

***Final Project Programming Languages***

WordPress Security Scan (Brute Force and SQL Injection)

Alan Kuri García

A01204805

Assessor: Dr. Benjamín Valdés

*Problem:*

WordPress is a semantic personal publishing platform with a focus on aesthetics, web standards, and usability. Based on PHP and MySQL, released on May 27, 2003 by its founders Matt Mullenweg and Mike Little.

“*Since March 2016, WordPress is powering 26.4% of the Web, and it’s on the rise…Also interestingly on a daily basis there are over 500 sites being created on WordPress*” (Tomovic, 2016).

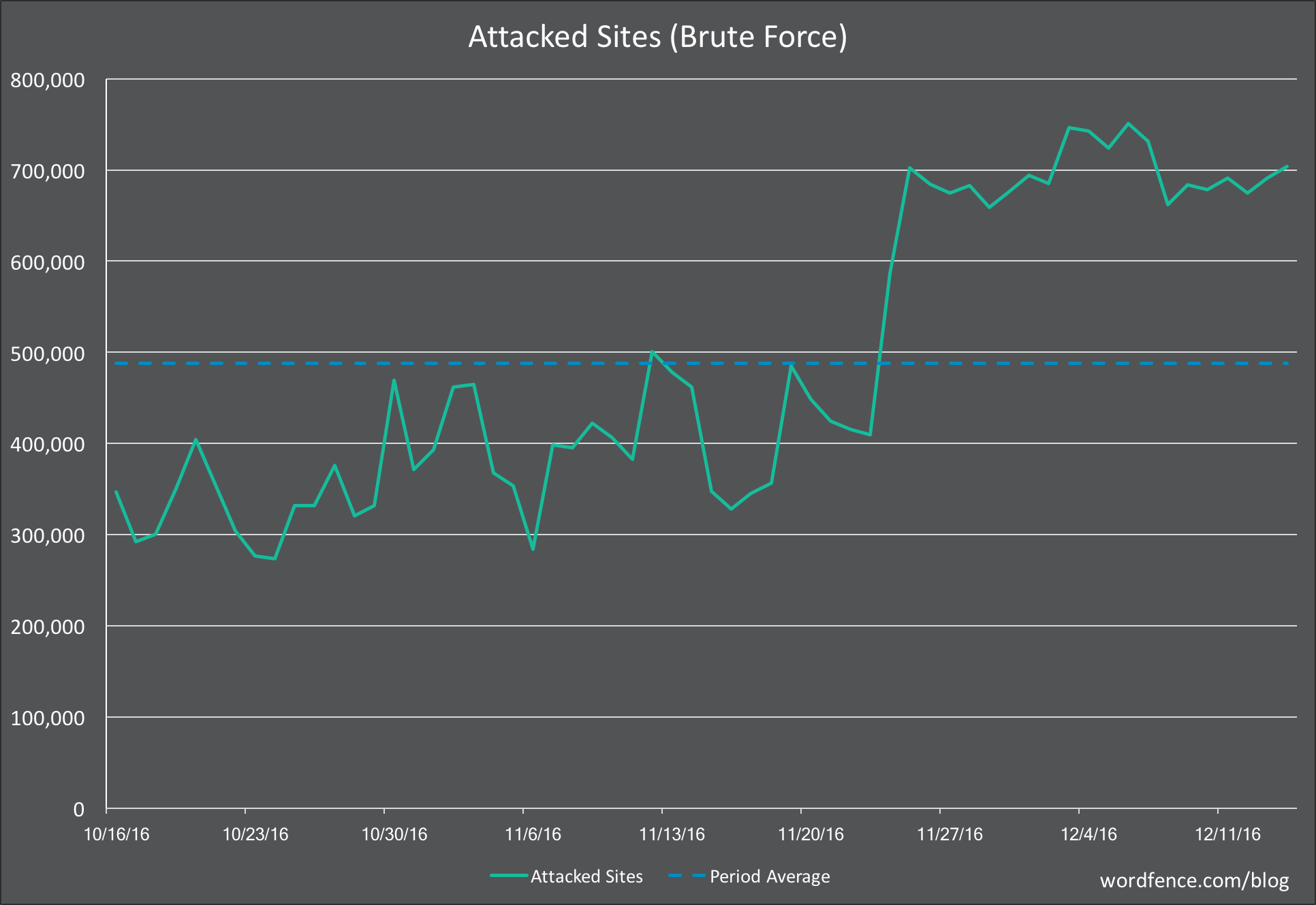
According to World Wide Web Technology Surveys since the March 1st, 2018 The most popular Content Management System (CMS) is WordPress with 30.5% of webpages in the web. (There are more than 73 million WordPress sites).

Various famous sites are powered by WordPress, some examples:

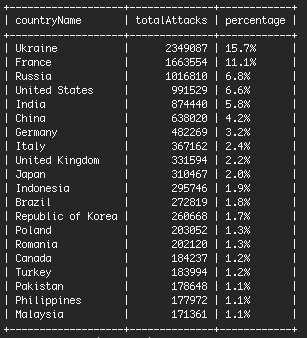
* The New York Times.
* Wall Street Journal’s Spe akeasy.
* People Magazine.
* National Geographic.

