PiSecurityCheck: Server Security Check in a Hand

Pio Treglia
Grad. at Università "La Sapienza" di Roma, Rome, Italy
piotreglia90@gmail.com

***Abstract* — Nowadays, due to the Ukrainian-Russian war, Denial of Service attacks against major Institutions across Europe are increasing. The majority of them are application layer (L7) attacks in which slow HTTP attacks play a major role. In this paper, it is presented PiSecurityCheck, an Android application designed to check in an intuitive and fast way with a minimum amount of bandwidth, if a web server may be prone to slow HTTP attacks. It will be shown how a mobile application can emulate a DoS attack, based on different parameters set by the user. Apache and IIS will be tested in their default configuration and the results compared with slowhttptest output, to corroborate the validity of PiSecurityCheck.** (*Abstract*)

Keywords - android; mobile attack; cybersecurity; slow dos attack; denial of service

I. INTRODUCTION

During 2022, due to the geopolitical unrest related to the Russian-Ukrainian crisis, the number of Application Layer DDoS (L7) increased by 82% with respect to 2021 [1]. Advanced Persistent Threat Groups from both factions (for example the pro-Russian KillNet or the pro-Ukranian IT Army of Ukraine) supported their nations resulting in sparse attacks to the main institutional websites and portals of different countries in the world (e.g. US [2], UK [3], Germany [4], Czech Republic [5]).

One of the main kinds of application layer attacks that have been used is slow HTTP. This kind of threat is different from the typical DoS. In fact, the legacy well-known Denial of Service attack was usually based on a flooding strategy, with millions of requests and network-consuming traffic, to exploit and waste all the available bandwidth of the recipient. In the SDA (Slow DoS Attack) instead, the attacker aims to open as many HTTP connections as possible, simulating a sender with degraded network performance, to waste the webserver resources leading it to discard any other legitimate request.

A classification of this attack has been given [6] and a general taxonomy has been detailed in the past [7]. For the sake of this paper two implementations will be taken into account, namely the Slowloris HTTP attack and the slow body HTTP attack (also known as R.U.D.Y.).

The paper is structured as follows. Section II gives a general review of both attacks, underling how they are still valid today and how spread they currently are. Section III presents PiSecurityCheck, a new Android application made to help administrators and tech personnel spot, check and configure correctly the infrastructure they manage. Section IV shows the usage of the application with Apache and IIS web servers. Section V reports the conclusion and future works.

II. REVIEW OF SLOW ATTACK STRATEGY

A. *HTTP Protocol*

HTTP is an application layer protocol [8], which allows the transfer of hypertext pages through the use of methods and variables sent with an undergoing TCP connection. A common HTTP request ("Fig. 1") is made up of:

- request line: includes the method (e.g. GET, POST, and others), request-target, which can be either a URI or an URL, and the HTTP version;
- headers: used to send additional information like cookies and authorization tokens. They are case sensitive and are defined by a name, a semicolon ":" and a value;
- message body: used to exchange information between client and server. The request and the headers must all end with a CRLF (carriage return and line feed) and a final empty line (CRLF) indicates that the request is complete and can be processed.

Once the server receives the request, it validates all the methods, headers, and fields and sends the response back to the sender based on the resource requested. The body part of the response is used to transfer the information like an HTML page to be displayed in a web browser.

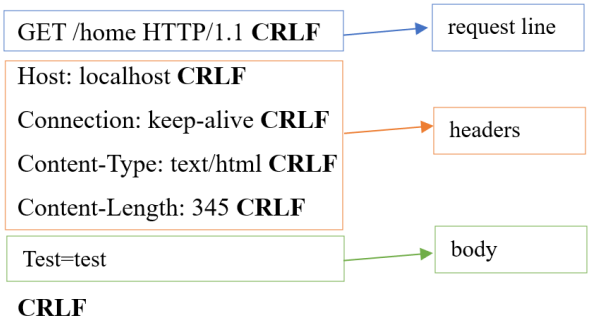

Figure 1. Standard HTTP request.

