

Engineering Robust Server Software

Vulnerabilities

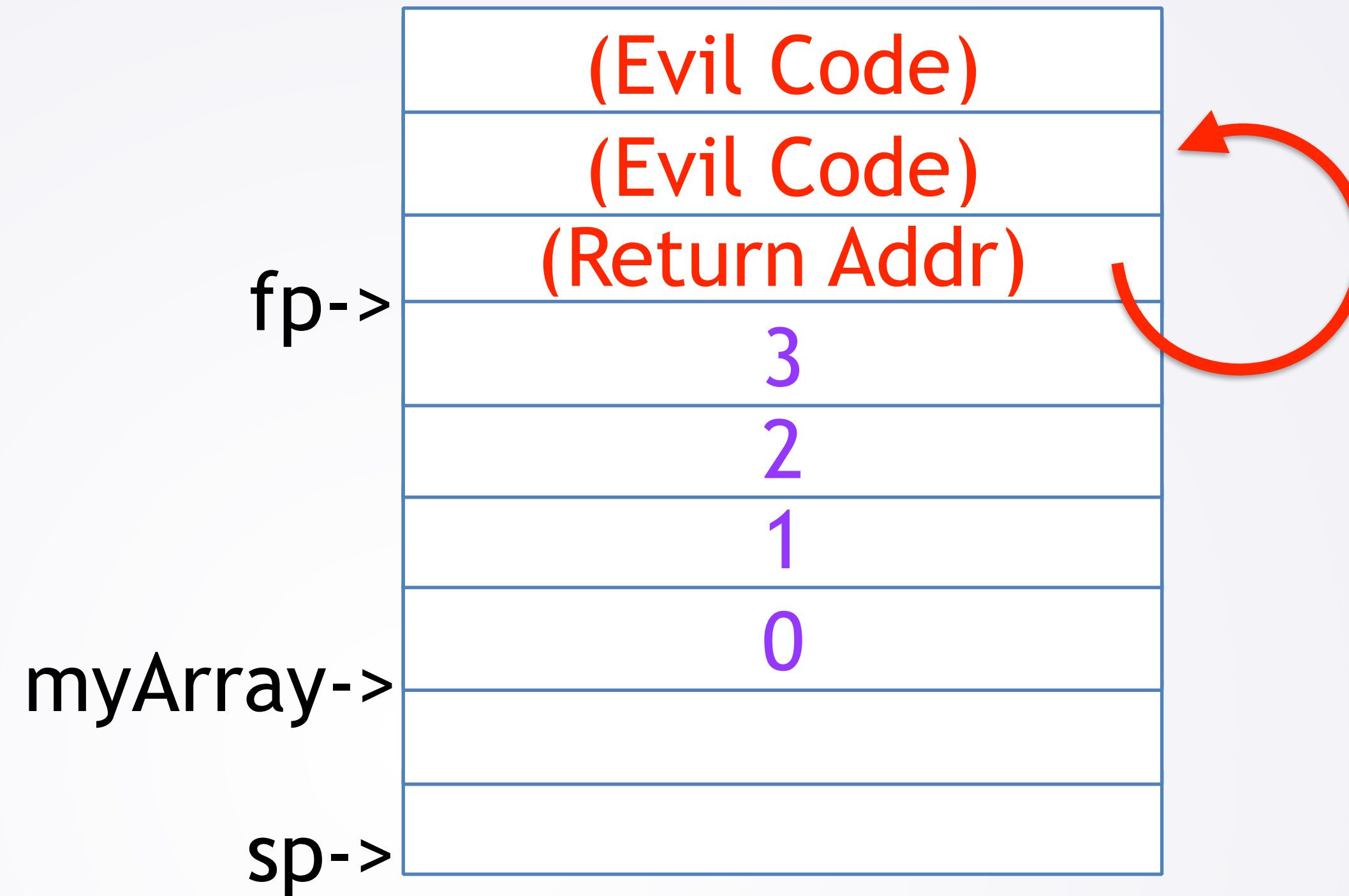
Common/Famous Vulnerabilities: Do Not Do!

- Common vulnerabilities
 - Buffer overflow
 - Failure to sanitize
 - SQL
 - Command injection
 - Cross-site Scripting (XSS)
 - Cross Site Request Forgery
 - Privilege Escalation
 - Time of check to time of use (TOCTTOU)
- Famous vulnerabilities: Dirty COW, Heartbleed, Apple goto

Buffer Overflow

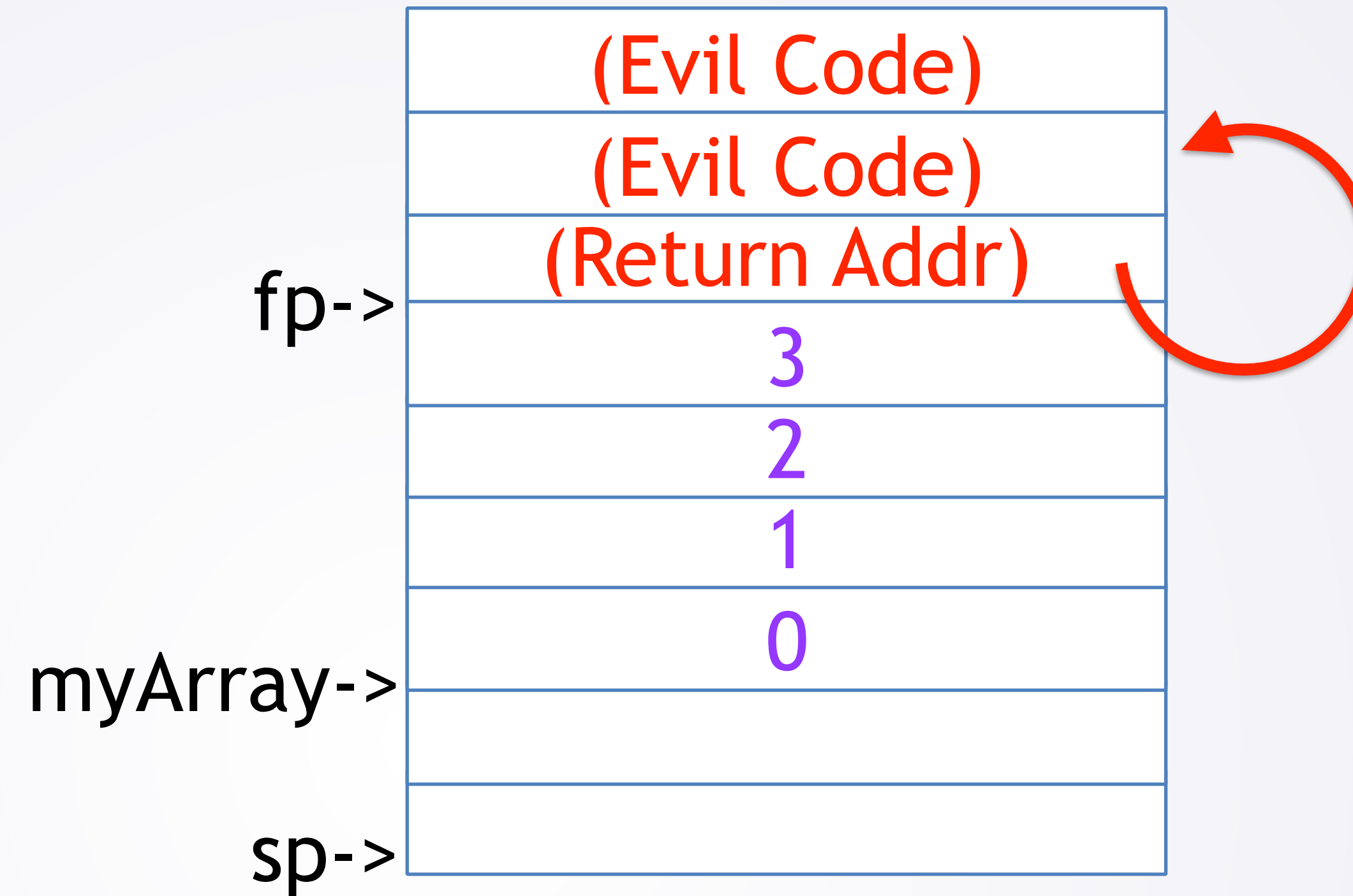
- Common security vulnerability: buffer overflow
 - Allow more data to be read into an array than space in that array
- Why is this so bad problem?

