encription
IT Security and Forensic Services

An Overview of the 2013 OWASP Top 10

By Mike Sheward

Lists are all over the Internet. Just yesterday I found myself wasting 12 minutes of my life reviewing 'the 7 greatest Aston Villa kits of all time'. After a few moments debating the merits of each kit and whether it was worthy of its position on the last, I reminded myself it actually didn't matter – and then moved on.

One much less superfluous list you'll find on the Internet is the OWASP Top 10; a list compiled by (unsurprisingly) the Open Web Application Security Project (OWASP). Specifically, the data for the list is provided by seven companies that specialise in application security work, either as consultants or as producers of application security tools.

The data provided by these groups' accounts for about a half million vulnerabilities swept from many hundreds of applications.

If you aren't familiar with OWASP, their purpose is to encourage and promote better web application security practices. They do this by producing training, tools, conferences and articles that teach various audience members how to think more securely. The top 10 list is to OWASP as the iPhone is to Apple. A highly praised product that is the culmination of a lot of hard work.

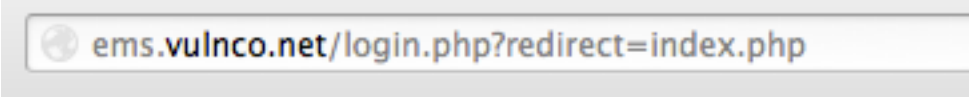
Indeed, the OWASP top ten has found its way into a number of security testing standards over the years – a sign of how much faith the industry has in its relevance and accuracy.

In this paper, we are going to run through the 2013 Top 10, and have a closer look at the vulnerabilities OWASP considers the most prevalent security issues in the world of web application security, starting from the bottom of the list and working our way up to number one – Radio One chart show style.

In at number 10 – the Phisherman's friend, 'Unvalidated Redirects and Forwards'

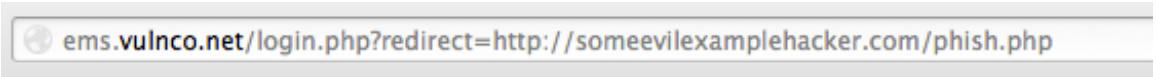
What is it?

In this example application, we note the following URL in the browser bar.



The purpose of this URL is to redirect a user to index.php after they have logged in. Unfortunately in this example the 'redirect' parameter is not subjected to

appropriate validation, and we can craft a URL that will redirect our user to a totally different server.



What happens if my application is vulnerable?

If your application features URL's that dynamically redirect users to other pages within the same application, or to other URL's outside of your application, you could be at risk from an unvalidated redirect.

This means that an attacker could send what appears to be a legitimate URL for your application, to redirect the user to a phishing or other malicious URL.

How do I fix it?

Your application should use a whitelist to ensure that the URL's it is redirecting to are validated before completing the request.

If you application is permitted to redirect to another application by design, this should still be whitelisted, and a message indicating to the user should confirm to them that they are being redirected away from your site. You see this approach often on news websites, which link to content from other pages.

At 9, 'Using Components with Known Vulnerabilities'

What is it?

The name gives it all away really! In summary, unless you are building everything, including your operating system and the programming language you use to put your application together from scratch – there is a chance that your application could be exploited via a vulnerability in one of these components.

What happens if my application is vulnerable?

Of all the items on the OWASP Top 10, this is probably the one that every single application ever made could be vulnerable to.

With that in mind, there is a degree of risk acceptance here. Occasionally, a car will break down because of a faulty component, leaving the driver stranded on the side of the M4. However the driver chose to accept the risk that such an event may occur, because the likelihood of it not occurring was good enough to press on..

How do I fix it?

That is not to say that we should just accept the fact that third party components may contain vulnerabilities, and we should just lay back and take it – not at all. There is still a fair bit we can do to help ourselves.

Keeping track of the versions of the components you use, and how they are configured is a great idea. Applying security patches as soon as they are released is another. Add running security scans to look for misconfigured or outdated components and you should feel a lot more comfortable.

