OWASP WebGoat Lab - Setup

Thomas Martin

March 16, 2018

1 Objectives

Due to the prevalence of websites, they make an attractive attack vector. OWASP have produced an extensive collection of example sites that are vulnerable to specific attacks. During this and the next session, we will explore those weaknesses and how they can be exploited.

Note that familiarity with HTML and Javascript will help with many of these problems. You can brush up on your skills with any of the many tutorials online, such as `http://www.w3schools.com/html/default.asp` and `http://www.w3schools.com/js/default.asp`.

2 Configure WebGoat

WebGoat is contained in a single file and needs Java to run. While it can be run anywhere, the tools needed to complete the exercises are already present in Kali so it makes sense to run it in a Kali VM. If you are running the full Kali VM setup in previous labs, Java is already installed. If you are running Kali Light or an older release of Kali[1], follow the instructions in the Appendix.

Note: If you are running WebGoat from a Live CD version of Kali, all changes will be erased on reboot. Installation of WebGoat will have to be repeated, and there will be no record of which lessons you have completed. It is recommended you take appropriate notes of what you did and how. You should also make sure you are running Kali with 2GB of memory when running WebGoat.

2.1 Install WebGoat

Download WebGoat 7.1 from Moodle or from the following site[2]:

`https://github.com/WebGoat/WebGoat/releases/download/7.1/webgoat-container-7.1-exec.jar`

Create a dedicated directory and move the downloaded jar file into it. Run the command from within that directory:

[1] You can check if Java is installed by running the command `java -version` in a terminal. Webgoat 7.1 does not seem to work with Java 1.6.0.

[2] Your VM will need to be in NAT Network mode so that it will have Internet access.

```
java -jar webgoat-container-7.1-exec.jar
```

There will be a long scroll of text, but providing there are no problems, eventually you will get a message saying:

```
...  INFO - Browse to http://localhost:8080/WebGoat and happy hacking!
...  org.apache.coyote.http11.Http11Protocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
```

Visit the link and use the provided username and password to log in. WebGoat does keep track of which challenges you complete in a given session. This is displayed with a green tick-mark. It is not instant, sometimes you have to refresh the page before it appears. And obviously this will not be saved on reboot if you are using a Live CD (the entire installation described above will need to be repeated).

2.2 Configure Proxy

At this point you can browse the WebGoat pages to get a feel of the lessons. Each lesson generally has:

- A lesson Plan
- Some hints (attempt the lesson without looking at these initially, but if you are not making progress either ask for help from the instructor in person or consult some of the hints)
- Source code (an attacker generally would not have this access, but it makes understanding the attacks easier)
- The solution (viewing the solution instead of attempting to solve the challenge will not be instructive).

Many of the lessons require the ability to view parameters going between the browser and the webserver, as well as sometimes modifying them. There are many tools in Kali that can do this, but the recommended one is `owasp-zap`. It can be started from "Applications" - "03 - Web Application Analysis" - "owasp-zap". When you run it, you will get an error message. Zap generally listens on port 8080 to proxy connections to the webserver (usually port 80). However, WebGoat is already listening on port 8080. So the first thing to do is go to "Tools" - "Options" - "Local Proxy" and change the port number from 8080 to 8081. In the browser (assumed Firefox), right-click above the URL bar and enable the Menu Bar. Then go to "Edit" - "Preferences" - "Advanced" - "Network" - "Settings". Select "Manual proxy configuration", set "HTTP Proxy" to localhost, port to 8081, and erase "localhost, 127.0.0.1" from "No Proxy for".

When you start using Zap in WebGoat, you may be surprised at how many requests there are, which makes it difficult to access the ones you need. Most form submissions are done with `POST` rather than `GET`. These POST requests are generally the important ones we will be interested, and are relatively easy to pick

out from the list of requests. There are other patterns that will be easier to spot with practice (e.g. URLs with `WebGoat/attack` instead of `WebGoat/service`).