Buffer Overflow



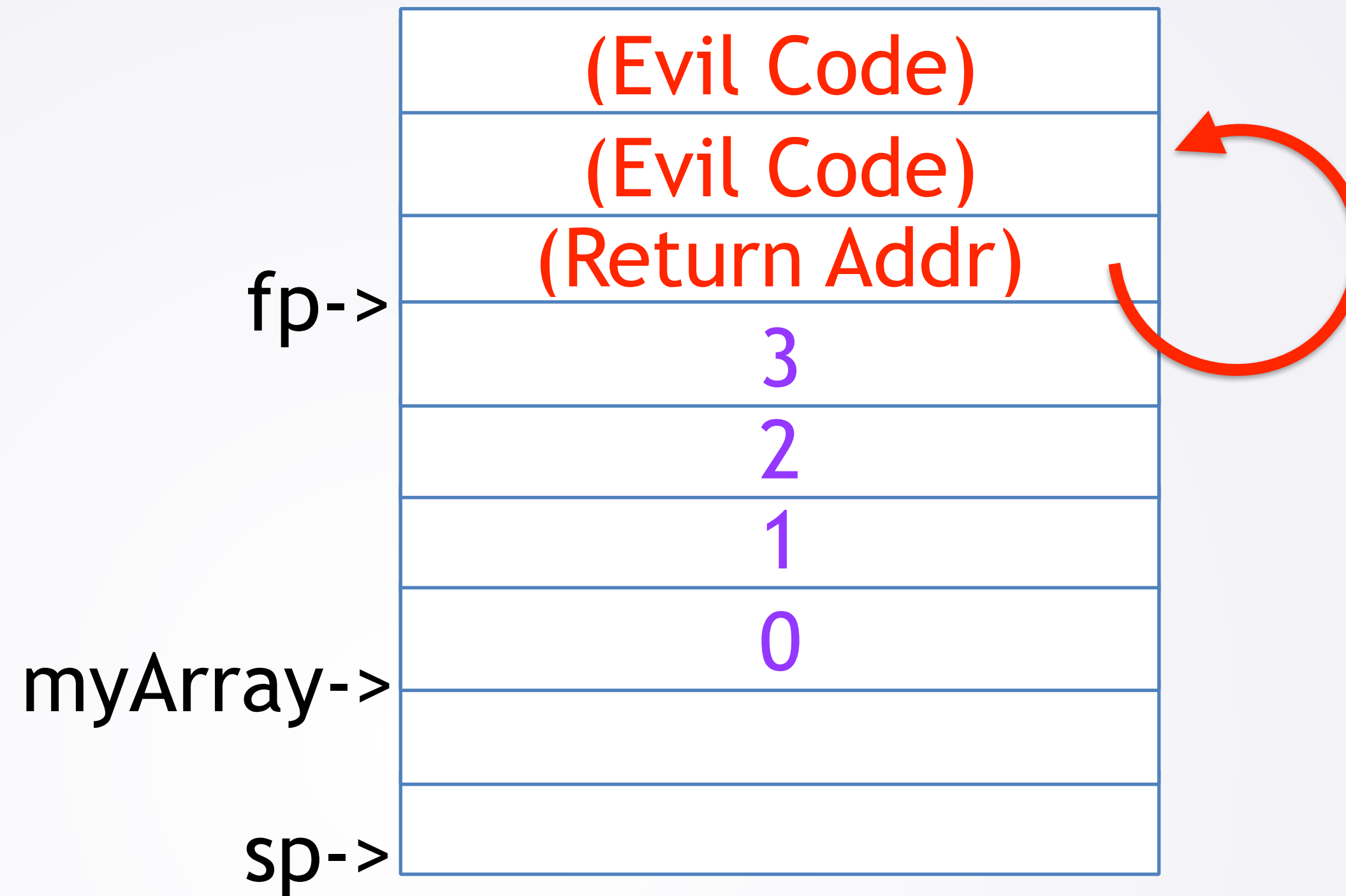
- Common security vulnerability: buffer overflow
 - Allow more data to be read into an array than space in that array
- Why is this so bad problem?

Buffer Overflow



- What happens when the function returns?

Buffer Overflow



- What happens when the function returns?
 - Begins executing instructions that were delivered by attacker!
 - Runs with same permission as whatever program
 - Running as root? Completed compromised.

