Web application security testing - A Guardian website example.

12 July 2013 · guardian, investigation, questioning, security

When you read a blog post like this, or an article on a website, can you be sure its the ‘real thing’? How would you know if it had been doctored?

Lets assume the ‘server’ is fairly secure and hasn’t been hacked into. So the content is going to be OK isn’t it?, it looks OK..? So we’ve checked the location bar at the top of our web browser and it definitely has the right website/company name. No funny-looking misspelled names, possibly meaning I’m reading a fake site.

And to be doubly sure, the browsers location bar states its using HTTPS and even has that reassuring little padlock we’ve come to look for and trust. OK, so to recap:

- The website’s server is secured. (Well - for the the purposes of this, lets give them the benefit of the doubt)
- The logo, words, content and layout all appear to be kosher.
- We are using the correct website address. (No unusual spellings e.g.: www.goole.com etc)
- The page is secured using HTTPS. (Warm glow from the on-screen padlock)

(Don’t worry - this actual page is not secured via HTTPS, unlike our hypothetical example above)

An increasing part of my testing is application-security related, investigating websites to answer just these sorts of questions. A few months ago, In my own time, I took a quick look at the Guardian website. I’ve used the Guardian as an example before (http://www.investigatingsoftware.co.uk/search/label/guardian), as well as interesting news they have have some cool API tools (http://www.guardian.co.uk/open-platform) to learn with. Like many news websites, the Guardian lets users create an account, and log in. This log-in form is essentially the front end to the Guardian’s id.guardian.co.uk system, and like all software it has problems - things that can upset its users or owners.

Similar to ‘normal’ functional testing, you can reverse engineer how a web site or application works by a combination of trying different inputs and examining exposed parts of the system (JavaScript/HTML/Cookies etc). Security related issues in some respects are easier to find, as you are not constrained by ‘typical’ system usage. Those oft-ignored ‘edge cases’ are quite often useful attack vectors. But just like a functional problem, the context in which the bug exists is important - What is the cost to the company to fix/not-fix? What’s the risk of not

fixing? Are we a target for this sort of threat? Is this a compliance issue? Are we already being hacked in this way?

After examining how the Guardian's log-in page worked (in April), I found that the Guardian's 'id' system was vulnerable to a reflected cross-site scripting (XSS) attack. The web page could be 'polluted' with code or content that wasn't from the Guardian. In this case that was via the URL, I could include my own code and execute it when the user loaded the page in their browser.

The 'reflected' term used above means that its not the Guardian's website that contains the bad/polluting code. But rather their website just reflects the bad-code back to the user - when you request a web page in a certain way. Visiting the Guardian's website directly, by manually typing in the URL, would make us immune to this particular issue. But unfortunately, the Web is errh a web, and we click links all the time - especially on things like Facebook or twitter, where the links are often even obscured or shortened.

The bug could be exploited by amending a normal looking Guardian URL to include some extra/different data:

https://id.guardian.co.uk/signin?returnUrl=%27%0D%3C/SCRIPT%3E%3CSCRIPT%3Ealert%28%27HACKED%27%29;//%0D%3C/SCRIPT%3E

(The issue is fixed now, the above URL does not exploit anymore.)

