TDDC90 - Web Security Lab

Version: 2020-11-17

1 Preliminaries

The goal of this lab is for you to try out some of the vulnerabilities that have been discussed during the lecture. You will be running a web application that has many vulnerabilities built into it, and it is your task to experiment with a few of these vulnerabilities. You should understand the vulnerabilities and the mitigations to the risks they pose. You will write a report where your responses should be concise and informative, and should reflect the fact that you have understood the risks and mitigations.

1.1 Setup

The lab environment consists of a virtual machine (VM) serving a deliberately vulnerable web app called "Damn Vulnerable Web App" (DVWA). The web app can be accessed with any web browser, using a uniquely designated port on `localhost`. (We use Firefox in the lab.) Since the vulnerable app is isolated inside the VM, the risk of accidentally damaging files in your home folder when attempting exploits is eliminated. Also, to make the lab a bit more realistic, you will not be able to access the VM directly. We use a VM based on User Mode Linux (UML). The instructions below describe how to set up the VM on your university account (either on a physical machine in the SU-rooms, or on the ThinLinc server).

1. Run the script `/courses/TDDC90/websec/scripts/setup_websec.sh`

   You will be asked to provide your Webreg group number. **It is very important that you provide the right number, as this is used to compute a unique port number for connecting to the VM.**

   The script will create a directory called `websec` in your home directory, containing a copy-on-write VM image and a number of shell scripts.

2. To start the lab, run the script `start_websec_group`*`NN`*`.sh` in your `websec` directory, where *`NN`* is your group number. You should see some scrolling text, indicating that the VM is starting up. This takes about 15 seconds.

3. After the VM has started, you can access the vulnerable web app using a browser. For convenience, we have supplied you with a script in the `websec` directory that launches Firefox with the correct URL, based on your group's unique port number. Simply run (in a different terminal) `launch_browser_group`*`NN`*`.sh`.

   Note: When running the lab via ThinLinc, it is possible to access and attack lab instances belonging to other groups by changing the port number in the URL. Doing so without the other group's explicit consent **would be considered abuse, and may lead to suspension from using LiU's computer systems**! We recommend that you always use the provided script to access the vulnerable web app, in order to avoid inadvertently using the wrong port number.

4. The vulnerable web app will greet you with a login screen. Log in using username: *admin* and password: *password*.

   The security level, which determines how hard it is to exploit the vulnerabilities, is set to "low" by default. You can change it by clicking on "DVWA Security" in the menu.

5. To shut down the VM, run the script `shutdown_websec.sh` in the `websec` directory. This will take a few seconds, and you should see some text in the console where the VM is running, indicating that the machine has shut down.

   Note: Always use the shutdown script to power off the VM. Don't just hit Ctrl+C or close the terminal, as this might cause file system corruption in the VM.

If at any time you want to reset the web app to its original state, click on "Setup / Reset DB" in the menu, or alternatively navigate directly to `http://localhost:`*`portnumber`*`/setup.php`, and click on "Create / Reset Database" at the bottom of the page.

Running the lab on your own computer

Note that we do not provide technical support for running the labs on your own computer. Also, remember that DVWA is designed to be highly vulnerable. If you install it on your own machine, *it is your responsibility to make sure that you don't expose your system to attacks from the internet!*

If you are using a recent 64-bit x86 Linux version, it is, in many cases, possible to run the lab by simply copying the lab files to your own machine:

1. First install the packages `uml-utilities` and `vde2` using your package manager.
2. Copy the `scripts` and `uml` directories from `/courses/TDDC90/websec/`, and place them in a local directory on your computer.
3. Edit `scripts/setup_websec.sh` and change the variable `WEBSEC_ROOT` to the local directory in the above step.
4. Likewise, change the variable `UML_BASE` in `scripts/start_websec.template` to the path of your local copy of the `uml` directory.
5. If you are using a web browser other than Firefox, you will also need to change `scripts/launch_browser.template` (or simply start the browser manually using the correct port number).
6. Run the setup script (`scripts/setup_websec.sh`) as described above.

If this approach does not work for you, the easiest route is to install DVWA in a VirtualBox or VMWare virtual machine. You can find some instructions for this on the DVWA GitHub page: `https://github.com/digininja/DVWA`

1.2 First task (detailed example)

This is your first task, which we will cover in detail so you understand what should be done for each vulnerability listed in Section 2. We will use *brute force* as an example. ***Note that this is an actual task and your report should contain the answers to the questions at step 6 and 7 below.***

1. Click on "Brute Force" in the menu.

2. For each attack there are links to web pages that explain the vulnerabilities in more detail, use these references if you need more help. (Even if you do not need more help, these references are mostly good).

3. Try several combinations of username/password in an attempt to brute force your way into a protected area.

4. Now click on "View Source"[1]. In this window you will see the part of the code that makes the web application vulnerable to this particular attack. What is shown is the code for the *low* security setting (if you have selected *low*).

5. Click on "Compare All Levels". The code for all four security settings is now shown.

6. For this task you should write in your lab report an explanation of the solution used in the *impossible* security level. Explain how and why it works. (It is not enough to just write the names of the functions called, you should explain what they do.)

   Note that the *impossible* security level also uses mitigations against other types of vulnerabilities, such as SQL injection and CSRF. You do not need to describe these protections. Instead, focus on how *brute force* attempts are mitigated.

7. The solution shown in the *impossible* security level consists of two complementary techniques. One of them is *not* recommended in production code, because it can introduce another vulnerability. Explain what the problem is with the solution.

   ***Hint**: Each request to the web page will spawn a new* `php` *process to execute the PHP code. Every operating system has a limit on how many processes that can be active at the same time. (For example, on Linux the limit is 32768 by default.)*

2 Tasks

Follow the general workflow from the brute force example given in Section 1.2, and for each attack write answers to the questions in your report. Remember, when asked to explain how the *impossible* security level code mitigates a vulnerability, it is not enough to just write the names of the functions used, you should also explain what the functions actually do and why it works.

- **Command Execution:** Which commands were you able to run, and what was the output. How is this attack mitigated in the *impossible* security setting?

- **File Inclusion:** How should you go about to see the contents of the file named `php.ini` that is in the root of the web application folder? What is the contents of the `/etc/passwd` file (just copy/paste a snippet, no need for the entire file)? How does the *impossible* security solution mitigate this vulnerability?

  ***Hint:** Note that the included text shows up in the uppermost part of the page, above the DVWA user interface.*

- **SQL Injection:** The passwords are hashed in the database, however since a weak hash function has been used they are still valuable for us. What should you enter in the field to stage a SQL injection that retrieves all the username and password combinations from the database? How is this vulnerability mitigated in the *impossible* security level code?

[1] Note that we have disabled the button "View Help" next to it.

- **XSS Stored:** What did you post to the comments board and what was the effect? What happens if you post `<SCRIPT>alert("XSS!")</SCRIPT>`? Give an example of how XSS can be exploited in a real-world setting. How does the *impossible* security level avoid this problem?