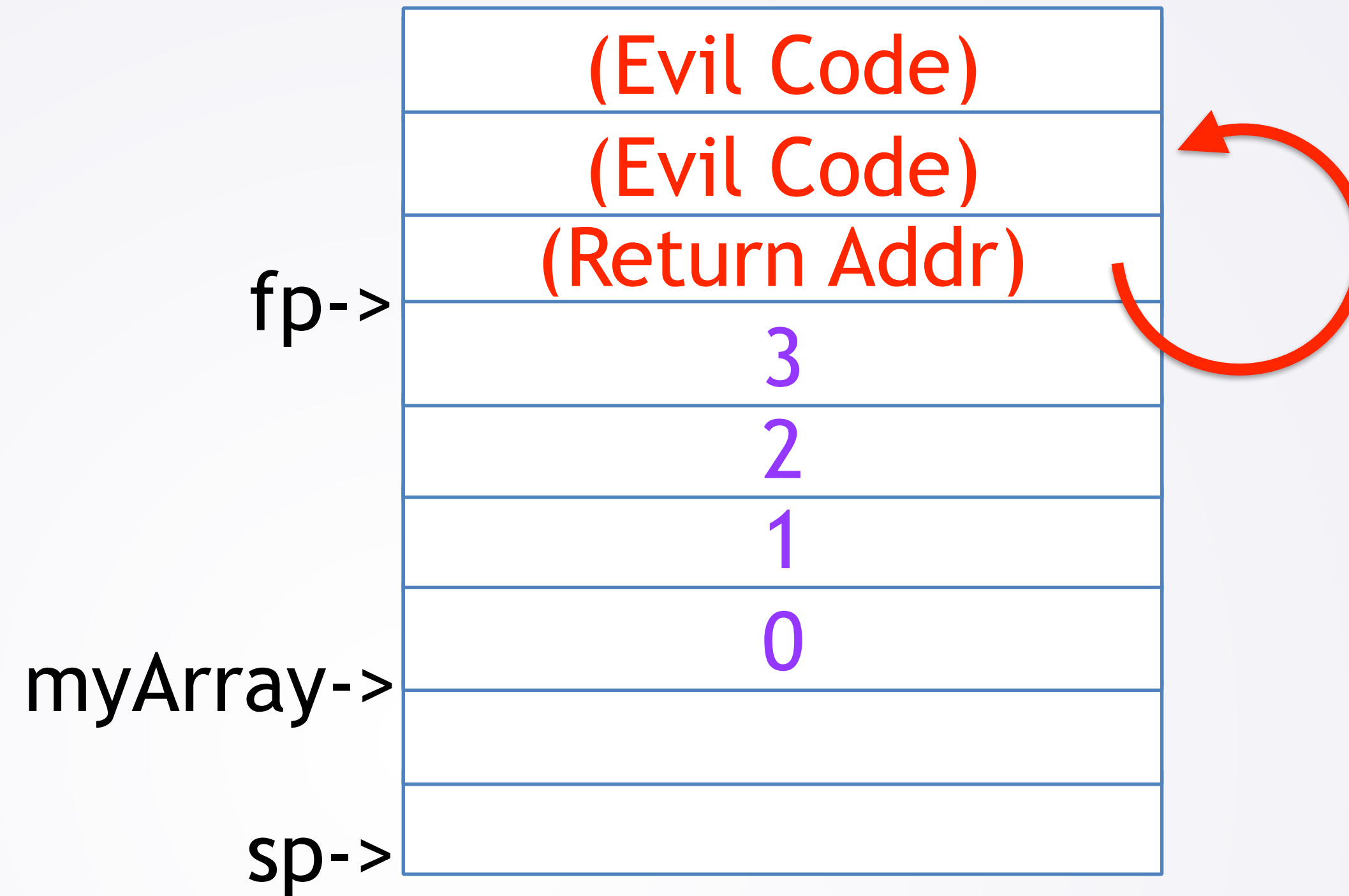
Buffer Overflows

- Buffer Overflows result from programmer carelessness
 - Use of terrible functions (e.g., `gets()`)
 - Assuming the user will not input more than a certain size
 - Not ensuring that space allocated matches size limit read
- Memory safe languages (Java, python, sml,...)
 - Not an issue: receive array index out of bounds exception (or similar)

No Execute Protection

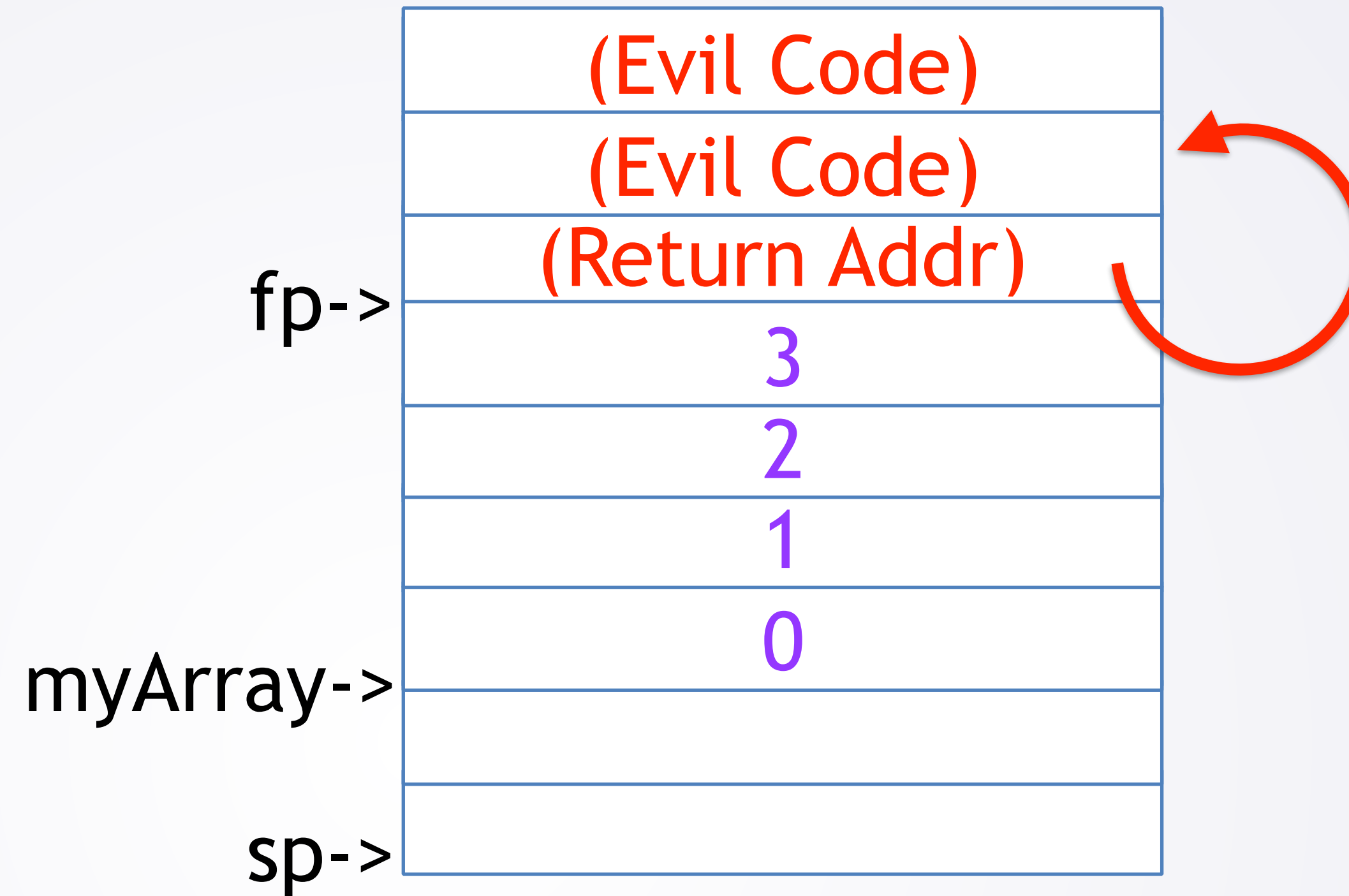
- Hardware defense: No Execute Protection (NX bit)
 - Mark stack pages as Read/Write/Non-executable
 - Available in Intel/AMD processors since early 2000s
- How does this help?

