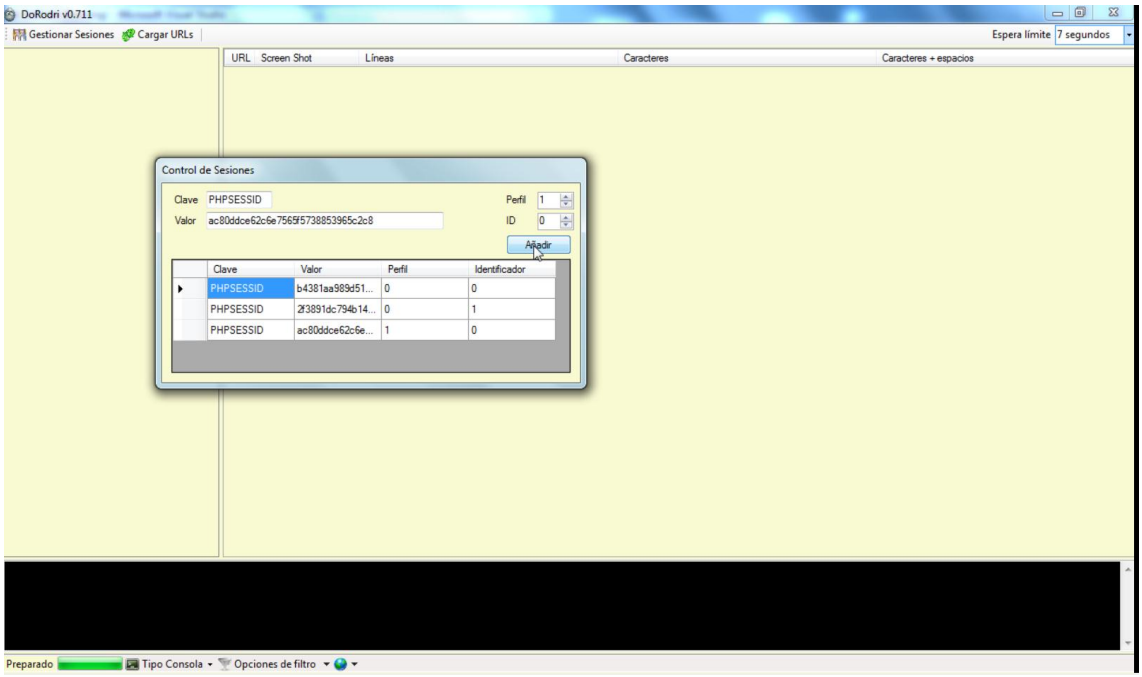
Failure to restrict access detection tool

May, 2012

Fernando Andina Meira

José Miguel Soriano de la Cámara



Intro

The vulnerability known as “Failure to restrict access” has been escalating positions among the most common vulnerabilities in web applications. Poor training in information security and development cycles often too fast and hardly planned help this problem so spread, together with wrong configurations and unprofessional maintenance.

One of the reasons that make really difficult to control this issue is the lack of tools that help to have early detections. In this project we deal with the question of how to focus this vulnerability and we offer a solution that will help security analysts to early detect this bug in their applications. Instead of an approach centered in a detection algorithm, which is unfeasible, we have opted for an analyzing tool, which gathers data relevant for the analyst to quickly decide whether there is or not a bug in the web application.

How to deal with this bug

Even though the vulnerability is conceptually easy to understand and the reasons why it can appear in web applications are also very straight-forward to see, when you have to deal with an application design than has to answer to this: “We need an application that is able to get N webs pages of N different kinds with N different sessions and automatically decide if one of them is suspicious of having a failure to restrict access tool bug, taking into account that there are N different ways to restrict someone´s access to a web resource” is when you start seeing the unclear side of this bug.

As we all know, the internet is a varied place, and you can find as many kinds of web applications and you can find developers with different opinions. If you want to develop and detection tool that is able to handle such a scenario, you need either a super-complicated algorithm able to deal with every possible situation or a super-complex configuration area able to define every possible kind of web application. Oh, and don´t forget to update your algorithm/config area every time a web developer releases a new AJAX or a new way to restriction warning panel.

A purely automated approach is just not an option, as far as we can recommend.

What to do then? Help the security analyst

What we are trying to say here is that whatever you try to do, at the end it has to be a human who actually validates if a web page is what it is supposed to be with a certain active session or we have a restriction access failure bug.

If it doesn’t not matter what we do to automatically detect a bug because a human has to review it, then, why we don´t change to approach to help the analyst?