Because its popularity WordPress has been a natural target for security exploits. Since November 24th, 2016, WordPress Scan (WPScan) noticed a rise in brute force attacks to WordPress sites, as reminder a brute force attack is one that tries to guess your username and password to sign into your WordPress website. The following data will show how the attacks have increased during the next 3 weeks since that date, and where are they coming from.

The dotted blue line indicates the average number of attacked sites for the 60 days shown.



The following table shows the top 20 countries sorted by number of attacks only in a time lapse of 24 hours.



The question after seen this numbers, Is WordPress secure? The answer is “It depends”. WordPress itself is very secure as long as WordPress security best practices are followed. WordPress runs on open source code and according to WordPress.com - Employees has a team of 449 persons specifically devoted to find, identify and fix WordPress security issues. That’s why keeping WordPress updated to the latest version is essential to maintain the website secure.

It’s important to note that WordPress security vulnerabilities extend beyond WordPress core into the themes or plugins installed on a website. According to WPScan, of the 3,972 known WordPress security vulnerabilities:

* 52% are from WordPress plugins. (A plugin adds and extend functionality to a WordPress page, to this day there are 54,908 plugins).
* 37% are from core WordPress. (source code).
* 11% are from WordPress themes. (visual templates).

According to Kristen Wright the 4 must common vulnerabilities found in WordPress sites are:

* Brute Force Attacks.
  + As explained before in this paper, brute force attacks consist in sending attempts to a webpage trying to guess a user and password.
* File Inclusion Exploits.
  + Occur when vulnerable code is used to load remote files that allow attackers to gain access to a website. This attack is common used to gain access to the most important file in WordPress Installation, the wp-config.php.
* SQL Injections.
  + Usually occurs when you ask a user for an input (username/password), and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.
* Malware.
  + Code that is used to gain unauthorized access to a website to recollect sensitive data, usually stored in files. There are thousands of malware types, but the most common WordPress infections are:
    - Backdoors
    - Drive-by downloads
    - Pharma hacks
    - Malicious redirects.

*Solution:*

A solution that will help the WordPress community users, owners and developers is a WordPress Security Scan. The best scenario would be a scan that detect all 3972 vulnerabilities, but, because of academic reasons the Security Scan will only cover brute force attacks and SQL injections.

Why this solution help users?

As a user it’s difficult to trust a page with your personal data if you don’t know the page security level, but, if you really need to sign out to that webpage and want to know if your data will be secure, the WordPress Security Scan will tell you the answer.

Why this solution help owners and developers?

As a web developer beginner, it’s difficult to keep track of all good practices, one of them its security, if a developer or owner want to check if their pages are secure at least for the most common WordPress security vulnerabilities just introduce the URL and find out.

What makes your WordPress site more vulnerable to successful attacks?

1. Weak passwords
2. Not Updating WordPress, Plugins or Themes.
3. Using Plugins and Themes from untrusty sources
4. Using Poor-Quality or Shared Hosting

How can we protect our WordPress site?

1. Use Strong Passwords
2. Install a WordPress security plugin
3. Set up proper permissions on your server
4. Activate WordPress Brute Force Protection

How the program works?

The program will receive an URL from a WordPress site and will return if the site is vulnerable to SQL Injection and Brute Force Attacks.

Classes in the program:

* HttpRequest:
  + This class obtain the GET method to the URL that receives

(return all the html code of the given page).

* + Is a thread, in the run function:
    - The first step is to get a connection, the library HttpURLConnection help with this task.
    - Create an object type HttpURLConnection and call the method GET.
    - Safe the result in a response variable.
    - The system will print the exit code to notice if the connection was successful (the code will be 200, put the flag in 0). And if the connection isn’t successful, the program sends the error (flag in 1).
  + The catch will be the normal IOException with a message of thread terminated
* SQLInjection:
  + This class perform the SQL Injection to the URL that receives with a static attack: “log=’ or ‘1’=’1”.
  + Is a thread, in the run function:
    - This class make a POST request (that works like the GET request of the class HttpRequest) to the URL with the parameters in log-in and user fields with the attack: “log=’ or ‘1’=’1”.
    - If the response code of the attack is any error of redirect, denial of service or blocked it return a Site is protected (flag in 0). If don’t it will return the normal 200 success (flag in 1).
  + The catch will be the normal IOException with a message of thread terminated
* BruteForce:
  + This class perform the brute force attack to the URL that receives with the fields user: [flowers@hotmail.com](mailto:flowers@hotmail.com) and password: hola123.
  + Is a thread, in the run function:
    - This class make a POST request (that works like the GET request of the class HttpRequest) to the URL with the parameters in user: [flowers@hotmail.com](mailto:flowers@hotmail.com) and password: hola123 fields.
    - If the response code of the attack is any error of blocked or denial of service, it returns the site is protected (flag in 0). If don’t it will return the normal 200 success (flag in 1).
  + The catch will be the normal IOException with a message of thread terminated
* Main:
  + This class (main thread) perform the read of the URL, also the creation and management of all the threads for the Scan to work.
  + First read the URL and attach it with “/admin” for the GET and “/wp-login.php” to access the log in page of the URL.
  + Then create 2 objects of type SQLInjection and HttpRequest
  + Create 2 threads the first will execute what’s in HttpRequest class and the second will execute what’s in SQLInjection
  + Once the thread is created, we first start the thread that contains the HttpRequest class and while this thread is alive view the flag, if the flag is 1 (error in get), interrupt and join the thread to stop it.
  + Then we start the SQL thread and the same logic as the last thread, while is alive view the flag if the flag is 1 send the message “Your site is vulnerable against SQL Injections”.
  + Now we use the ExecutorService library that automatically provides a pool of threads and API for assigning tasks to it. We create a thread pool with capability of 10 threads. In a for cycle iterate 10 times ( because we plan to launch 10 brute force attacks). Then we create task number i (from iteration) of type BruteForce receiving the URL. Now we execute the task, with the submit function that allow us to execute and get the return value of the given task. Between each iteration we put a sleep of 5000 milliseconds to let the HTTP request (POST) respond to us. Then get the flag value of the task and if it is 0 we need to end the ExecutorService but with shutdownNow that destroy the ExecutorService immediately, it does not wait to the task to finish, we use this function because we only care about the flag to know the answer we don’t need the thread to keep running we already got what we need. To finish this function, we view if the iterator goes to 9, if true then send “Your site is vulnerable against brute force attacks”.
  + Finally, we interrupt first the HttpRequest thread apply a join, the do the same for the SQLInjection thread, interrupt and join, and, to end the ExecutorService by shutdown to reject any other task is given but letting the task that was executing finish.