Buffer Overflow



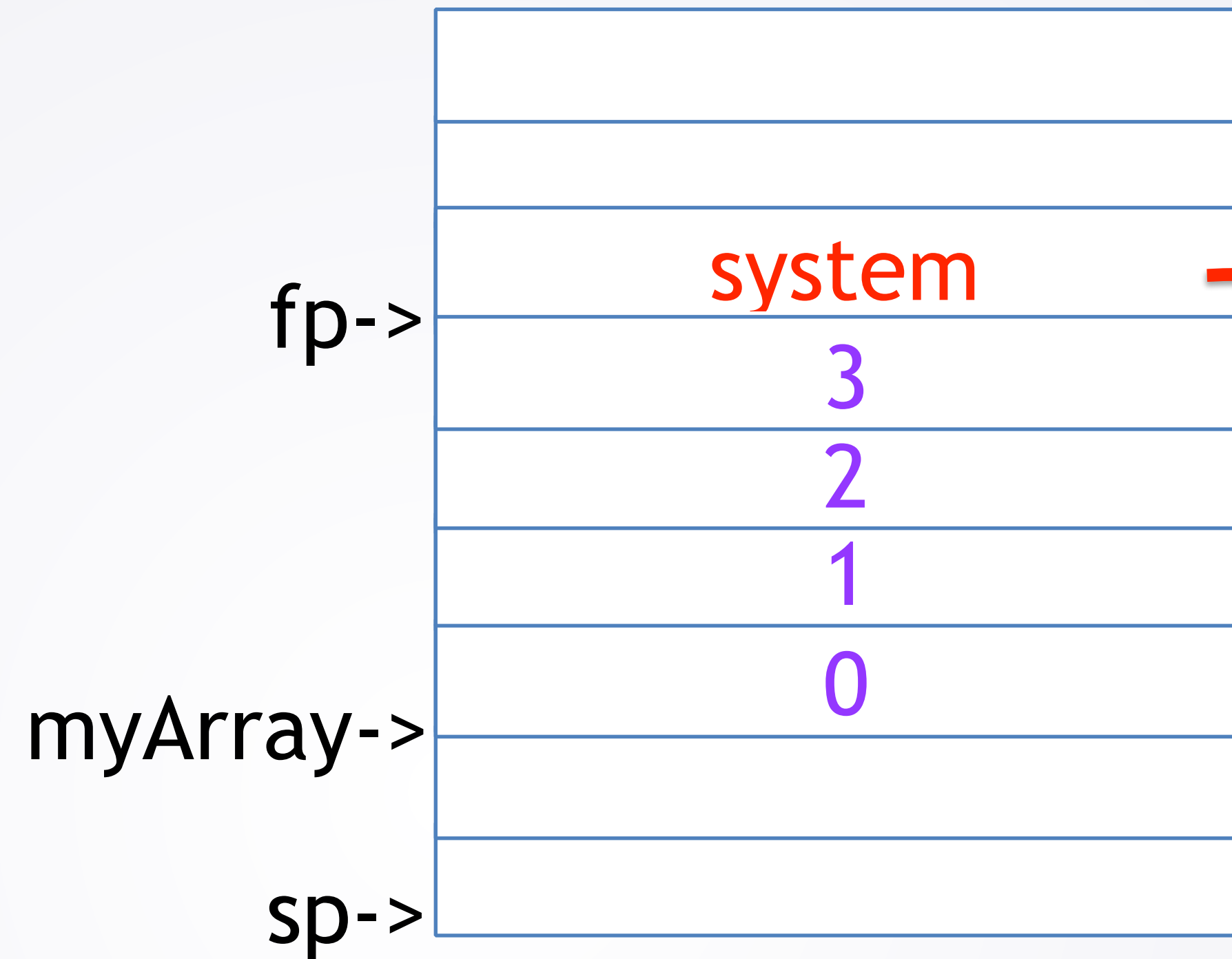
- If stack is not executable, returning to it -> segfault

Buffer Overflow



- If stack is not executable, returning to it -> segfault
 - Is this a perfect defense?

Return To libc Attacks



- Instead of returning to custom crafted code on stack
 - Return to existing code (often found in libc)
 - E.g., make the return address the system() library call...
 - Need to arrange for useful arguments

Operating System Defense: ASLR

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int x = 3;
    printf("%p\n", &x);
    return EXIT_SUCCESS;
}
```

```
drew@erss:~$ ./a.out
0x7fff67605974
drew@erss:~$ ./a.out
0x7fffa5689c44
drew@erss:~$ ./a.out
0x7ffc0adf03b4
drew@erss:~$ ./a.out
0x7ffdce2d1904
```

- Address Space Randomization
 - Loader randomly adjusts address layout

ASLR: Weaknesses

- ASLR is not perfect either
 - NOP slide: attack code starts with many NOPs
 - Attacker can succeed by guessing any location in NOP slide
- Similar ideas can be applied to other data
 - //////////////////////////////////////bin/bash
- Attacker may be able to learn information about layout
 - Format string injection
 - Timing attacks against branch predictor:
 - <http://www.cs.ucr.edu/~nael/pubs/micro16.pdf>

Format String Injection

%p %p