The analyst knows pretty well what he or she (normally him) is expecting from a web page, he knows if he is dealing with a CMS, a forum, a shop, an ERP, or whatever you want. He knows is restriction is made of 401 or warnings or if it is handled as a redirection. He knows what is supposed to be the difference between a session of an admin, and editor, a registered user or an unauthenticated guest. He is the best tool to detect a failure to restrict access bug.

What the analyst doesn’t have is time. Time to manually go through all the links of a web application with all the different sessions and spending time in all them reviewing is everything is ok. So let´s help the analysts by giving them a solution that can help them to validate better and faster all the pages with all the session.

Our solution

What we are going to describe here is a process that we find useful to design an application that helps to deal with these bug. Many small details are left in favor of the clarity of the important points. We´ll provide an implementation to help you to see how it works in real time with a UI. Following are the steps that the application has to manage to be a useful application for the analyst.

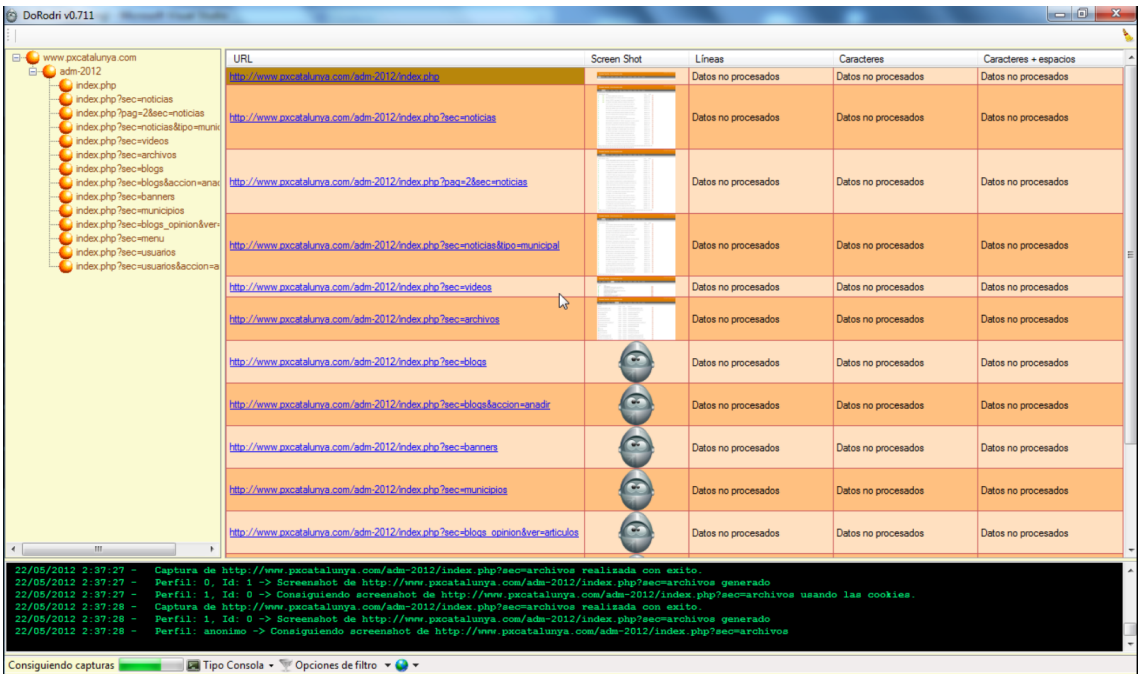
We have to notice here that it is completely in the field of the personal opinion what would the best usable user interface to show all this, so we don´t describe it here, because we think there should be solutions much better than ours.

Step 1: get all the URLs and cookies that you need.

This is always a white-box analysis. You need to be given all the URLs and sessions related to the application to be able to perform a realistic analysis with actual results.

The URLs tree can be gathered using spiders like OWASP ZAP or BURP, or probably even better manually (those links in Javascripts are always tricky), the cookies can be take using the same tools or other like Firebug.

When you have this entire ready, URLs and sessions IDs, we can proceed to the second step.



Step 2: crawl everything with every session

- Go to a URL with the cookie with the highest privileges. This one will be the reference to compare with the others.
- Download the page and take a screenshot of it
- Go the same page with the next cookie in privileges and repeat the process.
- When you don´t have any more cookies, download the page without any session at all.
- Repeat all this for every URL you need to crawl.

Step 3: Analyze data and compare

Now we have all the data as it is, a lot of HTML code, it is time to see what we can know from it.