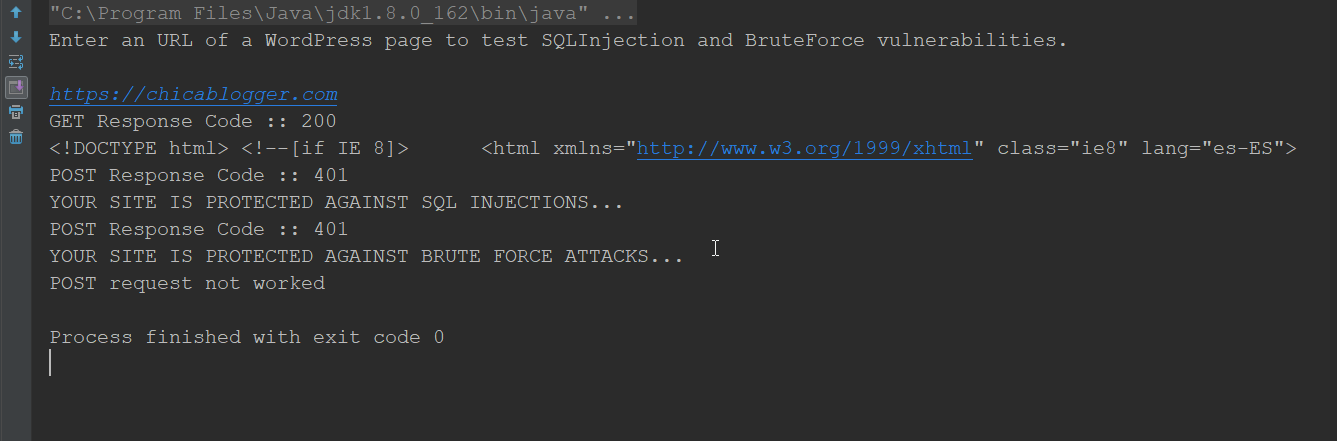
*Instructions:*

* Set up IntelliJ
* Open the project on IntelliJ
* Click Run Program
* Program will ask for an URL to test security (must be WordPress site).
* The program automatically will tell you if your site is secure or vulnerable against SQL Injections and Brute Force Attacks.

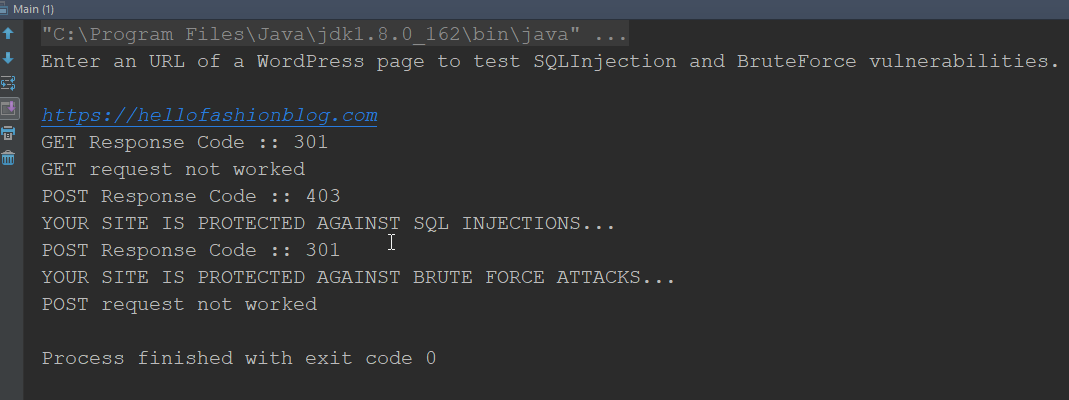
*Results(Testing):*

**All test performed in this document were carried out for academic purposes and in case of gaining access to the webpages, no malicious activity was carried out.**