```
char * str = NULL;
size_t sz = 0;
if (getline(&str, &sz, stdin) > 0) {
    printf(str);
}
```



- How dangerous is this code
 - VERY

0x6020d2

0x7ffff7dd3790

0xa70

0x2070252070252070

0x7025207025207025

0xff00000000

0x602010

0x78

0xac2dc1a8e1744800

0x7fffffffef470

0x400692

0x4006a0

0x7ffff7a2e830

str

SZ

Saved FP

Saved RA

- Even more potential danger: %n conversions!
- From the man page for printf:

n The number of characters written so far is **stored** into the integer pointed to by the corresponding argument. That argument shall be an int *, or variant whose size matches the (optionally) supplied integer length modifier. No argument is converted. The behavior is undefined if the conversion specification includes any flags, a field width, or a precision.

Failure To Sanitize Inputs

- Format strings:
 - Example of data with special meaning (%)
 - We don't want the special meaning, but end up with it anyways :(
- For printf format strings best choice is to just
 - `printf("%s", theString);`
 - or use `puts` which does not format output
- Other situations: **sanitize** input
 - Remove, or escape special characters

SQL Injection



SELECT * FROM students WHERE name = '*student*';

SELECT * FROM students WHERE name = '*Robert*'); DROP TABLE Students;--';

Guarding Against SQL Injection

- Django:
 - Using built in model operations will sanitize vs SQL Injection
 - If you write RAW query strings, use an appropriate library
- Java:
 - Use PreparedStatement
- C++:
 - Use quote in pqxx::work

Command Injection

- Danger: using shell to execute command with user-input argument
 - some-command blah blah *userinput*
- What is the danger here?

Command Injection

- Danger: using shell to execute command with user-input argument
 - some-command blah blah *userinput*
- What is the danger here?
 - `some command`
 - xyz && another command
 - xyz || another command
 - | some command
 - ; another command

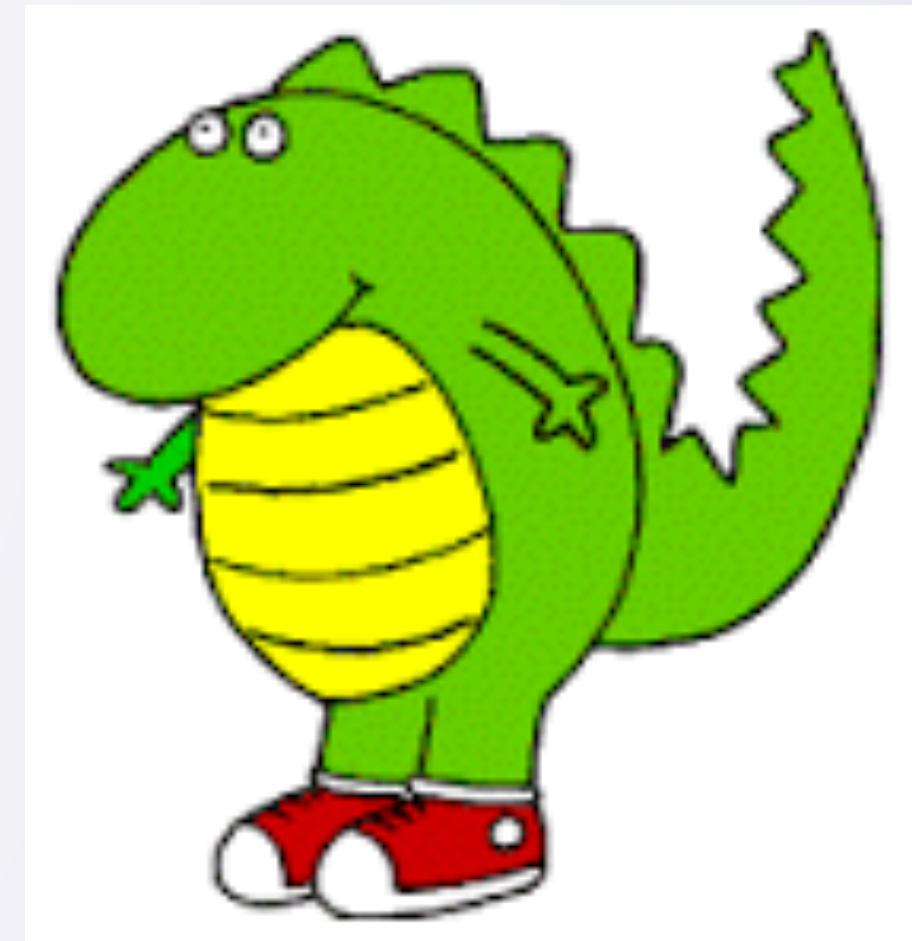
Cross Site Scripting

<https://bobsrecipes.com/contest/enter>

Welcome to Bob's recipe website!
We are hosting a contest for the best
recipes. Enter yours below to win!

Submit

I'll host a contest and
let users type in their
favorite recipes!



Cross Site Scripting

<https://bobsrecipes.com/contest/viewAndVote>

Recent entries in the recipe contest. Vote below

Halloumi with Date/Walnut Paste

Put 1 cup dates, 1/2 cup walnuts
1 tsp balsamic vinegar and 1 tbsp
warm water in the food processor.
Blend until it forms a thick paste.

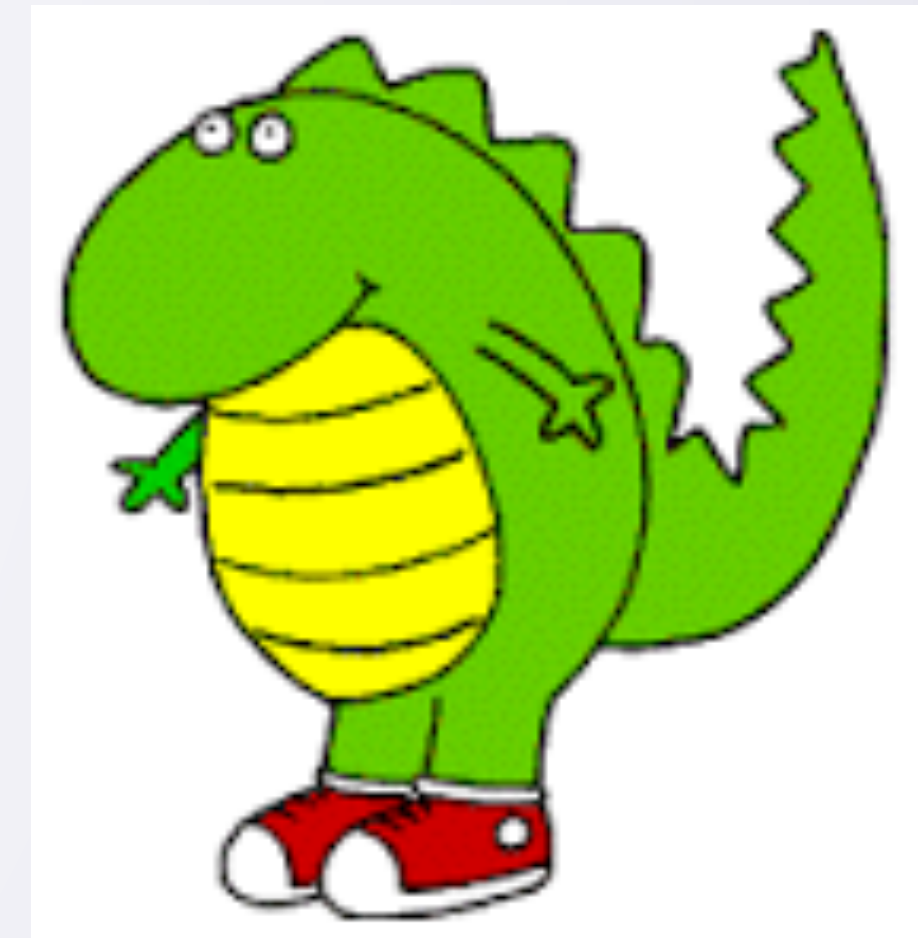


Chocolate Chip Cookies

Cream together 1/2 cup butter
3/4 cup brown sugar, and 3/4
cup sugar.



I'll host a contest and
let users type in their
favorite recipes!



Cross Site Scripting

<https://bobsrecipes.com/contest/enter>

Welcome to Bob's recipe website!
We are hosting a contest for the best
recipes. Enter yours below to win!

Submit

I'd love to participate
in this "contest"



Cross Site Scripting

<https://bobsrecipes.com/contest/enter>

Welcome to Bob's recipe website!
We are hosting a contest for the best recipes. Enter yours below to win!

Homemade Pitas

```
<script type="text/javascript">(evil code)
</script>
```

Dissolve 1 tbsp yeast in 1 cup warm water
Let sit for 5 minutes, until foamy

Submit

I'd love to participate
in this "contest"



Cross Site Scripting

<https://bobsrecipes.com/contest/viewAndVote>

Recent entries in the recipe contest. Vote below

Homemade Pitas

Dissolve 1 tbsp yeast in 1
cup warm water

Let sit for 5 minutes, until foamy

Mix in 1 tbsp sugar and 1/3 cup olive oil



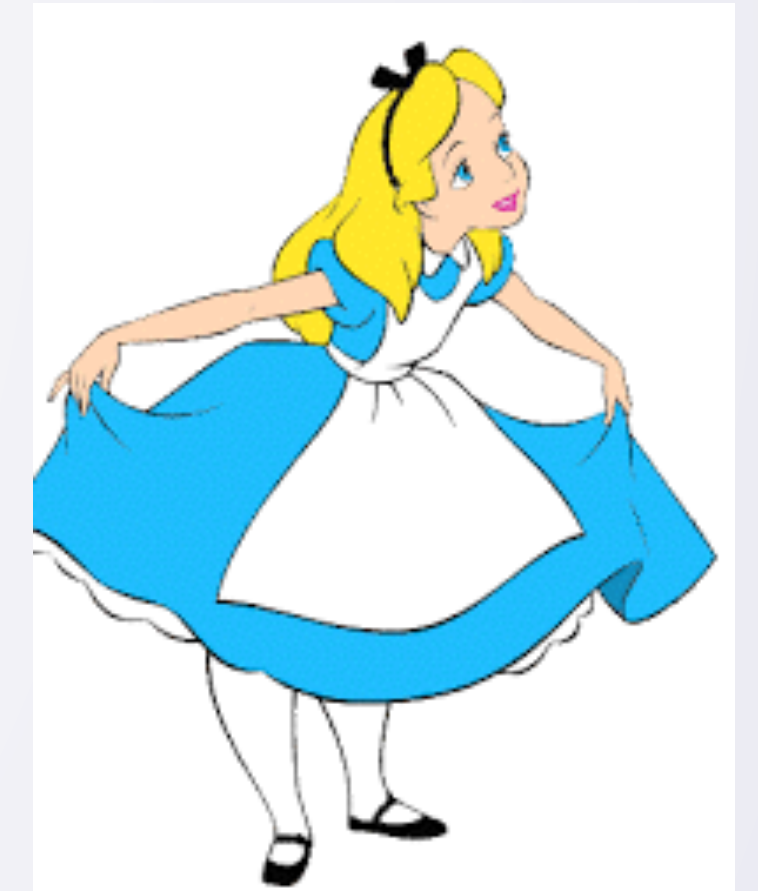
Halloumi with Date/Walnut Paste

Put 1 cup dates, 1/2 cup walnuts
1 tsp balsamic vinegar and 1 tbsp
warm water in the food processor.



Let's see how my
recipe is doing..

Ohh pitas!



Cross Site Scripting

<https://bobsrecipes.com/contest/viewAndVote>

Recent entries in the recipe contest. Vote below

Homemade Pitas

Dissolve 1 tbsp yeast in 1
cup warm water

Let sit for 5 minutes, until foamy

Mix in 1 tbsp sugar and 1/3 cup olive oil



Halloumi with Date/Walnut Paste

Put 1 cup dates, 1/2 cup walnuts
1 tsp balsamic vinegar and 1 tbsp
warm water in the food processor.



More like

Ohh **pwn3d!**



`<script type="text/javascript">(evil code)</script>`

Cross-Site Scripting

- Eve injects a `<script>` into HTML that will be viewed by other users
 - Alice's browser will run Eve's code
- Two main types
 - Persistent (what we saw): injected code stored on server
 - Reflected: injected code stored put in URL that user will click
- Vulnerable anytime you take un-sanitized data and display back to user
- Not to be confused with Cross Site Request Forgery (CSRF)

CSRF

- Cross Site Request Forgery:
 - Eve crafts a requests to change something
 - Gets Alice's browser to send that request while Alice is logged in
 - Alice's browser sends her authentication cookie
 - Site believes Eve's request

CSRF GETs (which shouldn't be a thing..)

- If site allows modification with GET requests
 - Bad! GET should be for reads only. Use POST!
 - Eve injects something like

``

- When Alice's browser loads this, it will try to GET that image..
 - If Bob's site allows this modification with GET, it will perform the action
- Note that Eve does not get to (nor need to) see the response

CSRF POST

- Ok, so Bob's site doesn't allow modifications with GET. Safe?
- No: Eve can still craft malicious POST requests
 - E.g., she can make a `<form>` and have a `<script>` submit it
- How to defend?
 - Generate random token which must be in POST data
 - Eve has a hard time guessing
- Django requires this by default for POSTs:
 - Put `{% csrf_token %}` inside `<form>` that will be sent back to YOUR site
 - Do not leak token to other sites!
 - Django handles the rest

Consider The Following Psuedo-Code

