returns to exit
Now in exit, exits nicely
pwnd!

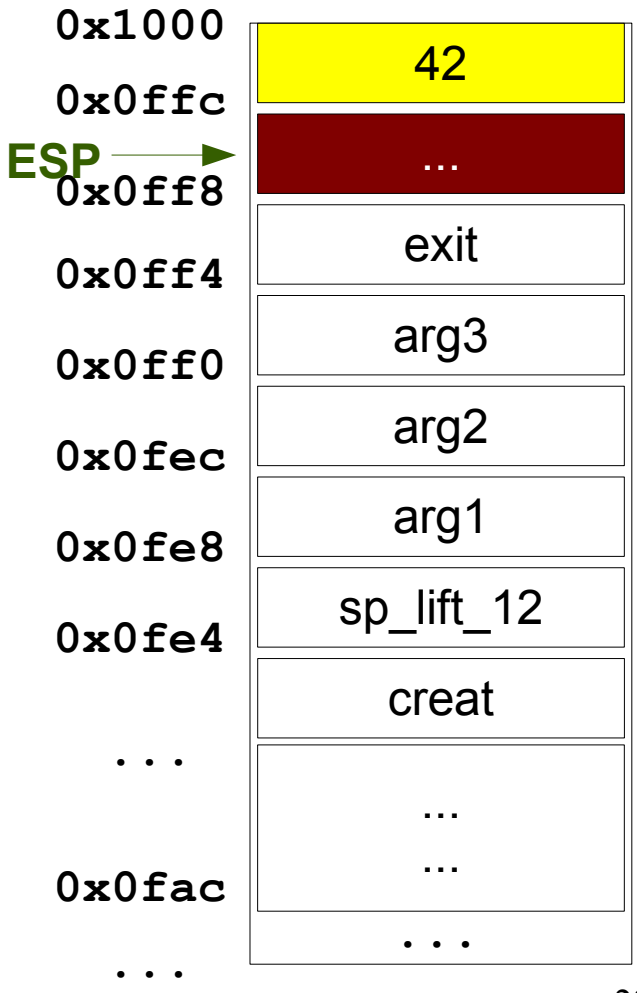


Not executable? Still pwnd!

How to execute stuff
without being executable?

SP lifting hack

- Call chain can be arbitrarily long
- Just need to find in libc what you need

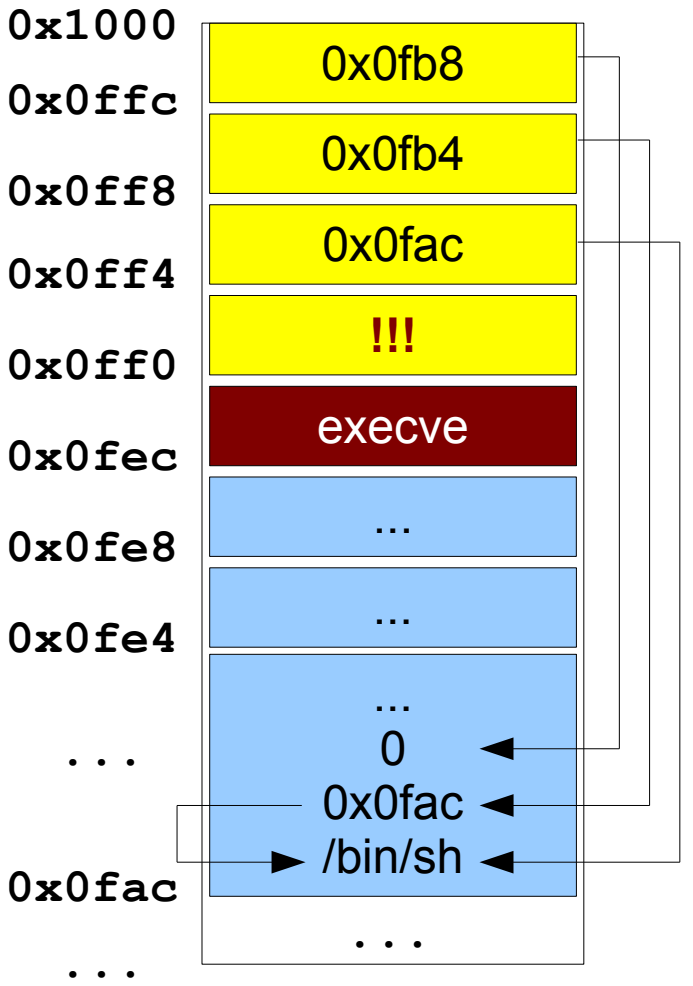


Not executable? Still pwnd!

What is the basic problem?

