OWASP WebGoat Lab (cont.)

Thomas Martin

March 21, 2019

Objectives

This week we will continue to work on the OWASP WebGoat web application, and explore the different attack techniques that can be used against it.

Teams

The problems have been collected into three sets. You will be divided into three teams, with each team working on a different set. After a team has completed their set, they will present to the rest of the class an explanation of the exact nature of the problem, and some general information on how to go about finding the solution (without giving the answer away).

1. Access Control Flaws and Authentication Flaws
2. Code Quality, Concurrency, Improper Error Handling, and Session Management Flaws
3. Cross-Site Scripting (XSS)

Note that there is another set of challenges that we will explore later in the unit when we cover database security.

1 Set 1

1.1 Access Control Flaws

The first challenge is simply a matter of trying all possibilities and seeing if any do not match up with the how the policy should be enforced.

In the second, you have an interface that allows you to select any file you would wish to view. The challenge is to view a different file in a different directory. This can be done by modifying requests in Zap as they are sent.

This challenge relates to the file system. If you open another terminal and run the `ls` command in your WebGoat folder, you will probably only see two files

(the original jar file and a newer `UserDatabase.mv.db` file. However, if you run `ls -al` you will see an additional folder. It is within this hidden folder that the files for the WebGoat application are stored. The code for the current page takes the option selected by the user and uses it to read and display a file back to them. Your goal is to force the page to display the file `WEB-INF/spring-security.xml` instead. The following command will be helpful in determining the working directory and the location of the target file:

```
find .  | grep <filename>
```

In Lab Stage 1, you have considerable access to several accounts. Use your access to regular users and admin accounts to understand how the HR system works (i.e. what values are sent as parameters for the various requests). Then log in as Larry and try to modify an action so that it does what an admin would have done if they had tried to delete Tom's profile.

Lab Stage 2 and Stage 4 are only possible with the Developer version of Webgoat, which we are not using, and so can be ignored.

Lab Stage 3 is very similar to Lab Stage 1 and should not pose any problem.

1.2 Authentication Flaws

The "Password Strength" check is a good exercise in seeing the difference in expected times to brute-force passwords. However, the formula used in the linked site has changed (to reflect access to faster processors), so the results you get will not match the values WebGoat is expecting. Perform the task, and click "Show Solution" to see the expected answer (for this challenge only).

The "Forgot Password" challenge requires you to find the password for the `admin` account. Since it is only asking for the favourite colour, and there are not that many common colours, it should not take long.

The "Multi Level Login" challenges use one-time passwords (TAN) with each login. It is not implemented very well. The first challenge is to login as Jane with a stale TAN (one that has been used already). The second is to use Joe's credentials and TAN to log in as Jane.

2 Set 2

2.1 Code Quality - Discover Clues in the HTML

When you right-click on an area of a webpage, one of the options is "Inspect Element". This is a simple way to immediately access the source code of a particular area of that page. It saves having to scroll through the entire source to find the part you are interested in. This is the only tool you should need for this challenge.

2.2 Concurrency

The description of the "Thread Safety Problems" says that this challenge requires two browsers. Kali only has Firefox installed, but fortunately, this task

can be completed with two Firefox windows. There is no need to install a second browser. Try to simulate two users trying to access their own details at the same time. Once you get the desired unusual behaviour, look through the source code and try to work out why it happens the way it does.

The second "Shopping Cart Concurrency Flaw" challenge also requires the use of a second browser window, but does not require as much fast-paced manipulation. Try different orders of actions in the different windows to get the desired outcome.

2.3 Improper Error Handling

This task requires tampering with parameters in transit, using OWASP ZAP or similar. Viewing the source code may help with figuring out how to succeed. There is some odd logic handling the authentication.

2.4 Session Management Flaws - Spoof and Authentication Cookie

You have been given two pairs of credentials for this page: `webgoat:webgoat` and `aspect:aspect`. The name of this task suggests you try to figure out how the site is deriving the cookie value. If you can work that out, and then replicate a valid cookie for user `alice`, you can spoof your way into the site.

2.5 Session Management Flaws - Hijack a Session

Note: Completing this challenge requires a tool that is not included in Kali and is no longer available. It is still worth going through it as there are some interesting concepts explored in the first half.

Also, this challenge uses Webscarab. This is another proxy, very similar to ZAP. It listens on port 8008 by default, so after running it ("Applications" - "03 - Web Application Analysis" - "webscarab") you just need to change the Network settings in Firefox to use the proxy on port 8008. Afterwards, you can change the port back to 8081 to use ZAP again.