```
File f = openFile(inputCommands);  
  
for each line in f  
    if(!checkUserCanExecute(line, currentUser))  
        return false;  
  
rewind(f);  
  
for each line in f  
    execute(line);  
  
close(f);  
return true;
```

Consider The Following Psuedo-Code

```
File f = openFile(inputCommands);
```

```
for each line in f  
    if(!checkUserCanExecute(line, currentUser))  
        return false;
```

```
rewind(f);
```

```
for each line in f  
    execute(line);
```

```
close(f);  
return true;
```

I have a plan... Anyone see it?



Consider The Following Psuedo-Code

```
File f = openFile(inputCommands);
```

```
for each line in f  
    if(!checkUserCanExecute(line, currentUser))  
        return false;
```

```
rewind(f);
```

```
for each line in f  
    execute(line);
```

```
close(f);  
return true;
```

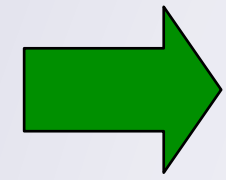
commands.txt:

```
change Eve's password to xyzzy42  
print Eve's Account Balance  
print Eve's Last Action
```



I have permission to execute all these commands....

Consider The Following Psuedo-Code



```
File f = openFile(inputCommands);
```

```
for each line in f
    if(!checkUserCanExecute(line, currentUser))
        return false;
```

```
rewind(f);
```

```
for each line in f
    execute(line);
```

```
close(f);
return true;
```

commands.txt:

```
Ichange Eve's password to xyzzy42
print Eve's Account Balance
print Eve's Last Action
```



Let's run this program..

Consider The Following Psuedo-Code

```
File f = openFile(inputCommands);
```

```
for each line in f
```

```
➡ if(!checkUserCanExecute(line, currentUser))  
    return false;
```

```
rewind(f);
```

```
for each line in f  
    execute(line);
```

```
close(f);  
return true;
```

commands.txt:

change Eve's password to xyzzy42

I print Eve's Account Balance

print Eve's Last Action



While this runs...
One quick change to input file...

Consider The Following Psuedo-Code

```
File f = openFile(inputCommands);
```

```
for each line in f
```

```
➡ if(!checkUserCanExecute(line, currentUser))  
    return false;
```

```
rewind(f);
```

```
for each line in f  
    execute(line);
```

```
close(f);  
return true;
```

commands.txt:

```
change Alice's password to xyzzy42  
Iprint Eve's Account Balance  
print Eve's Last Action
```



Bwahahaha!
While this runs...
One quick change to input file...

Example of TOCTTOU Attack

- Time of Check To Time Of Use
 - Race condition between **validation** and **use** of data
 - Attacker can present valid data
 - Then change the data before it is used
- Defense:
 - Ensure that data cannot be changed between validation and used
 - Previous example, either:
 - Execute each command as read
 - Read file into memory, then validate/execute from memory

Privilege Escalation