We can overflow

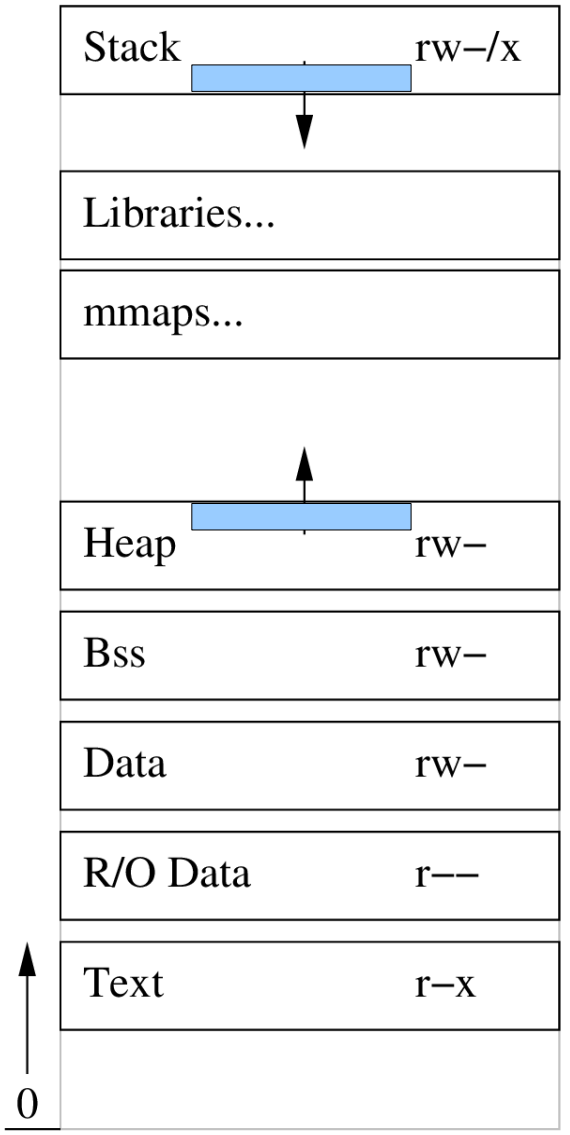
- from a **data** buffer
- into a **control** area



Stack overflow exploit needs...

Stack overflow exploit needs

- Executable stack
- Buffer on the ~~stack~~ **heap**
- Known position



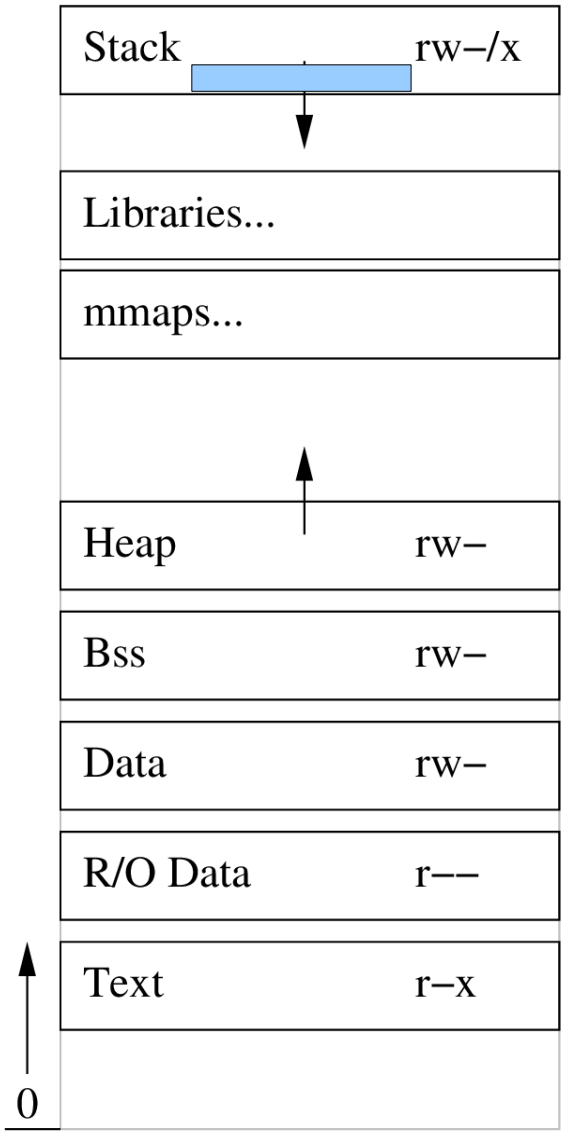
Heap overflow?

```
int litentier(void) {
  int i;
  char buf[64];

  printf("> ");
  fflush(stdout);

  gets(buf);
  i=atoi(buf);

  return i;
}
```



Heap overflow?