Web application firewalls and IDS systems can also lend a hand here, by using custom rules or an updated signature to block attacks against specific components, if you know what such an attack might look like. I do have to remind you though, that the root cause of an issue should always be addressed – a WAF rule is always a temporary fix.

Number 8, 'Cross Site Request Forgery'

What is it?

One of my favourites, this attack exploits the trust a browser has for a user. A crafty attacker will construct a state-changing request in their own code and trick the victim browser to issue that request to the targeted application. If the user is already logged into the target application, the session cookie information will be provided along with the request, and the app will have no choice but to believe that the request was legit, as it came from the victim user.

You can see this in action in one of our Deciphering Digital Security Video's, here:

http://www.youtube.com/watch?v=oSvl1cdF4UM

What happens if my application is vulnerable?

Your users could easily be duped into performing actions that they aren't even aware they are performing – and if this happens it won't be the users fault.

How do I fix it?

Include a random token in the body of every 'sensitive' request that a user can make, or even go so far as to require re-authentication in the most sensitive of request types (such as money transfer).

Lucky for some, especially if you are an attacker, at 7, 'Missing Function Level Access Control'

What is it?

This vulnerability can be explained simply as giving someone more access than they are supposed to have. The actual way that this vector may present itself is very dependent upon the specific application or environment.

It could be than an extra button pops up on a UI, or it could be that an attacker is able to brute force their way to an admin page, simply by browsing to /admin in the application.

Another classic example is an exposed 'test' or 'development' page that is left behind.

What happens if my application is vulnerable?

This depends on the sensitivity of the function and your application. Obviously if a regular user can perform administrative functions on a payroll application, some folks might find there bonuses being reduced this year – while others might have an unexpected windfall.

How do I fix it?

Make sure that an application is written in such a way that new roles and permissions can be added in a centralized way. You don't want to be adding bits of code in different places each time. Avoid one offs.

Make sure that authentication checks are performed on the server side, and do not just rely on information provided by the user.

At 6, 'Sensitive Data Exposure'

What is it?

Another case of the name saying it better than I ever could. This vulnerability is all about the risk of sensitive information handled by your application being exposed to folks who aren't supposed to see it.

What happens if my application is vulnerable?

We've all seen the stories about organisations that have had massive data breaches. Healthcare records, credit card numbers or other personal information – if any of these go walkabout you are likely going to have a bad few days, and then months, and then possibly never recover.

How do I fix it?

This is a question that warrants quite a long answer, so lets summarise as best we can.

Determine which data handled by your application is sensitive, and then make sure it is encrypted at rest and in transit.

If you are storing sensitive information, and you don't really need to, you have to question why you are storing that information in the first place.

Ensure that all data is encrypted using appropriately strong algorithms, and that any keys used to encrypt data are managed properly.

In at 5, 'Security Misconfiguration'

What is it?

I've always considered this to be a sort of 'catch-all' entry on the top ten lists.

Basically this entry covers any number of issues that may come up if an organisation is not following security best practices.

This includes, but is not limited to: missing software patches, an increased attack surface because unnecessary ports are open or features are enabled, default user accounts and/or passwords, verbose error messages or misconfigured libraries.

What happens if my application is vulnerable?

Depending on the severity of the issue, this could include full-blown exploitation of the one or more servers hosting the application.

How do I fix it?

Repeatable hardening procedures and proper testing before deployment. This includes the use of a vulnerability assessment tool to pick up many of the issues that we've discussed in this list entry.

At 4, 'Insecure Direct Object References'

What is it?

A direct object reference is a URL parameter that directly references another system object.

For example:

http://www.example.com/customerfiles/customer23212.pdf

If this object should only be accessible to one user of your application, and access controls are not properly enforced another user might be able to brute force their way to it.

What happens if my application is vulnerable?

You could find that your application exposes sensitive information about one user to another, and that does not go down well.

How do I fix it?

Use an indirect reference, which is tied to a particular user or session. If you do this, an attacker will no longer have the ability to directly target resources. An indirect reference might look like this:

http://www.example.com/customerfiles/343301231

Combine this with an access check for any request from an untrusted source and things will get a lot more secure.