3 Introductory OWASP Lessons

The lessons and challenges in OWASP vary in their difficulty, from the simple to those that are sufficiently complex that they are beyond the scope of this unit. And they are arranged in alphabetic order (as they appear in the left-side menu bar), not in order of complexity. We will start with some of the easier challenges to get to grips with how they work and the general approach, and then move on to the more tricky challenges.

3.1 Insecure Communication (Insecure Login)

Ideally, this should be done with the server running on a separate machine. Sniffing the password would then require Wireshark. However, since we are limited to a single machine, OWASP ZAP can be used to determine what was sent. ZAP can log a lot of traffic, especially if it has been running for a long time. It sometimes helps to delete all records or restart ZAP[3].

Go to the Insecure Login page, submit the pre-filled form, and try to find the corresponding POST request in ZAP. Once you have found it, select the "Request" tab, and you should be able to see the value of the submitted password. If you enter the correct password, you will have completed Stage 1 of this task.

Stage 2 of this task requires a `HTTPS` server, which we have not configured so we will ignore this and move on.

3.2 Insecure Storage (Encoding Basics)

This section is not a challenge, but a demonstration of some of the different encoding schemes.

3.3 Parameter Tampering - Bypass HTML Field Restrictions

The HTML in this page is written to restrict the possible user input. You must find another way to submit the form such that all six fields have invalid values. Simply browsing the options presented on the page limits what values can be picked (e.g. the drop down menu can be either "foo" or "bar"). In the above "Insecure Login" exercise, we were able to see the values sent to the server. In this exercise, we need to change them and submit six invalid values.

Switch to the ZAP window, and press the green button in the toolbar so that it turns red. When you go back to the browser and submit the form, ZAP

[3]Deleting many records has caused ZAP to crash, so you may wish to use the safer option of simply restarting ZAP

will pop up and allow you to edit the request before it gets sent to the server. When you are done, press the red button to switch it back to green, and press the "play" button (request is send and all further requests are processed without interruption).

3.4 Parameter Tampering - XML External Entity (XXE)

This task requires in-depth knowledge of XML. I would consider this beyond the scope of this course. Feel free to attempt it, but do not spend too long. It is worth having a look at the solution at any rate.

3.5 Parameter Tampering - Exploit Hidden Fields

Take a look at what happens during normal use of the site, and then try to subvert it.

3.6 Parameter Tampering - Exploit Unchecked Email

This example is not very realistic. The purpose of the page is to allow you to send mail to the website admin using either your own Gmail login, or the provided credentials. It is very unlikely that any site would actually do this. However, the two tasks are very straight forward. First, send a message with some malicious script. A useful line of Javascript for this purpose is:

```
<script>alert('Put some text here!');</script>
```

Second, find a way to change the destination of the email to one you pick.

3.7 Parameter Tampering - Bypass Client Side JavaScript Validation

This is similar to "Bypass HTML Field Restrictions", but the forms are restricted using JavaScript instead of HTML. Can you submit the form such that no violations are detected by the JavaScript but all 7 violations are detected by the server? Alternatively, disabling the JavaScript in the browser would achieve the same ends.

4 Summary

These exercises demonstrate some of the OWASP Top 10 Risks in Web Applications. As we will continue to see over the coming sessions, 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.

In the next section, we will start to look at some of the more challenging problems in WebGoat. You will be split into teams to work on different problem sets.

Appendix A: Install Java

Download the latest version of Java:

`https://www.java.com/en/download/linux_manual.jsp`

(You do not want the RPM, and only use the 64-bit version if you are sure your Kali VM is 64-bit. You should get a `.tar.gz` file. Run this command while you are in the Download folder:

```
tar zxvf jre-8u121-linux-i586.tar.gz
```

(Note that the file name will likely be different and you will need to replace it with the name of the file you downloaded.) Move the decompressed folder to `/opt`:

```
mv jre1.8.0_121 /opt
```

To be able to run Java from anywhere, issue the following commands:

```
update-alternatives --install /usr/bin/java java /opt/jre1.8.0_121/bin/java 1
update-alternatives --set java /opt/jre1.8.0_121/bin/java
```

Check `java -version` to make sure it has installed correctly (it should be version 1.8.0_121).