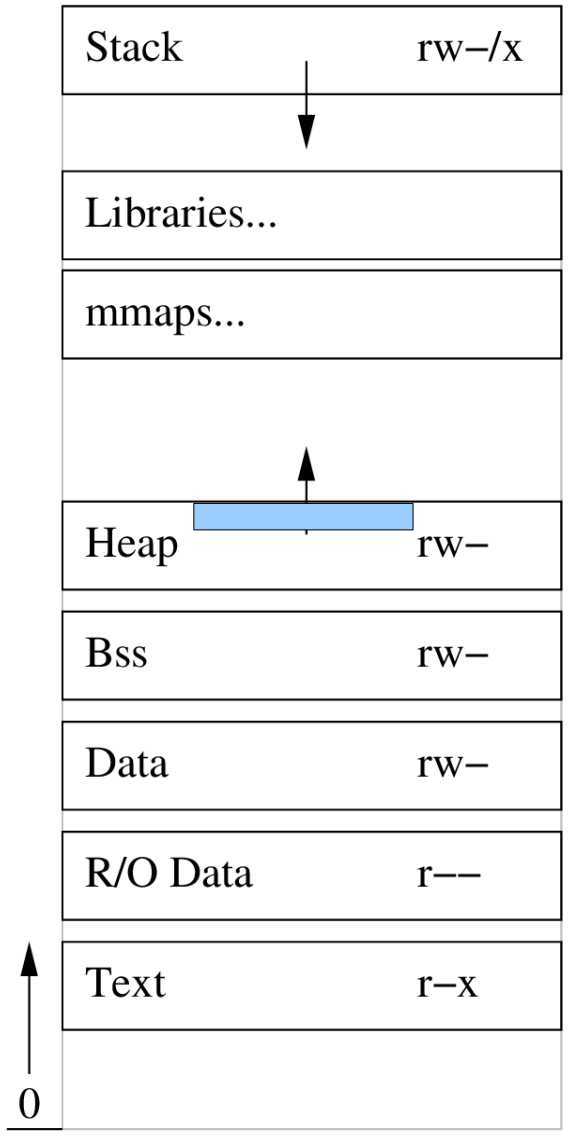
```
int litentier(void) {
   int i;
   char *buf = malloc(64);

   printf("> ");
   fflush(stdout);

   gets(buf);
   i=atoi(buf);
   free(buf);
   return i;
}
```

No way to overflow into the stack

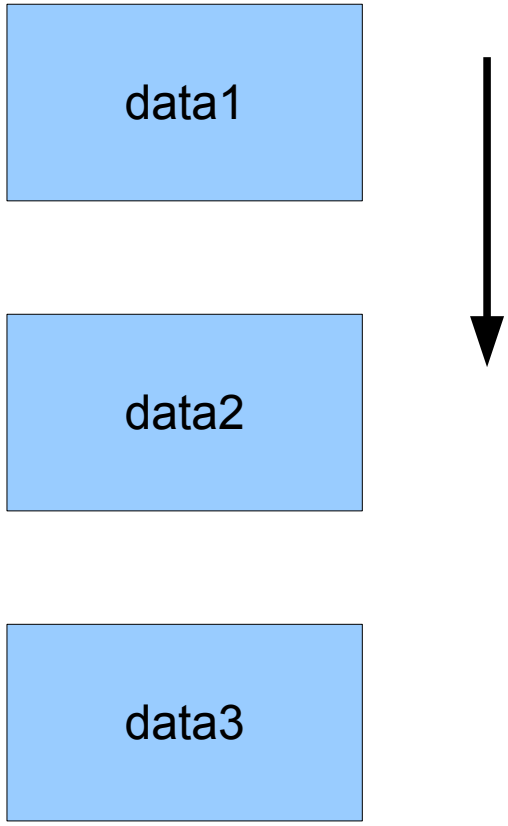
Something else?



Heap overflow?

Heap structure (simplified)