The challenge focuses on the WEAKID Cookie value. If you try any username/password pair, the server will set a cookie value for the WEAKID paramater that is submitted on all subsequent queries. If you delete that value from a request, your browser will be given a new value from the server. You can do this in Webscarab by going to the "Proxy" tab and clicking the checkbox "Intercept requests". Delete both the WEAKID cookie value and POST parameter, then click "Accept change". In the next window that pops up, uncheck "Intercept request" and click "Cancel ALL intercepts". You should notice that the WEAKID has changed.

Two values are not enough to work out what is going on. And generating them this way is too time consuming. Click on "Show Solution" and follow the instructions for how to automate Webscarab generating 50 values.

2.6 Session Management Flaws - Session Fixation

This task has four separate stages. The first simulates a phishing email where you need to set the parameter `SID` in the URL. Stages 2 and 3 simulate the victim clicking on the link and logging in. Finally, as the attacker, you use your knowledge of the value of SID (that you specified in Stage 1, to log in as the victim.

3 Set 3: Cross-Site Scripting (XSS)

Note: If you are unfamiliar or out of practice with writing HTML and Javascript, you may struggle with some of this section. If necessary, review the tutorials at `http://www.w3schools.com/html/default.asp` and `http://www.w3schools.com/js/default.asp`.

3.1 Phishing with XSS

This page presents a search dialog. As is standard, in the results page it reflects back what you entered. The problem is that it makes no effort to sanitize what you entered, meaning you can include any HTML markup, including scripts. The aim of the exercise is to enter input into the form that will result in a believable page requesting username and password, that when entered will be sent to the URL `http://localhost:8080/WebGoat/capture/PROPERTY=yes&ADD_CREDENTIALS_HERE`.

Note: This task is somewhat vague in what is being asked, and incomplete. The final result is not actually a feasible attack as the dangerous page is only reflected back to the attacker. Later tasks give examples where the dangerous content could be directed to other users (necessary for XSS). Consequently, I would not spend too much time on this task, but at the least look through the details provided in the hints and see exactly what they are doing.

3.2 Stored XSS Attacks

Another example with user input accepted with no filtering. All you have to do is create a message that when viewed, executes a single javascript command: `alert("Insert your message here!");`

3.3 Lab: Cross Site Scripting

Half of the six stages require the developer version, and most of the rest will not make much sense without them. Have a read through and attempt at least the first one.

3.4 Reflected XSS Attacks

This task is incomplete. The description mentions crafting a URL that could be sent to the victim, but the intended task (going by the hints and provided solution) only deal with inserting HTML into the page and getting executed in the attacker's browser.

3.5 Cross Site Request Forgery (CSRF)

In an actual CSRF attack there would be two separate sites. The first would be used by the attacker to expose their maliciously crafted HTML to innocent victims (e.g. any site with comments). The second would be the service where users store something of value the attacker wants to steal (e.g. online banking). This challenge is to embed the request to the second site in an image tag in a comment on the first.

However, working in WebGoat means the example is limited to a single domain (everything is on localhost). Furthermore, because of how WebGoat works, the URLs for the different pages are odd looking. They will appear as either:

`#attack/??????????/???`

(each ? representing a single digit) or

`attack?Screen=??????????&menu=???`

For the purposes of this task, the target URL is

`attack?Screen=??????????&menu=???&transferFunds=5000`

where the numbers match the Screen and menu values for this challenge. These are always displayed on the right column of each page.

3.6 CSRF Prompt By-Pass

This task introduces a protection mechanism that would prevent the previous attack. If you navigate to:

`localhost:8080/WebGoat/attack?Screen=?????????&menu=???&transferFunds=5000`

you will see that the bank site expects the user to click "Confirm" before the transaction is authorized. The previous attack would just load this request. Some carefully crafted HTML and Javascript needs to be written that will load this page and then submit the confirmation.

This task requires in-depth knowledge of HTML and Javascript. I recommend spending only a few minutes on it before examining the hints and/or solution.

3.7 CSRF Token By-Pass

Another security mechanism is introduced to prevent the previous attack. Now a token is provided with each initial request and must be submitted to confirm the transaction.

As before, do not spend too long on this one.

3.8 HTTPOnly Test

No explanation needed for this. Read the instructions and follow them.

4 Extension Task

If you have completed the above challenges, start to work on the others (without looking at the provided solutions, and only using the hints if strictly necessary).

5 Summary

These exercises demonstrate many of the OWASP Top 10 Risks in Web Applications. As we have seen, some are relatively simple to exploit and some are very subtle. It is very important to understand these vulnerabilities as the use of web technologies continues to grow.