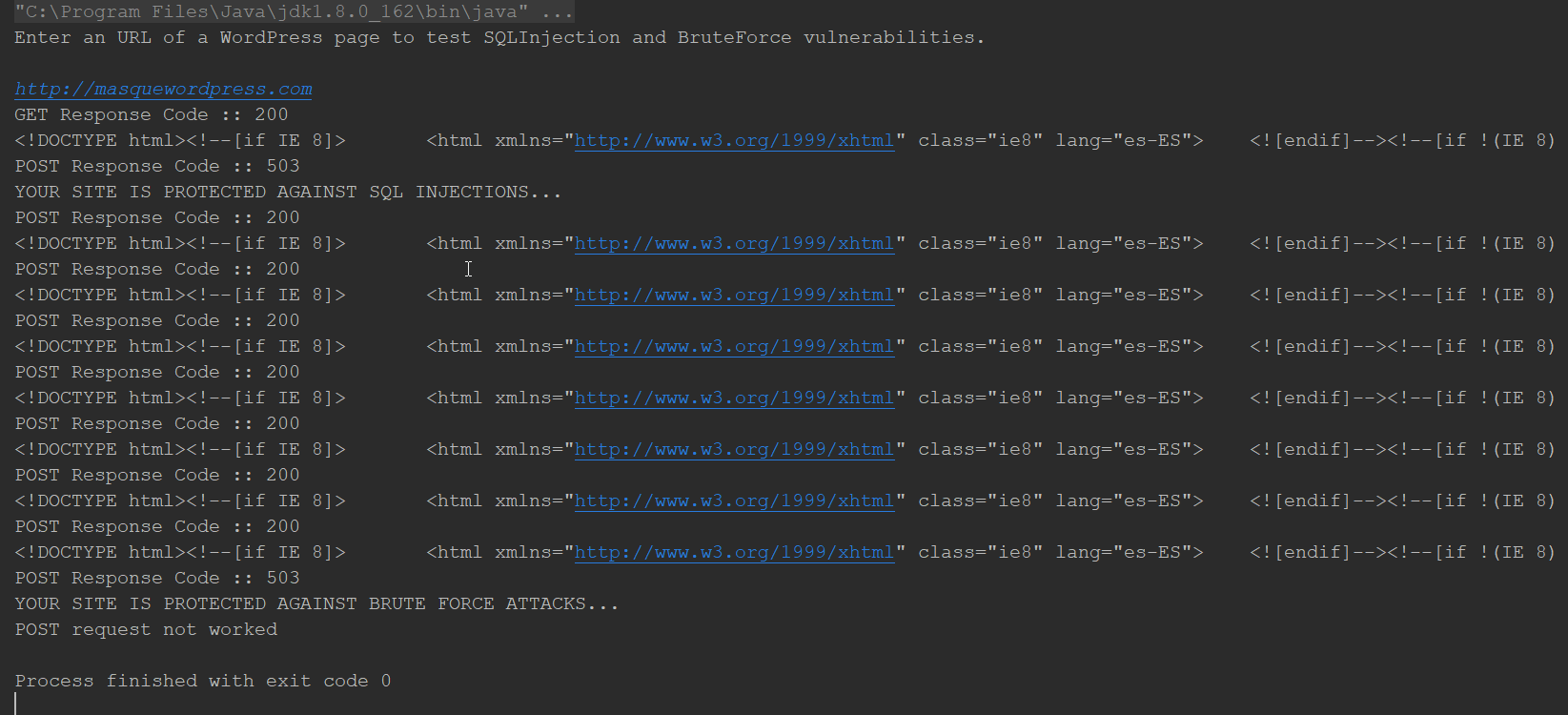
* <https://chicablogger.com>
  + Protected against BruteForce
  + Protected against SQL Injection