When you call **`malloc`** three times

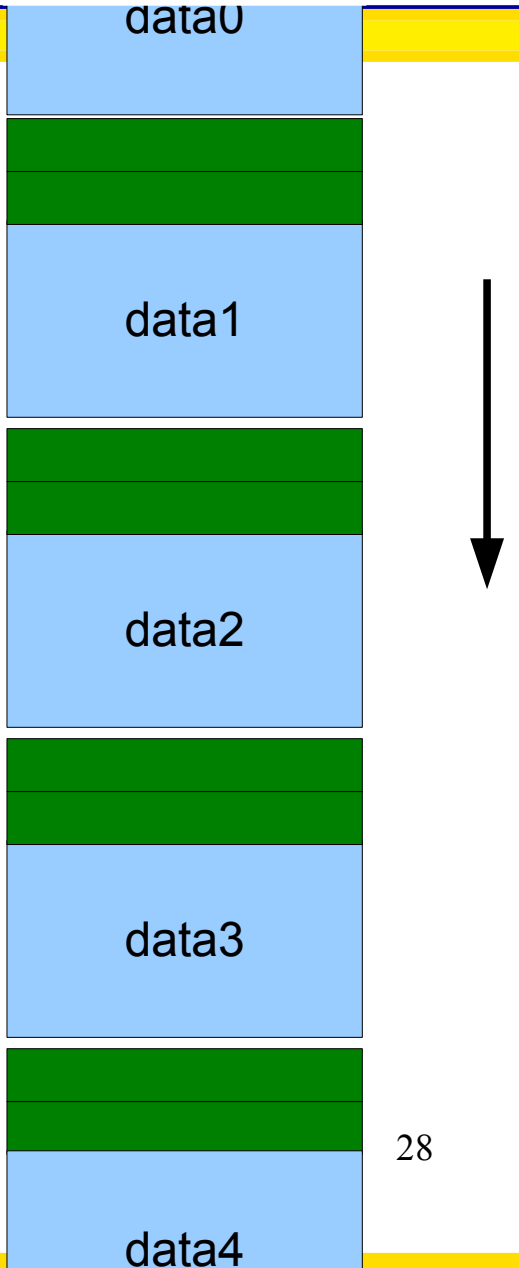


Heap overflow?

Heap structure (simplified)

When you call **`malloc`** three times

There are actually some additional headers



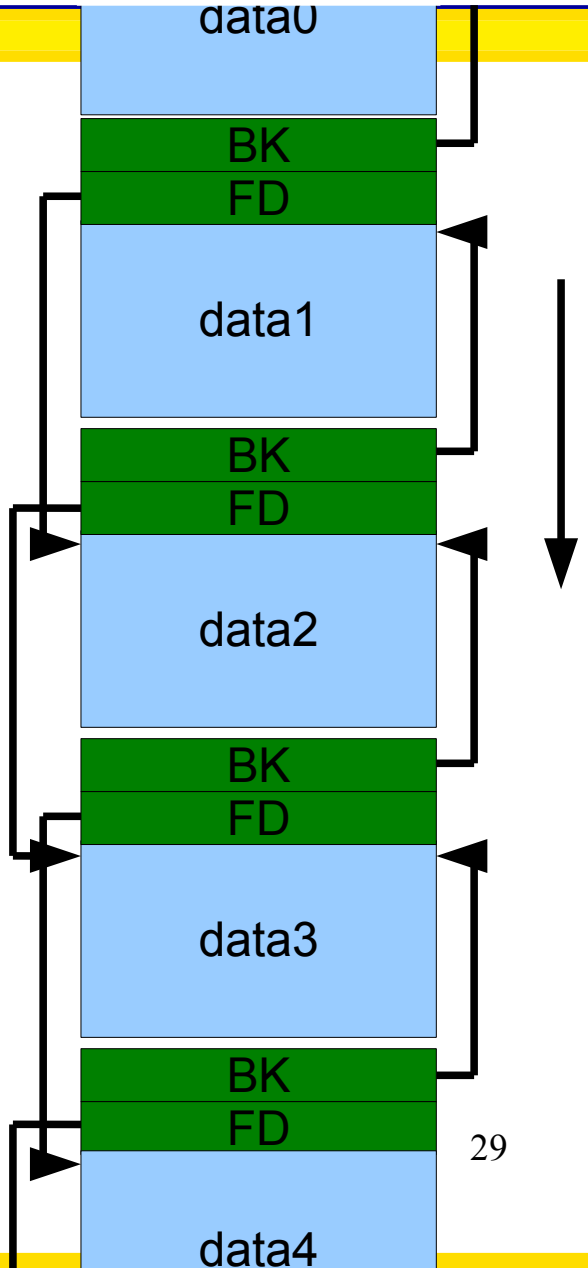
Heap overflow?

Heap structure (simplified)

When you call **`malloc`** three times

There are actually some additional headers
Which allow to jump between allocations:

- BK (back)
- FD (forward)



Heap overflow?

Heap structure (simplified)