B. *Slowloris Attack*

The Slowloris HTTP attack [9] exploits the normal behavior of a webserver that waits for the end of an HTTP request indefinitely (or the expiration of a preset timeout) before closing the connection. By default a web server allows slow connections to send information with a low transmission rate due to degraded communication channels. As a result, an attacker can keep multiple connections open by sending small pieces of information in a large amount of

time without sending the final empty line that states the end of the request. Doing so the server would waste all the resources waiting for the useless HTTP requests, leading to the unavailability of the service for a legitimate user.

C. *Slow Body HTTP*

The attack methodology is very similar to the one seen in the paragraph above. The final goal is to exhaust all the resources of the server by setting up multiple dumb connections that transmit data at a very low rate. But in this case, there are two main differences:

- the POST method is used;
- the attacker sends the header field "Content-Length" (normally used to announce to the recipient the dimension of the body of the request) set to a very large number.

Doing so the server would keep the underlying connection open, waiting for the useless connection to finally transmit all the body content announced in the first place inside the header. From time to time, the attacker sends a small piece of information (usually one or two bytes), just to be sure that the recipient will not close the connection, so the resources would be kept busy.

III. PiSecurityCheck Implementation

A. *Related Work*

In the past, an android application called "SlowDroid" has been already presented by Cambiaso, Papaleo and Aiello [11]. In that case, the main purpose was to demonstrate how a mobile phone could be turned into a threat and launch a DoS attack. Moreover, it allowed to setup unencrypted connections and start a Slowloris attack. Other implementations of slowloris-attack checking tools are currently available in a script form, like *http-slowloris-check* [20] or *slowhttptest* [12]. Most of them are not user friendly and do not present the results through an intuitive graphical user interface. So far, no other android or graphical based slowloris testing application have been presented.

B. *Application Overview*

The main purpose of the the PiSecurityCheck [10] application, is to give administrators a valid tool to:

- check, by using a minimal amount of bandwidth, if a web server may be vulnerable to the attack briefly explained in section II-B;
- emulate a real attack and take the proper countermeasures to enhance the security.

I developed this tool as an Android application because it is more simple and more practical. It needs a minimum amount of configuration to be ready to use. Moreover, sometimes similar tools or scripts that are currently widely used can be misleading. Using an application that actually emulates an attack, can help to double-check the accuracy of other well-known software. This aspect in particular will be shown in section III, where it is presented a comparison between the PiSecurityCheck application and slowhttptest [12].

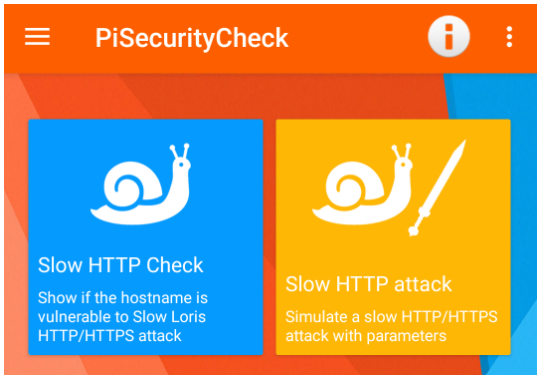

Figure 2. Two features implemented in the application

C. *Functions*

PiSecurityCheck has two main functions that the user can select:

- *Slow HTTP Check*, which allows the user to check if the web server may be vulnerable to the Slowloris attack seen in section II-B;
- *Slow HTTP attack*, with which the user can launch an attack, by configuring some editable fields