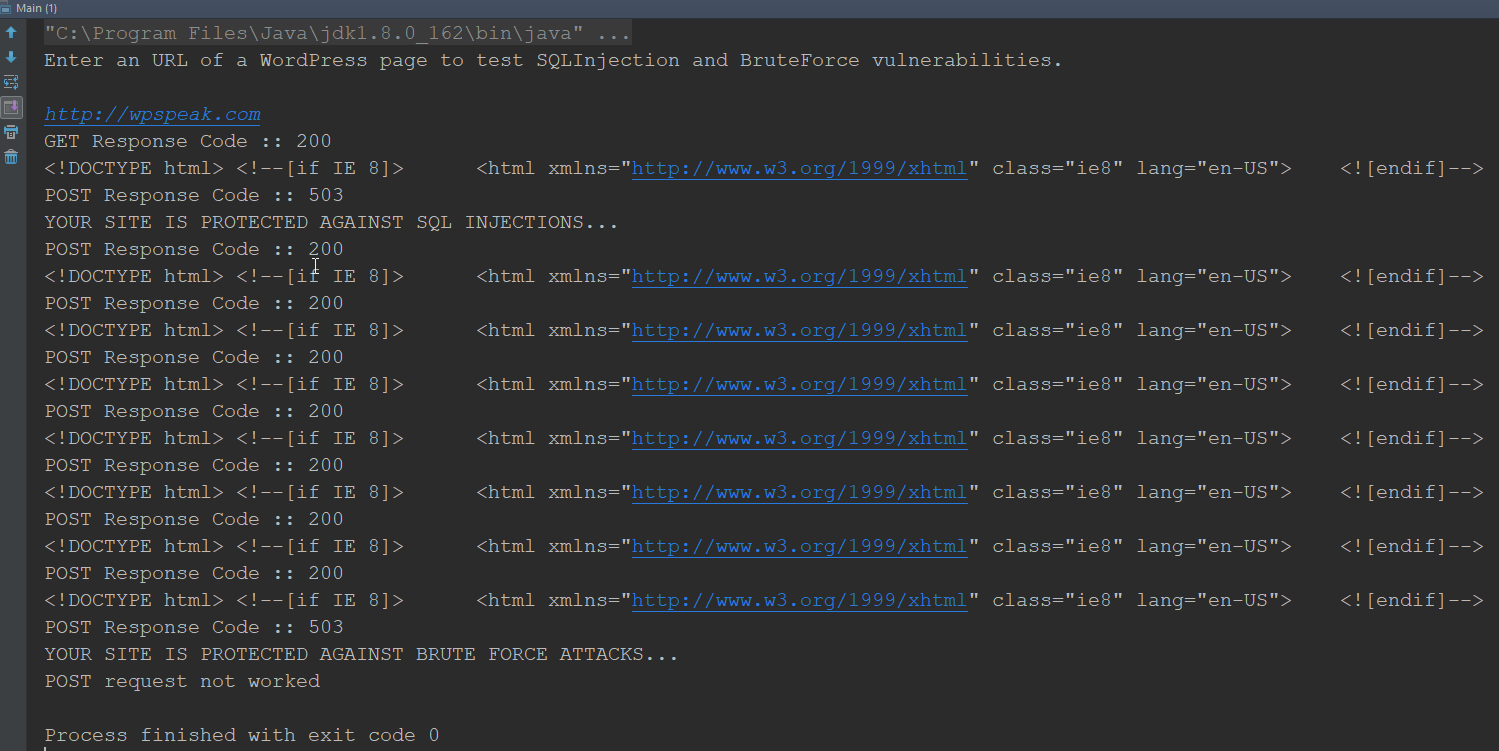
* <https://www.hellofashionblog.com>
  + Protected against BruteForce
  + Protected against SQL Injection



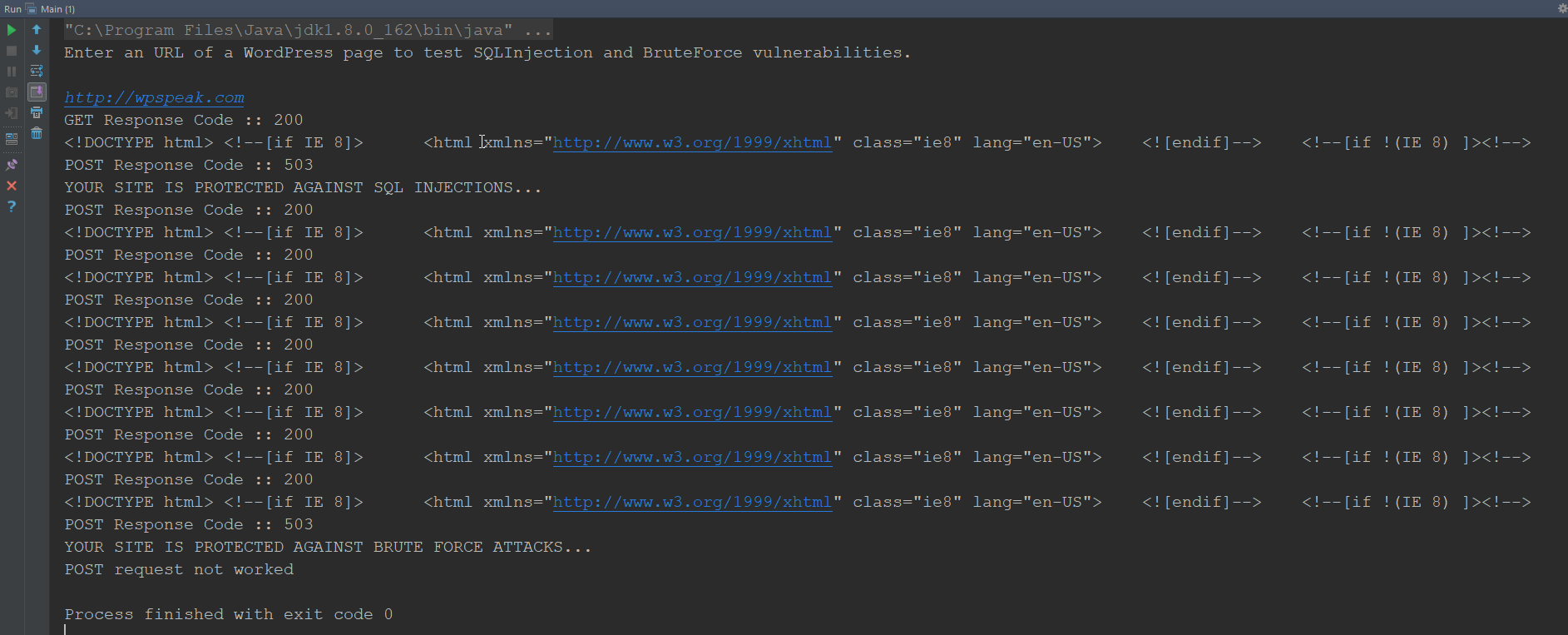
* <http://masquewordpress.com>
  + Protected against BruteForce
  + Protected against SQL Injection.