```
[eve@linux] $ cat ~alice/secret.txt  
ls: /home/alice/secret.txt : Permission denied  
[eve@linux] $
```



I wish I
were root..

What Might Eve Do?

- Find bug in setuid binary (or service running as root)?
 - "trick" it into doing privileged actions for her
- Find files with wrong permissions
 - Shouldn't be suid but is?
 - Is writeable but shouldn't be?
- Exploit kernel bug?
 - Dirty COW: up next

Privilege Escalation: Not Just Shell

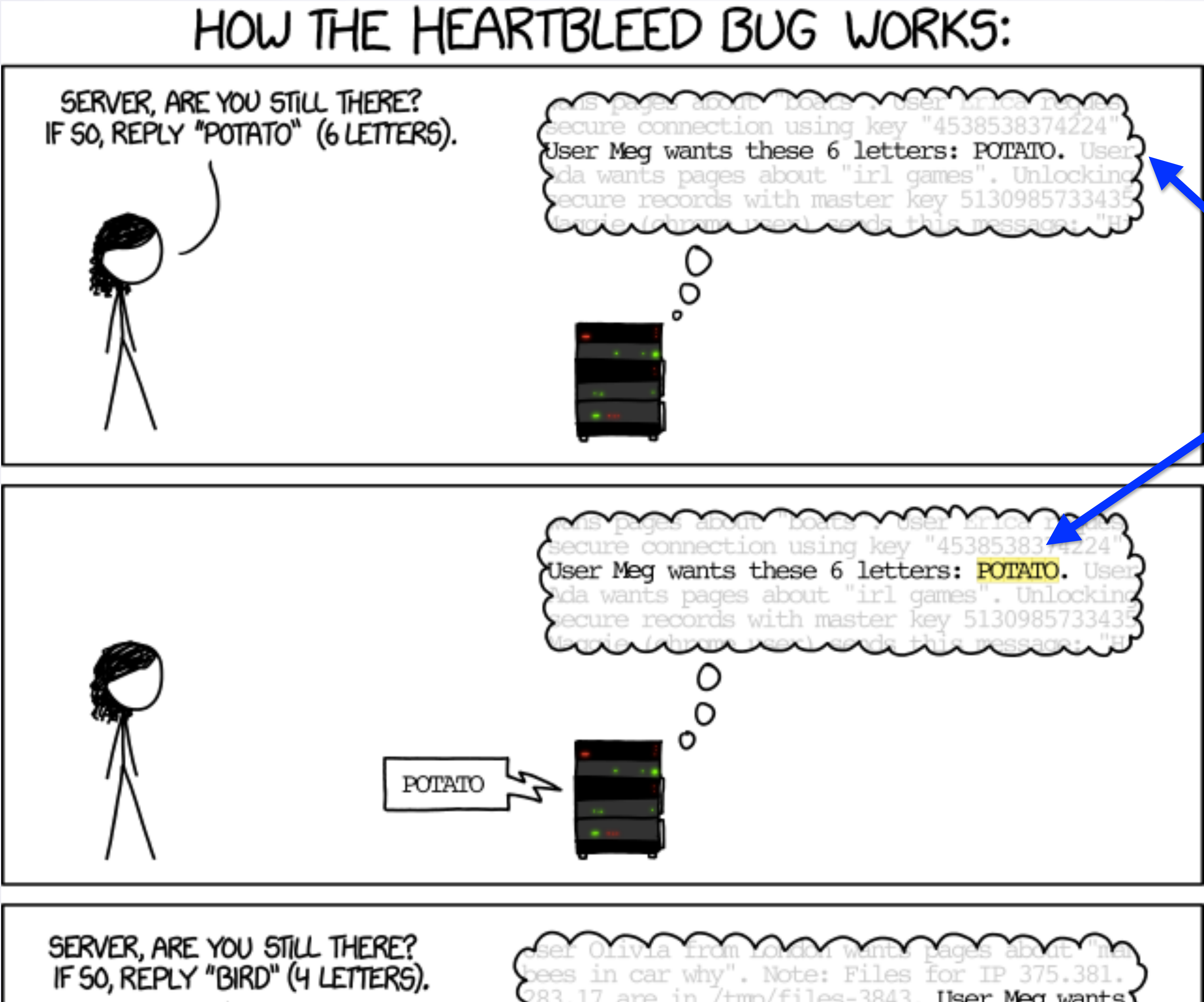
- Can have privilege escalation bugs in other settings
- Webapp:
 - Can Eve alter her permissions?
 - E.g, Admin functionality w/o proper checks?

Dirty COW

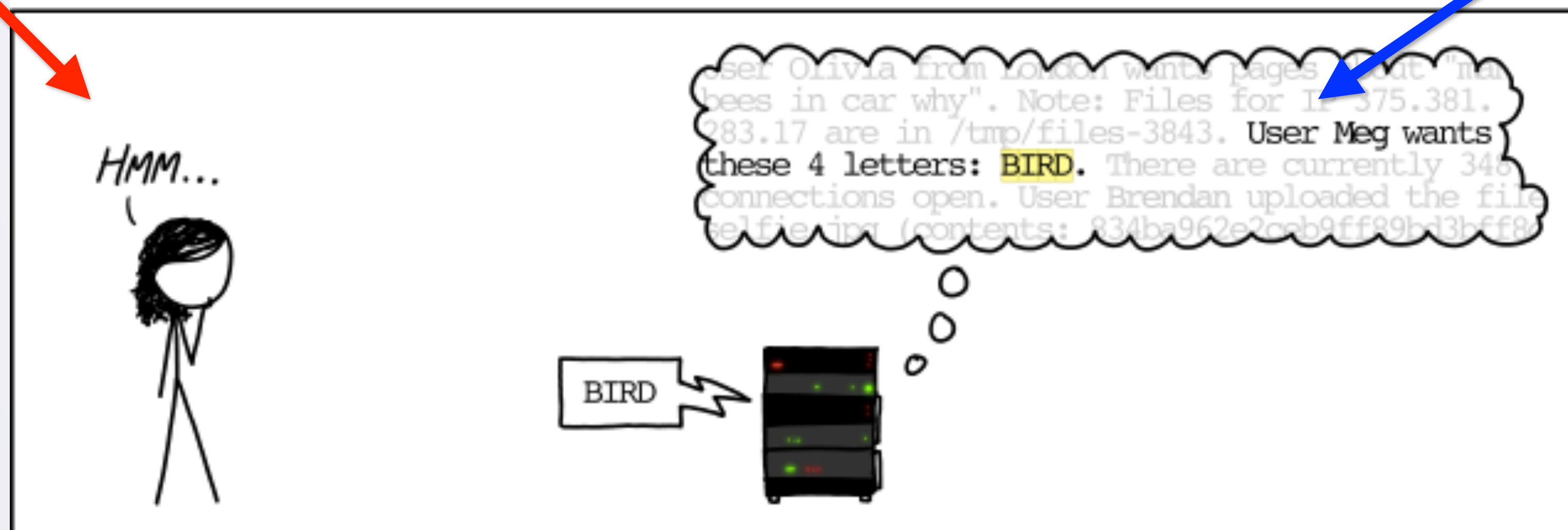
- Serious Linux kernel vulnerability (fixed 2016)
 - Race condition in COW handling
 - Could allow writing read-only data
 - mmap file read only
 - End up writing to file!
 - Allowed privilege escalation:
 - User could become root
- Linux includes Android
 - Could be used to "root" Android devices
- <https://raw.githubusercontent.com/dirtycow/dirtycow.github.io/master/dirtycow.c>



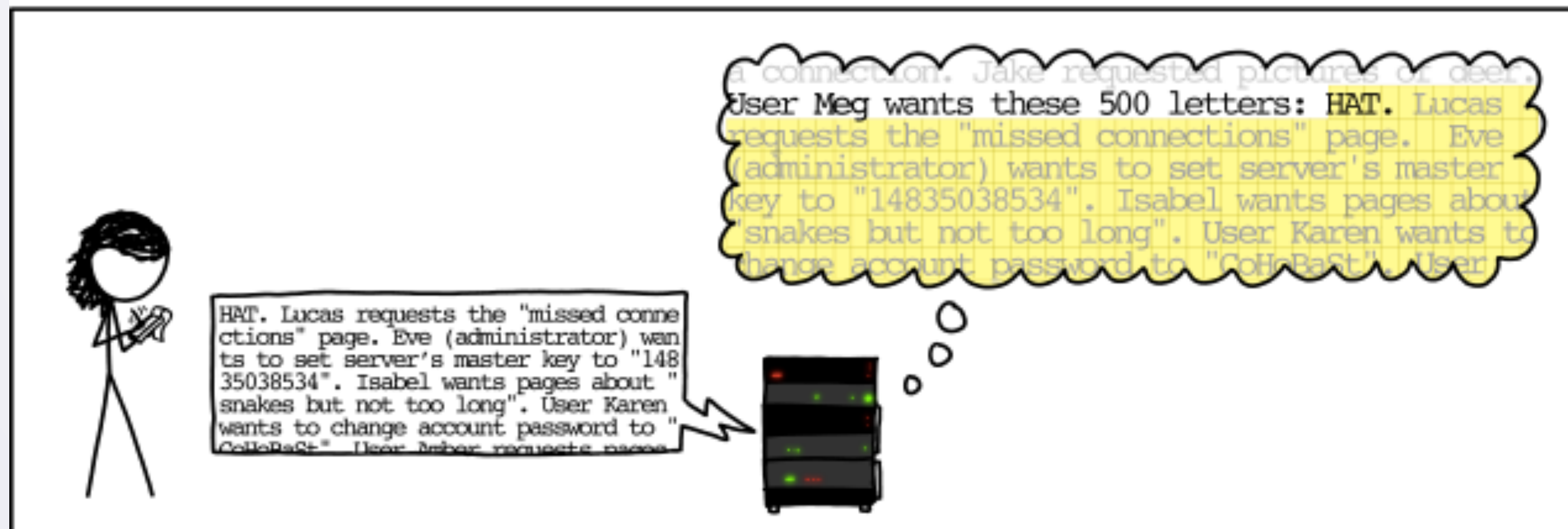
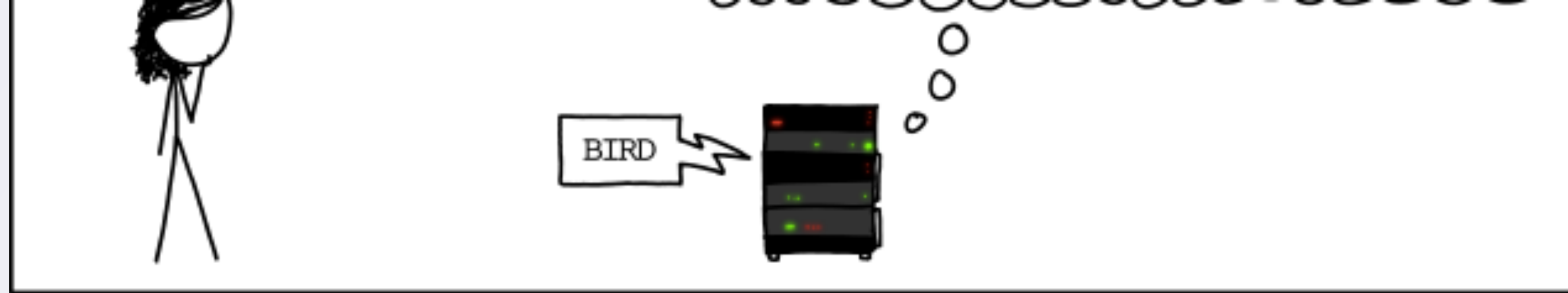
Heartbleed (Explained by xkcd)

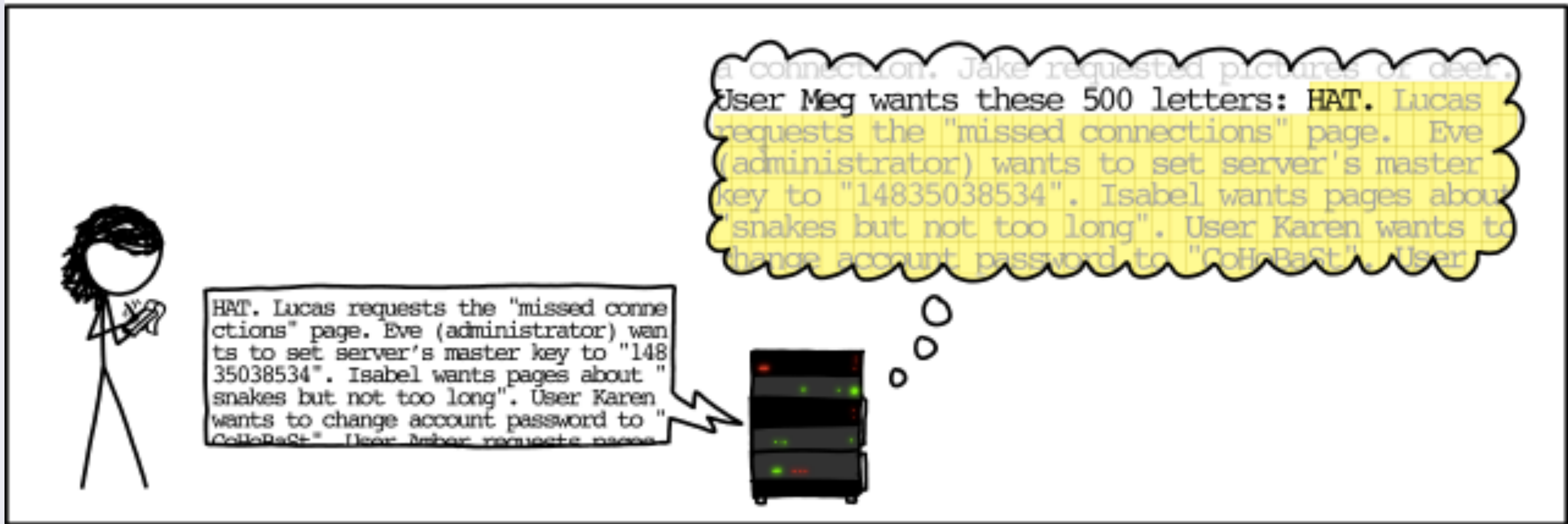


TLS Heartbeat



TLS Heartbeat





Apple Goto