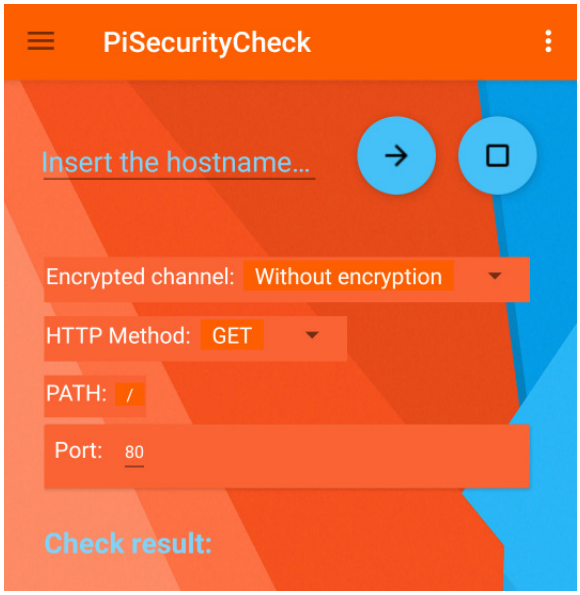

Figure 3. Input and configuration to be set by the user in the Slow HTTP Check functionality

Both functions have five main parameters that can be edited by the user:

- hostname (`String`): the host check/attack;
- channel: specifies if the socket has to be opened through a SSL/TLS connection, or with an unencrypted one;
- HTTP Method: either GET or POST;
- path (`String`): the path of the hostname to whom deliver the request;

- port (`int`): on which port the user wants to perform the check/attack

This application has been specifically designed to have all these fields because the aim is to have a flexible tool to be used intuitively. Above all, it is very useful to conduct a test on a non-standard port (like 80 for HTTP or 443 for HTTPS) and analyze in practice if the web server is actually vulnerable. The function Slow HTTP attack has two more editable parameters:

- number of connections (`int`): specifies the number of parallel connections that the application opens during an attack. The maximum number of connections is limited to 1024;
- test maximum duration (`int`): defines a timeout in seconds, that will be used as the duration of the attack. The maximum duration is set to 120 seconds if a bigger value is provided.

The above limitations have been introduced because the aim of the application is not to build a cyber weapon. For this reason, the application has been also obfuscated in order to avoid decompilation of the application

D. *Graphical User Interface*

The Graphical User Interface has been designed having in mind a material design pattern [14]. Efficiency has also been taken into consideration, so the information to be displayed is rendered on a need-to-use base. This results in showing information only when the user actually needs to see the particular item in a list (without pre-loading all the information in the view). It also "recycles" already-seen information without destroying the view but saving the references in a so-called "recycle bin" [13]. All these features result in better performances and above all lower battery consumption, while keeping in mind the goal of having a user-friendly application. The start and stop buttons of both functions are FloatingActionButton and together with the Toolbar allow the application to implement the third dimension defined by Google in the material design guidelines.

E. *Slow HTTP Check Implementation*

Through Slow HTTP check function, based on user input, it will be tested either:

- Slowloris vulnerability, if the selected method is GET;
- Slow body vulnerability, if the POST method is selected

1) Slowloris vulnerability: the approach followed in the implementation, is based on a previous article [16]. In the beginning, two sockets are created and fed with two identical HTTP GET requests without the final CRLF. Doing so the application would emulate a slow connection and the web server would wait for the client to send any additional bytes. After a period T (in the application T is fixed, equal to 10 seconds) a "refresher" is sent to the web server using the second socket (instead the first stays still,

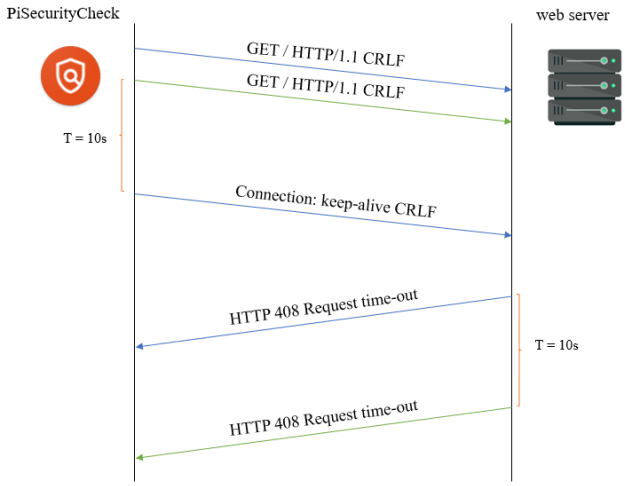

Figure 4. Request and response exchange of the Slow HTTP check function with the GET method selected.