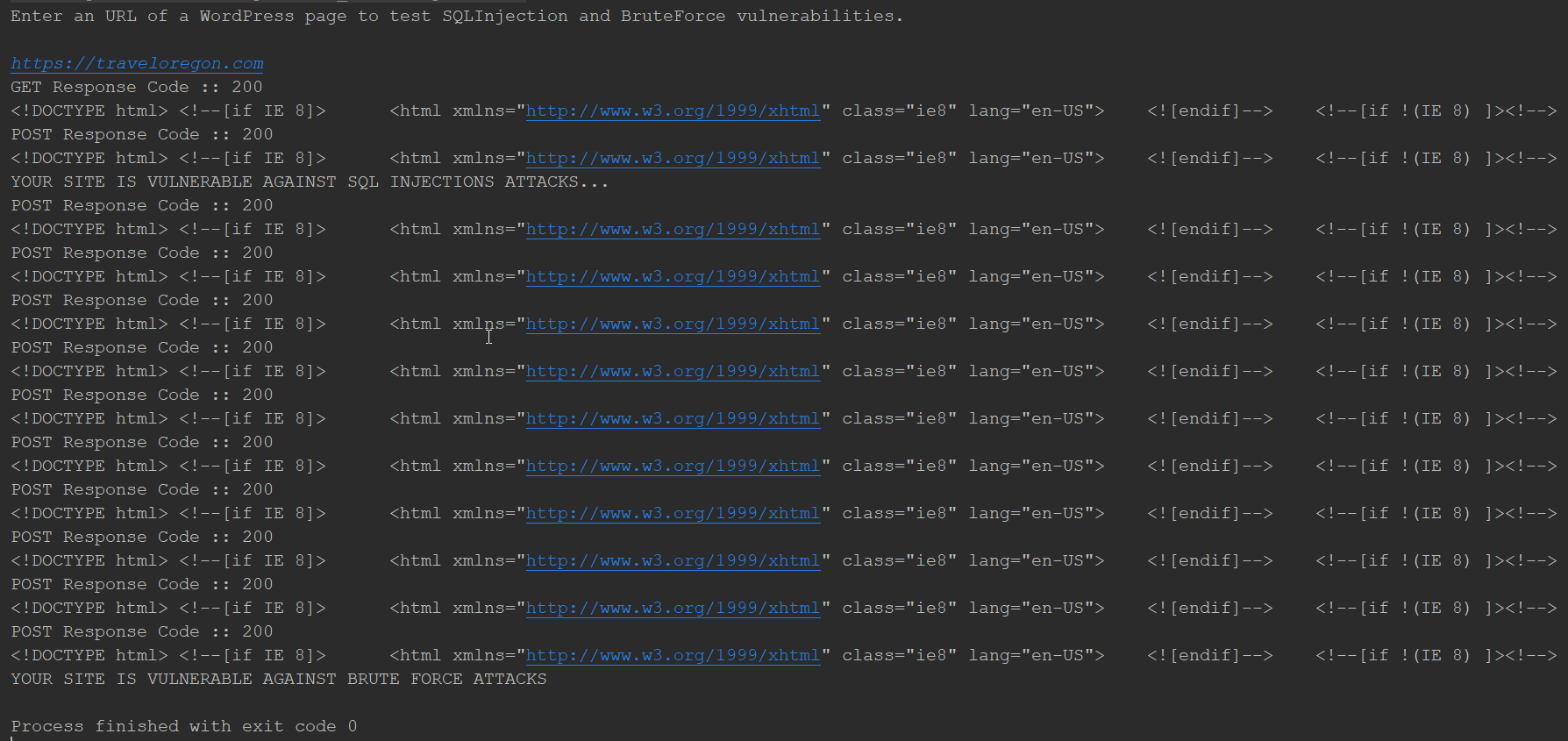
* <http://wpspeak.com>
  + Protected against BruteForce
  + Protected against SQL Injection