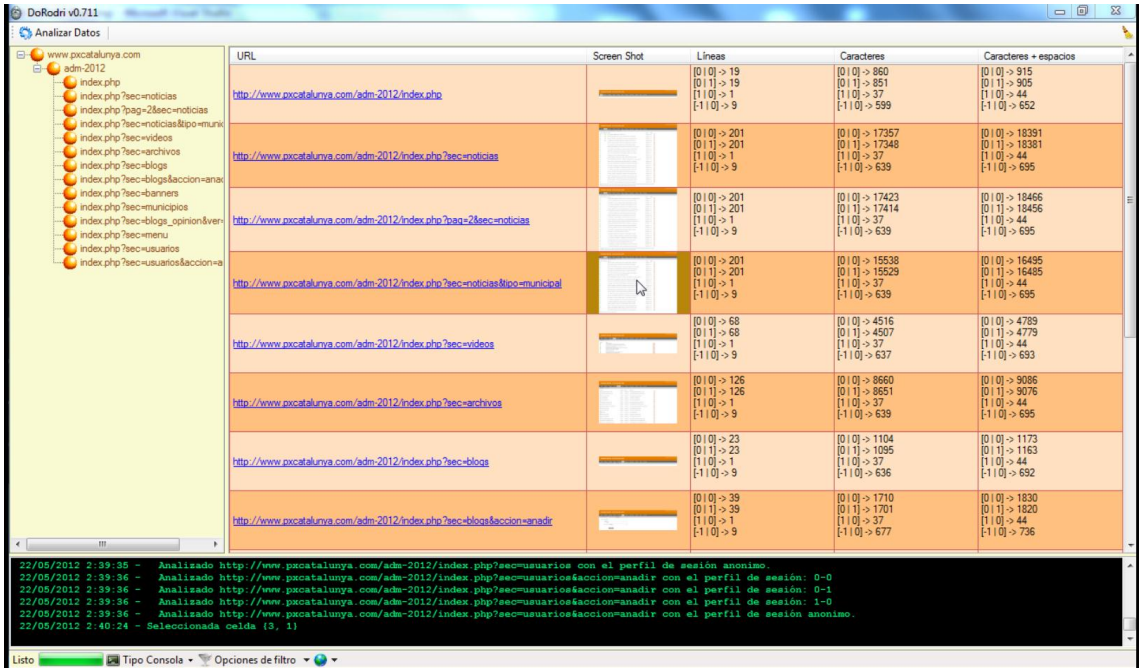
When reviewing manually, first thing you will do is to visually compare the same web with two different sessions. Let the screenshots to be quickly comparable.

After the visual checking, when two pages are expected to be very similar but not exactly the same, it is useful to have some statistics about them. We think these include: characters, characters including spaces and the number of lines. These give a very well idea about how different the code of the same page loaded with two different sessions is. But you can add as many as you want: words counter, HTML tags counters, keywords, etc.

Another level we can add is the instead of just the difference measured in quantities (how much is the difference in total weight), but in *how much different it is one from another*. “Access granted” is only 1 character bigger than “Access denied”, but they are actually more different than that.

For this scenario we recommend the implementation of mathematical algorithms such a Damerau-Levensthein. The Damerau–Levenshtein distance is a "distance" (string metric) between two strings, i.e., finite sequence of symbols, given by counting the minimum number of operations needed to transform one string into the other, where an operation is defined as an insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters.

It is not efficient enough to compare to complete pieces of HTML code, but it can help you comparing line by line and gathering the biggest different and to calculate the % of difference between the code of a page with two different sessions.



Conclusions

This 3-steps system has some clear benefits:

- Little configuration: you need the URLs and the cookies (sessions). That´s it. And that´s all you will ever need for any kind of web page.
- The hard, tedious work is completely automatic: all the crawling process shouldn´t require any intervention from the analyst.
- All the info you need is just there: Is there any special metric you´d like to analyze? Add it. All the specialization is kept in one place.

With this approach what do you provide is the good framework that the analyst needs to quickly know whether a page has a potential issue with failure to restrict access vulnerability. You save him time, which the main reason we have this bug: lack of time to properly code, lack of time to properly test.

A good challenge that remains unsolved is which would the best UI to support this system. We propose one in our tool, but we are sure there could a lot better one.

We hope you have learned something by reading our research, and specially that we can help the community to fight against the spread of this small, simple, but sometimes catastrophic and embarrassing bug.

We are open to any ideas and collaborations, let us know!

Download our tool and check more in our site dedicated to this project:

http://failuretorestrictaccess.wordpress.com

Fernando Andina
José Miguel Soriano.