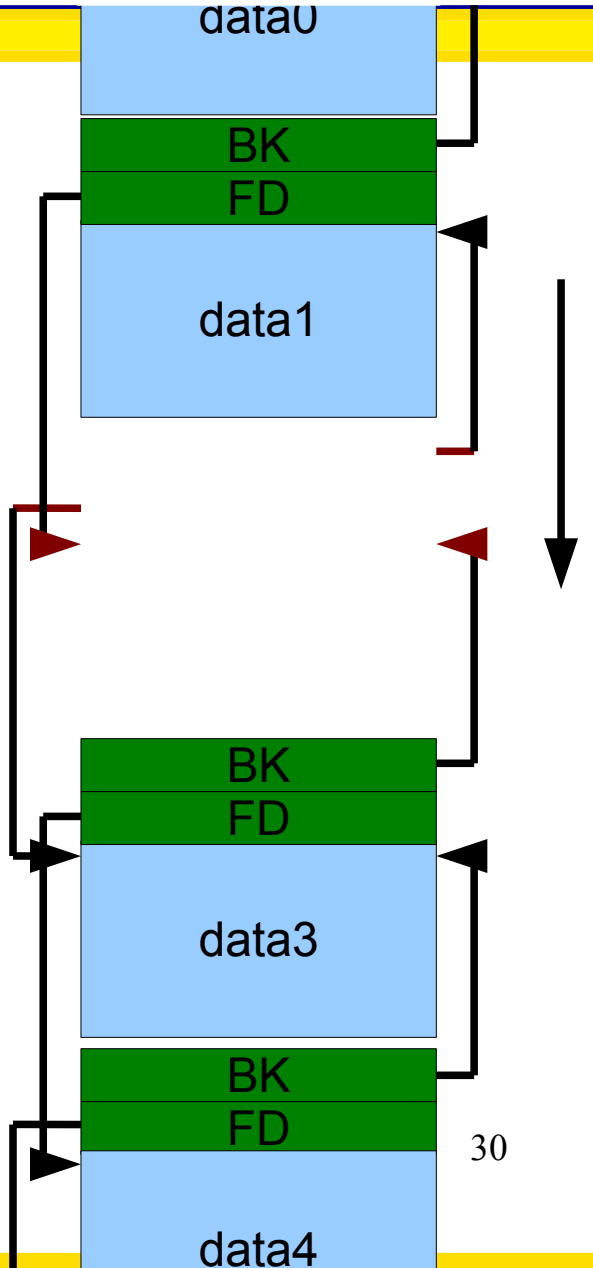
When you call **`malloc`** three times

There are actually some additional headers
Which allow to jump between allocations:

- BK (back)
- FD (forward)

Now free(data2)

- Have to update BK/FD



Heap overflow?

Heap structure (simplified)

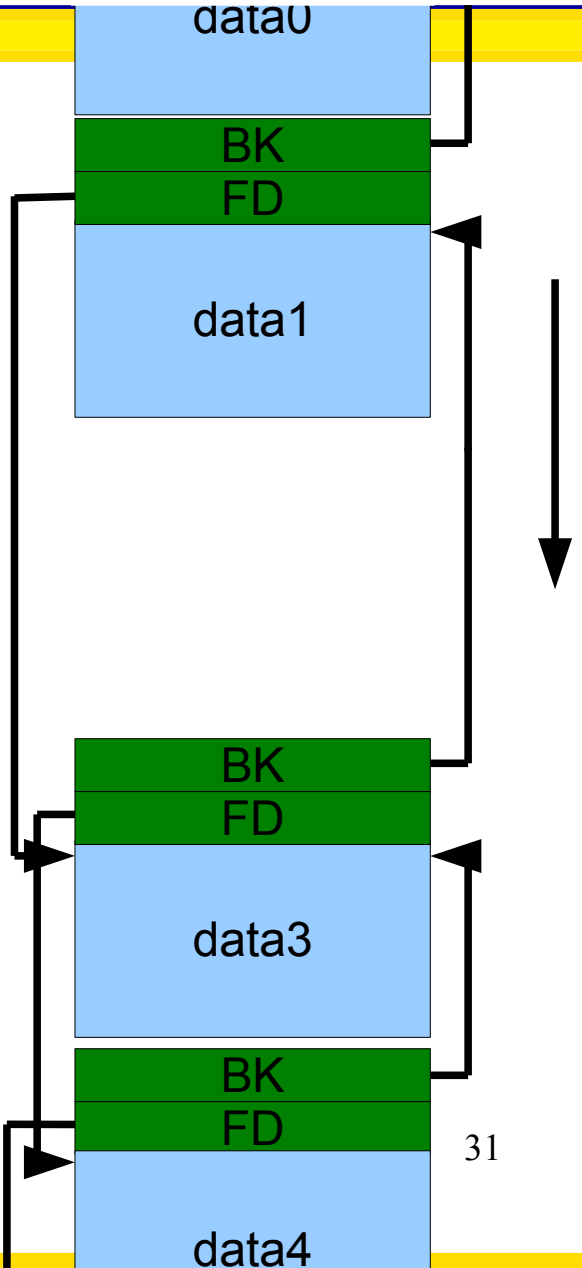
When you call **`malloc`** three times

There are actually some additional headers
Which allow to jump between allocations:

- BK (back)
- FD (forward)

Now free(data2)

- Have to update BK/FD



Heap overflow?