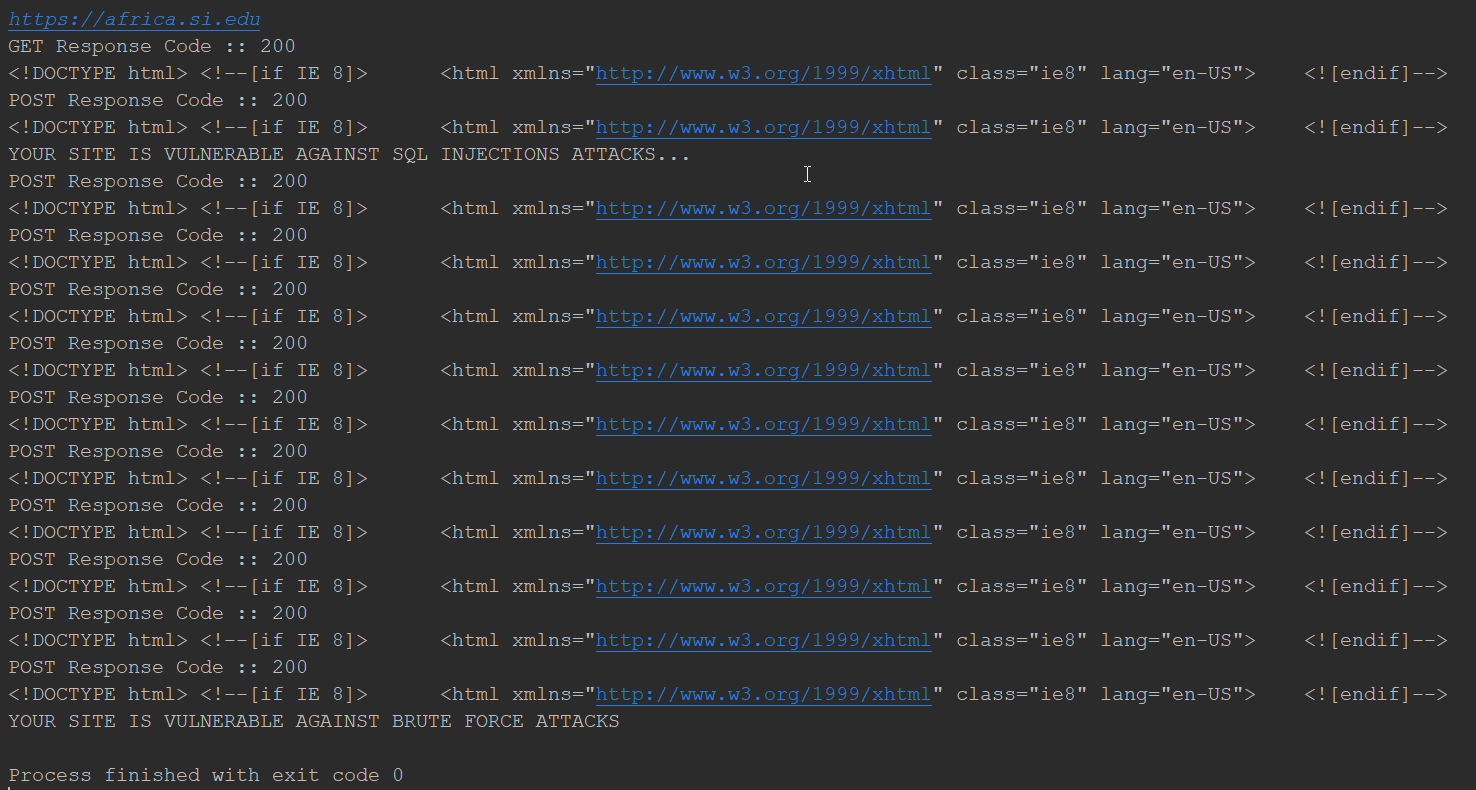
* <http://luismvillanueva.com>
  + Protected against Brute Force attacks
  + Protected against SQL Injection.



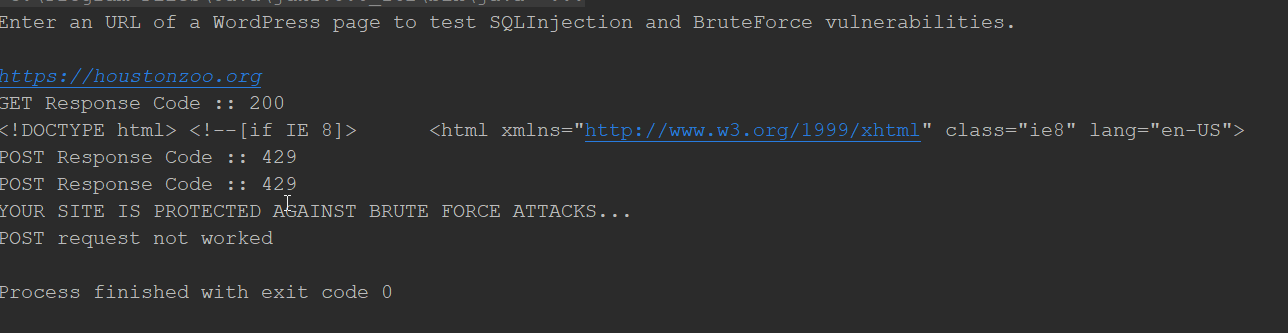
* <https://traveloregon.com/>
  + Vulnerable against Brute Force Attacks
  + Vulnerable against SQL Injection



* <https://africa.si.edu/>
  + Vulnerable against Brute Force Attacks
  + Vulnerable against SQL Injection



