Question 1

(a)

On my test system the stack looks like: (used gbp to confirm)

```
0xbfffefd0:              <- esp
...
0xbfffefe8: a+0x00       <- ebp-0x20
0xbfffefec: a+0x04
0xbffff0f0: a+0x08
0xbffff0f4: a+0x0c
0xbffff0f8: a+0x10
0xbffff0fc: i            <- ebp-0x0c
0xbffff000:
0xbffff004:
0xbffff008: saved bp     <- ebp
0xbffff00c: ret          <- ebp+0x4
--------------------------------
0xbffff010: &in          <- ebp+0x8
0xbffff014: len          <- ebp+0xc
```

a, i, ret and bp in order (1)

- Allow canary to be inserted at ebp-0x0c and everything to be shifted down by 4.
- Allow in and len from previous frame to be missed.

(b)

27; however it is likely that this won't happen and we'll segfault before that.

```
addr of
last byte   sizeof(uint64_t)
(20 -1)  +  8
```

One point for answer.

(c)

- Because each call to memcpy writes 8 bytes, we can overflow a and write into i. (1)

- If we can set i to something negative we can write to before a and then write continue to write upwards. (1)
- This may allow us to write to unallocated memory and crash the program, or worse (1)
- Any value of len > 13 (i.e. 14 and above) will allow us to alter the value of i as we'll overflow the array (1)

(accept any two for two marks)

(d)

- A canary is a random value placed onto the stack. (1)
- No program logic should change the canary value. (1)
- Before returning from a function we check the canary. (1)
- If its value has changed we assume someone has been playing around with the stack and crash. (1)
- Canary will help us detect if we overflow past a, but won't detect i being modified as i is after a in memory. (1)

(accept any three for three marks)

(e)

- It would be tricky.
- We cannot overflow past a enough to reach the return address, so a "classic" stack smashing attack is not possible. (1)
- We could reset i to be zero and trigger an infinite loop. (1)
- If we try and write past i (we can write at most three bytes past) we might alter some memory and maybe the saved bp (depending on memory layout) (1)
- A canary would detect this modification and prevent it (1).

Question 2

(a)

- A stored XSS attack relies on a script that is served as part of the web page, for example a forum post with an embedded script. (1)

- A reflected XSS attack relies on a script that is sent with the request and is copied into the web page dynamically. (1)
- If a user can inject JavaScript they can run programs in another users browser. (1)

(b)

- `http://vulnerable.net/<script>alert("XSS!");</script>` (1)
- Need to convince victim to click on link (1)

(c)

- Set up webserver to harvest responses. (1)
- Craft URL to send cookie to server (1)
- For example request an image from webserver that has the cookie as part of its path. (1)

(d)

(three marks)

```
Content-Security-Policy: default-src https://vulnerable.net;
  frame-src: 'none'
  img-src: 'self'
  script-src: 'self'
  font-src: 'https://themes.googleusercontent.com'
```

(e)

- No.
- CSP not supported by all browsers.
- Browsers can ignore policy.
- Doesn't fix underlying problem that JS is being injected. Better fix would be to sanitize the input before displaying.

(accept any two for two marks)