Heap structure (simplified)

When you call **`malloc`** three times

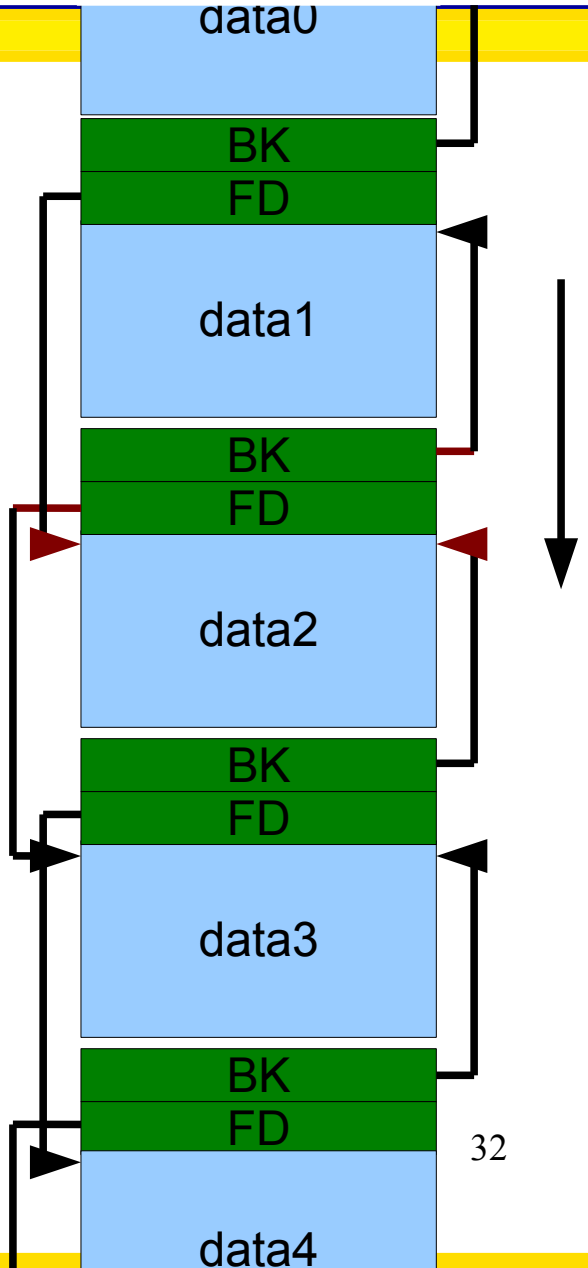
There are actually some additional headers
Which allow to jump between allocations:

- BK (back)
- FD (forward)

Now free(data2)

- Have to update BK/FD

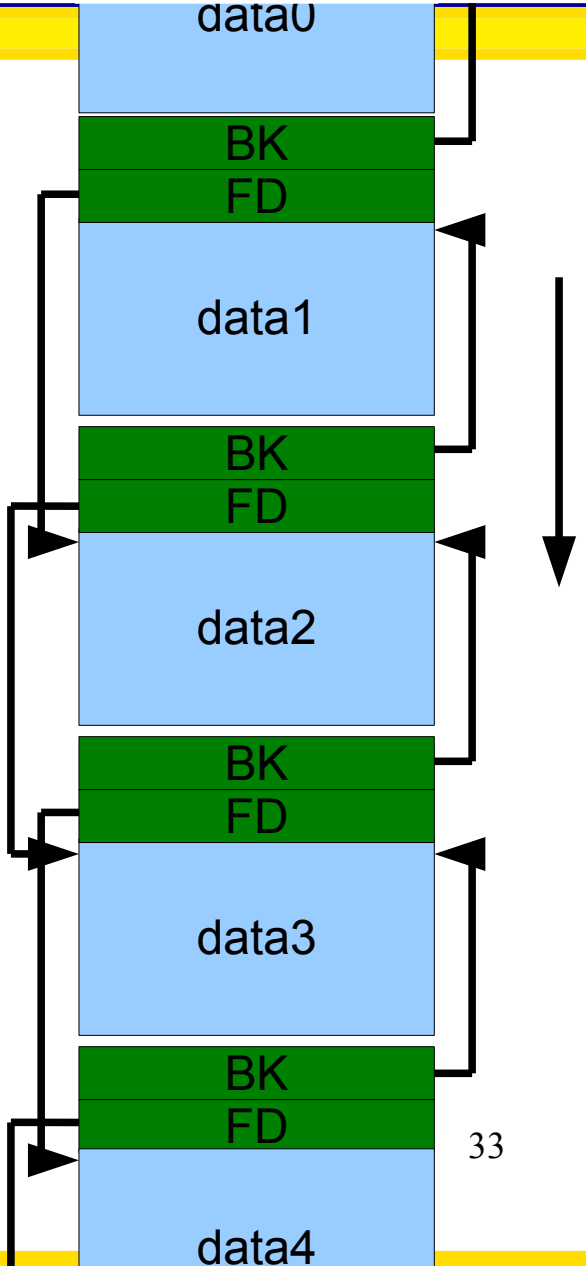
```
BK = data2->BK;
FD = data2->FD;
BK->FD = FD;
FD->BK = BK;
```



Heap overflow?