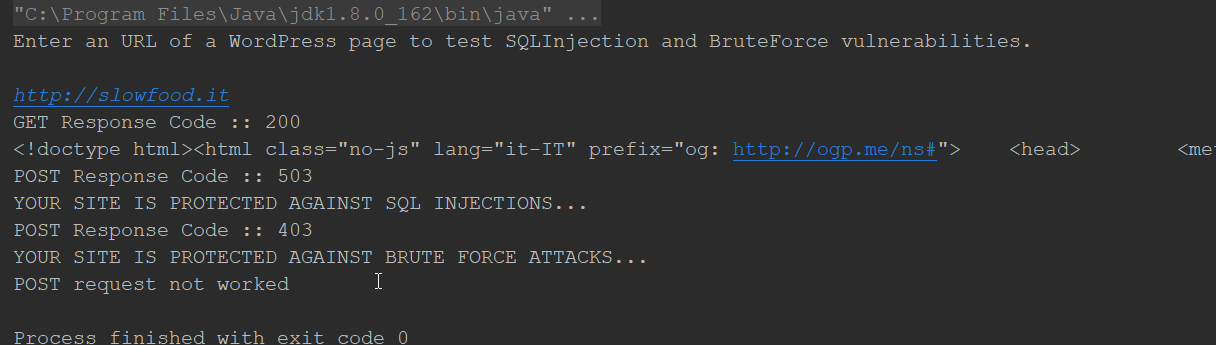
* <https://www.houstonzoo.org/>
  + Protected against Brute Force Attacks
  + Protected against SQL Injection



* <http://www.markwebber.com/>
  + Vulnerable against Brute Force Attacks
  + Protected against SQL Injection



* <http://www.cmoneda.cl/>
  + Protected against Brute Force Attacks
  + Protected against SQL Injection.



*Paradigm*:

Why compare it with CUDA?

Because of the number of threads could rise to thousands for multiples attacks I thought of using CUDA with parallel programming, but, here is why is better Concurrency in Java for this approach to the problem.

|  |  |  |
| --- | --- | --- |
| Problem | Concurrency in Java | Parallel in CUDA |
| HTTP requests | Because the GET and POST of HTTP in average can take 1 to 3 seconds if we execute 10 requests, maybe we only get 2 or 3 because we must let the server respond. Managing this problem with threads is the solution, we only make them sleep until the respond arrives. | You can handle requests in CUDA but, when executing code in parallel it’s not a good practice to use sleep, the ideal code (parallel) is to run all at the same time without interruptions. |
| Attacks | For this problem the best option is to handle each attack in a thread. At the moment one of the attacks is successful the problem should stop. With concurrency you can simply call an Interrupts() and using ExecutorService library to manage the threads. | CUDA is more for simulations and mathematics operations, it’s not good practice for the purposes of this program. |

*Conclusion:*

As we saw in this document, the security in WordPress pages is not bad, on the contrary it is relatively good. However, if good practices are not followed when designing or programming it, it can generate many security breaches. It might not sound risky to have a blog with low security, but when starting with bad practices it is difficult to get rid of that habit; when you are making a page with sensitive information and have important security flaws will generate legal, personal and economic problems. The program is a small contribution to the community of developers who want to test if your site resist’s the 2 most common attacks on WordPress pages, brute force attacks and SQL Injections, I hope that by using it, if any vulnerability is detected, you should follow the good practices and take little by little the computer security is one step ahead.

# Referencias

(s.f.). Obtenido de https://themeisle.com/wordpress-themes/

baeldung. (17 de Jan de 2018). *Baeldung*. Obtenido de Baeldung: http://www.baeldung.com/java-executor-wait-for-threads

https://wordpress.org/about/history/chapter3.pdf. (2003). *WordPress*. Obtenido de WordPress: https://wordpress.org/about/history/chapter3.pdf

Mander, M. (16 de Dec de 2016). *WordFence*. Obtenido de WordFence: https://www.wordfence.com/blog/2016/12/how-to-protect-against-brute-force-attacks/

mkyong. (22 de May de 2013). *Mkyong*. Obtenido de Myyong: http://www.mkyong.com/java/how-to-automate-login-a-website-java-example/

ORACLE. (s.f.). *ORACLE*. Obtenido de ORACLE: https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html

OWASP. (04 de Oct de 2016). *OWASP*. Obtenido de OWASP: https://www.owasp.org/index.php/SQL\_Injection

Pankaj. (2 de Apr de 2018). *JornalDev*. Obtenido de JornalDev: https://www.journaldev.com/7148/java-httpurlconnection-example-java-http-request-get-post

Smith, C. (05 de April de 2018). *DMR*. Obtenido de https://expandedramblings.com/index.php/wordpress-statistics/

Tomovic, N. (28 de Apr de 2016). *ManageWP*. Obtenido de ManageWP: https://managewp.com/blog/statistics-about-wordpress-usage

*Vishnu*. (10 de Jan de 2017). Obtenido de https://colorlib.com/wp/blogs-using-wordpress/

W3Schools. (s.f.). *W3Schools*. Obtenido de W3Schools: https://www.w3schools.com/sql/sql\_injection.asp

*W3Techs*. (s.f.). Obtenido de W3Techs: https://w3techs.com/

*Wikipedia*. (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos\_de\_estado\_HTTP

*WordPress*. (s.f.). Obtenido de PLUGINS: https://es-mx.wordpress.org/plugins/

WPScan. (s.f.). *WPScan*. Obtenido de WPScan: https://wpscan.org/

Wright, K. (16 de Jan de 2017). *IThemes*. Obtenido de Ithemes: https://ithemes.com/2017/01/16/wordpress-security-issues/