```
...
if ( (err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ( (err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ( (err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ( (err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);

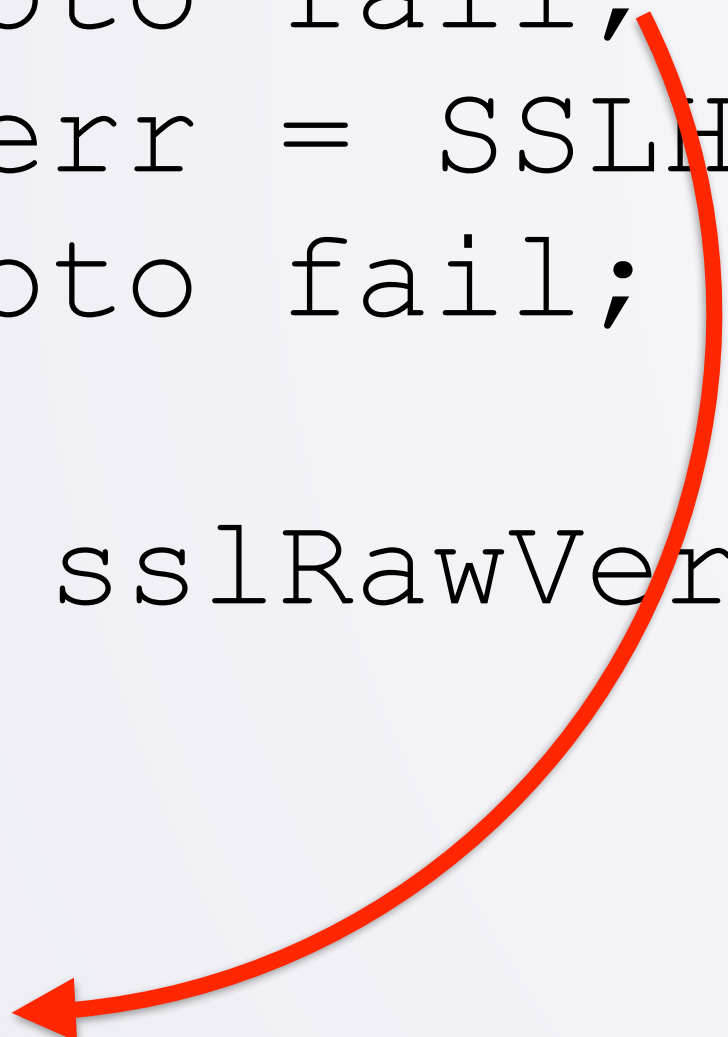
...
fail:
    //free memory, etc
    return err;
```

Apple Goto

```
...
if ( (err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ( (err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ( (err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0) {
    goto fail;
} goto fail;
if ( (err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);

...
fail:
    //free memory, etc
    return err;
```



Apple Goto

- Always use {} for bodies of anything (if, while, for, do)
- Test test test test!
 - There should have been a test case for this...
 - There should have been a test case for every one of those failing!

Barely Scratched The Surface..

- Cyber security experts?
 - We've barely scratched the surface!
- Covered the basics (most important/common things)
 - Use encryption (AES + RSA)
 - Hashes? Use SHA-256 or SHA-512 [and PBKDF2]
 - Variety of exploits:
 - Code carefully!
 - Think like a hacker