So basically free(data2) does

- BK = data2->BK;
- FD = data2->FD;
- BK->FD = FD;
- FD->BK = BK;



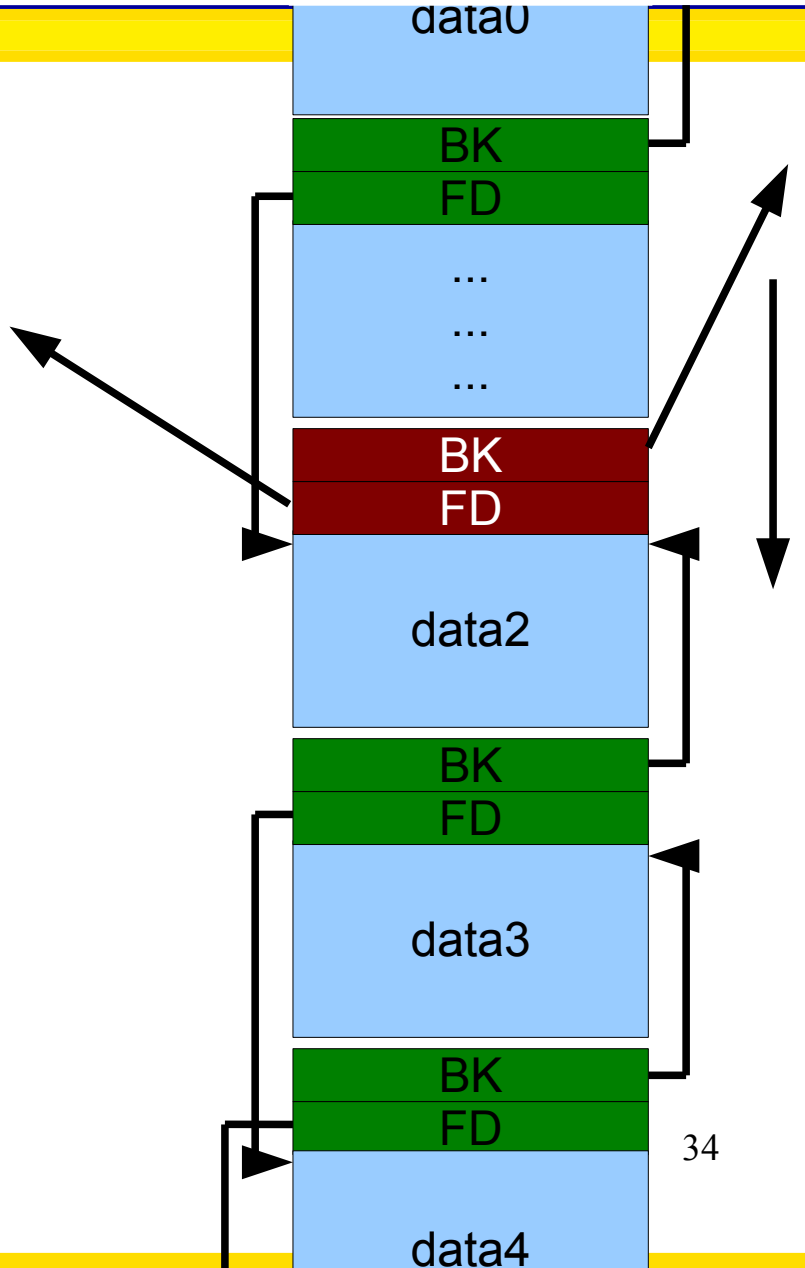
Heap overflow?

So basically free(data2) does

- BK = data2->BK;
- FD = data2->FD;
- BK->FD = FD;
- FD->BK = BK;

Now, what if I overflow data1?