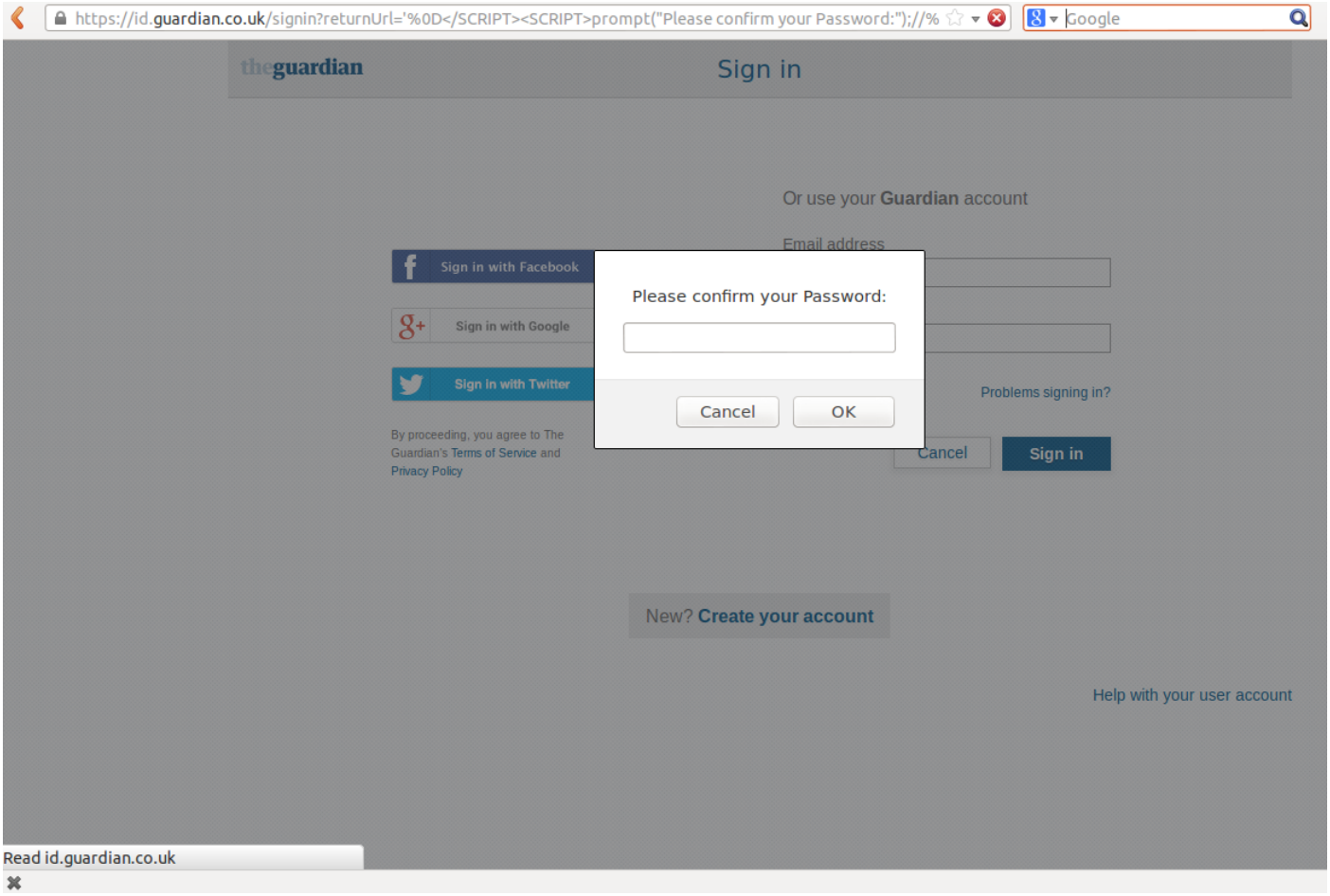
The web site would then incorporate that into its [returned] JavaScript code unchecked, instead of the normal un-tampered returnURL value:

…

```
**', '_parent');
                }
            });
        }
    }
}
```

…

My XSS code would execute on that page when opened via this modified URL. That modified code can be used to rewrite parts of the page, read a user's cookies or ask the user questions such as What is your password? E.g.:



The issue was particularly bad as it was on the log-in screen, a place where users would be expecting such a question. So despite being self-assured about the authenticity of the web page, thanks to it meeting the criteria mentioned above - A user could have been easily duped.

So what did I do?

I reported the issue to a contact at the Guardian and passed on the details of the bug. Following the conventions of Responsible Disclosure (http://en.wikipedia.org/wiki/Responsible_disclosure), I informed the Guardian of what I had found and that I might blog about the issue, after a given time period had expired. This gives the company time to fix the issue, and security researchers like me credit for our work.

What did they do?

They fixed the bug, thereby protecting their users. They also said thanks. That's a lot more than some companies do, so I'm happy.

What can you do?

As a tester, you can start looking for these issues yourself in your systems, there are plenty of resources available to help. For example OWASP have a testing cheat sheet for many application security problems, including reflected XSS (https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_%28OWASP-DV-001%29). Like other applications of exploratory testing, the real requirements are in your skills and mind-set and this comes in part from experience.

Your security testing skills may not let you know in advance if a system has been hacked when you come to read it, but at least you will have the skills to find out if it has been - or at least how easy it might be.