Number 3, 'Cross Site Scripting'

What is it?

Cross-site scripting is one of the most prevalent issues on the web, and the potential impact of such a finding is often ignored or downplayed.

A cross-site scripting flaw occurs when untrusted user input is returned to a page without be validated first. This allows an attacker to trigger a variety of behaviours

in the victim's browser, as the browser treats the malicious input as part of the application.

You can see an in depth look at Cross Site Scripting in our Deciphering Digital Security video here:

http://www.youtube.com/watch?v=gg4ICohrRXk

What happens if my application is vulnerable?

You users could find their session cookies being stolen by an attacker, which in turn equals a compromised session.

How do I fix it?

Validate all untrusted user input on the way in to your application, with proper filters and/or whitelists. Do the same on the way back out.

Number 2, 'Broken Authentication and Session Management'

What is it?

This category of vulnerability includes a variety of findings. It could include an inappropriate use of timeouts on session ID's. Meaning that a session ID never completely times out, so if it is obtained or brute forced it can be used to get a valid session.

It could also include the compromise of the password database, and the exposure of any unencrypted passwords.

Another common example of a problem that falls into this category is a bad 'forgotten username and password' reset logic, which may allow an attacker to redirect a reset link.

What happens if my application is vulnerable?

An attacker would be able to obtain access to your application as another user, which is something we generally try to avoid.

How do I fix it?

Make sure that sessions always time out after a predetermined period. Avoid exposing session ID's as part of the URL, and ensure that they are transmitted securely.

If cookies are used to store session cookies, use the HTTPonly and Secure flags to ensure that cookies cannot be submitted to another domain via a cross-site scripting attack.
As we touched on earlier with item 6, sensitive data exposure – make sure that passwords are stored using an appropriate hash and salting function.

And a number 1…(drum roll), 'Injection'

What is it?

An injection attack occurs when an attacker is able to 'inject' untrusted data into some kind of interpreter; this data is entered in such a way that it alters the execution flow of the interpreter.

The most prevalent injection flaw in the world of web applications is SQL injection, which occurs when untrusted input is thrown directly into a dynamic SQL query, altering the query to access or modify other parts of the back end database.

Injection flaws do not just impact SQL statements. Other types of injection include LDAP and command injection.

What happens if my application is vulnerable?

For SQL injection, the exposure could allow an attacker to dump the entire contents of a database, damage the integrity of data or even pop a system shell.

For command injection, the outcome will be the same, but faster.

How do I fix it?

Do not take unsanitised user input and drop it directly into any type of dynamic query.

Use a whitelist if possible to restrict the types of characters that can be included in user input. For example, if you know that you only expect numbers in an ID value supplied by a user, check to make sure that only numbers are present in the value passed to you by the client.

Never ever provide a mechanism for a user to drop commands directly onto a system shell, do this and it'll be a matter of minutes before bad things happen.

Conclusion

We've just completed our run down of the 2013 OWASP Top 10, and with that we've covered the most prevalent web application vulnerabilities in our world today.

Not all vulnerabilities can easily be dropped into one of these ten buckets. Often times a vulnerability will span multiple categories, and more complex vulnerabilities may be hidden behind the 'low-hanging fruit'.

You may have noticed that I didn't discuss which tools to use to look for a particular vulnerability. This is partly because the name of the best tool to use in a given situation will change over time, but also because reliance on tools alone to find vulnerability is bad practice.

The best approach is, and always will be to bake security into the application from the start.

Validate your security controls with different layers of human review, both internal and external, and then mix in automated tools to help expedite the process.

So, as OWASP themselves say – don't stop at 10. Root out all of the vulnerabilities that apply to your organization, and build a security culture that cascades down throughout everything you do. This is the best security tool you will ever have.

References

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Prepared by Encription Limited
www.encription.co.uk

+44 (0)330 100 2345

Email: enquries@encription.co.uk
Twitter: @encriptionit