- I can choose data2's BK at will
- I can choose data2's FD at will



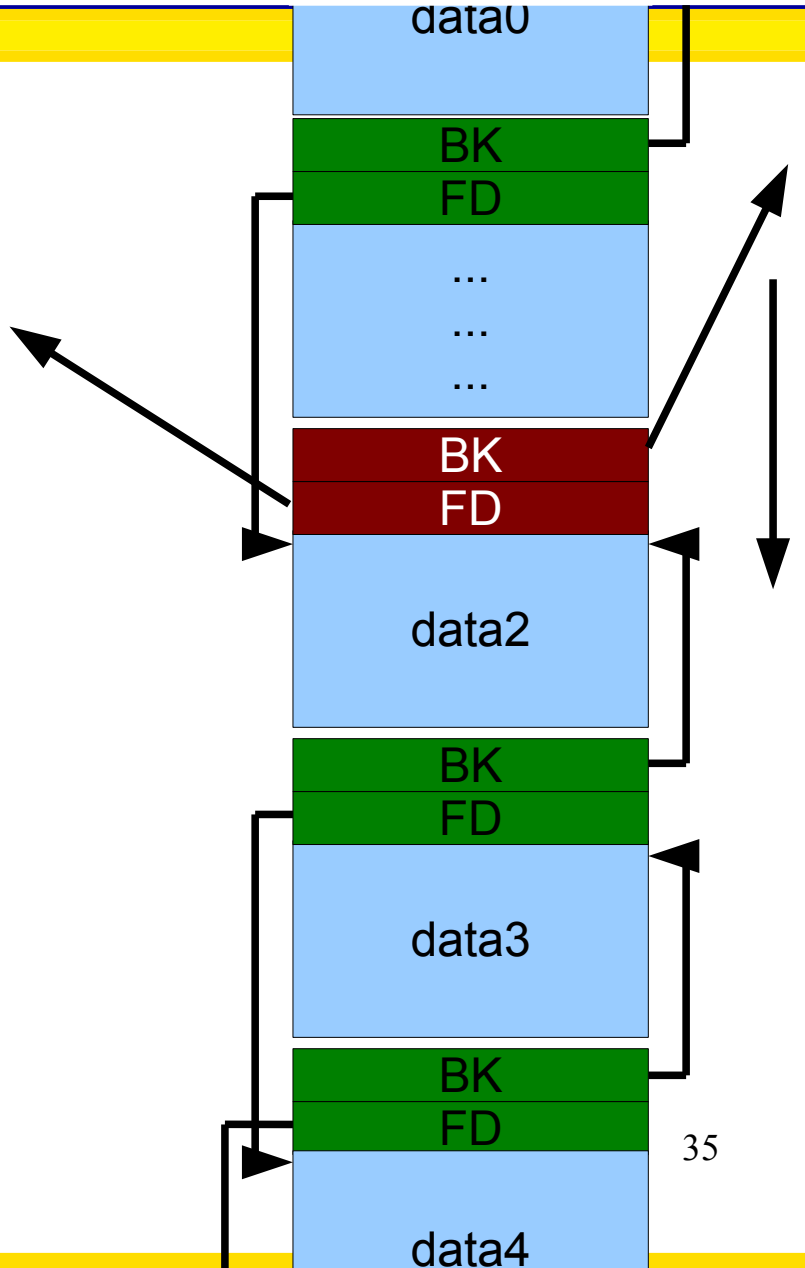
Heap overflow?

So basically free(data2) does

- BK = data2->BK;
- FD = data2->FD;
- **BK->FD = FD;**
- **FD->BK = BK;**

Now, what if I overflow data1?

- I can choose data2's BK at will
- I can choose data2's FD at will



Heap overflow?

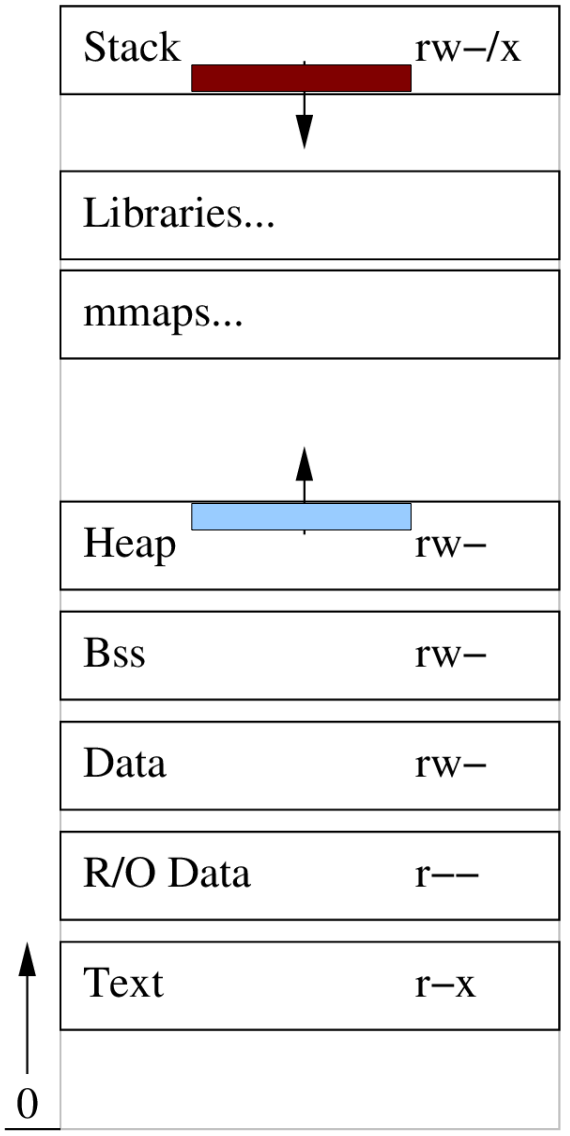
So basically free(data2) does

- BK = data2->BK;
- FD = data2->FD;
- **BK->FD = FD;**
- **FD->BK = BK;**

Now, what if I overflow data1?

- I can choose data2's BK at will
- I can choose data2's FD at will

In the end, I can divert the **control** in the stack



Countermeasures

Libc checks for coherency of headers

- More expensive free

Static code analysis / compiler-provided information

- Check for array bounds

Stack overflow exploit needs...

Stack overflow exploit needs

- Executable stack
- Buffer on the stack
- **Known position**
  - Next time!