not sending any other information). If the second socket is closed exactly or more than T seconds after the first connection, it means the server may be vulnerable to the attack. In fact, it perceived the connection as slow so it is configured to wait some time before closing the communication. This is a flaw because an attacker could send a "refresher" every T-1 second so the server keeps the connection open and the related resources are kept busy. A proper configuration of the server, as previously suggested [15], would limit this effect by setting:

- a minimum transfer rate for connections (so that the server would automatically close malicious requests);
- an appropriate fixed timeout for all the requests, so the additional byte sent will not extend the value of the timeout connection

2) Slow Body Vulnerability: the first check is done in the same way as the preceding paragraph, but the method used this time is POST. Additionally, it is also checked if the server is vulnerable to the slow body - R.U.D.Y (Are U Dead Yet) attack. In this case, a socket is opened and fed with a properly formed POST request. Moreover the header field "Content-Length" is set with a very large value (65543), emulating a large message body to be sent to the server. If the response is an HTTP code of 200, it means the web server has accepted the request and may wait for the last byte of the request to be correctly received. Using the same introduced in the last paragraph, could mitigate this behavior and easily defeat any similar attacks.

F. *Slow HTTP Attack Implementation*

This function has been designed to demonstrate in an empirical way if the administrator successfully implemented the correct countermeasures for the slow HTTP attack. So it is possible for the user, to insert and select the desired parameters and the application would start the attack if the right arrow is tapped. It has been implemented also a stop button, so it is possible to stop the

attack even when it has been already launched. Based on the selected method, the application would open a number of sockets (the default value is set to 300 and it is not possible to open more than 1024 parallel connections) with a timeout value specified by the user (the maximum value for the timeout is 120 so that after this time period all the connections are closed). Every socket is attached to a concurrent thread. If for any reason the thread crashes, the socket fails, or is interrupted by the server, the application automatically set up a new thread with a new socket and tries again to establish the connection. All the requests rely on a TCP socket and if the user selects the method:

- GET, the application sends a not properly terminated HTTP request (without the final CRLF). Every thread fills the socket every two seconds with a follow-up HTTP header similar parameter (code depicted in "Fig. 4"). Doing so the server would keep open up to 1024 connections, saturating eventually (if vulnerable to the attack) all the available resources;

```
Thread.sleep(SLEEP_TIME);
wr.write( str: "test_header: " + "test_header_value" + "\r\n");
wr.flush();
```

Figure 5. Piece of code where the junk "refresher" parameter is sent every 2 seconds.

- POST, the application transmits a well-formed HTTP request with a parameter "Content-length" set to a very large value (65543 - as shown in "Fig. 6"). In this case, every thread sends a randomly generated byte to emulate a slow connection thus inducing the server to wait for the remaining bytes.

```
wr.write(("POST " + PATH+" HTTP/1.1"+ "\r\n"));
wr.write(("Host: " + HOSTNAME + ":"+PORT+ "\r\n"));
wr.write(("User-Agent: Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 10.0;) " + "\r\n"));
wr.write(("Connection: keep-alive"+ "\r\n"));
wr.write(("Referer: SecurityCheck"+ "\r\n"));
wr.write(("Content-Type: application/x-www-form-urlencoded"+ "\r\n"));
wr.write(("Content-Length: 65543"+ "\r\n"));
wr.write(("Accept: text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5"+ "\r\n"));
wr.write( str: "\r\n") ;
wr.write( str: "test=t") ;
//wr.write("\r\n") ;
wr.flush();
```

Figure 6. Piece of code where the POST request is built with a large Content-length parameter set.

IV. PiSecirityCheck in Use

In order to check the features implemented in the application, two webservers have been installed on a Windows 10 system: Apache server and Internet Information Services (IIS) [18]. The PiSecurityCheck application has been used on both of them, in order to check their behaviour with the default configuration. Apache/2.4.54 with PHP 8.2.0 application server have been deployed by using the XAMPP [17] software. The web server has the default settings declared mainly in httpd-default.conf and `httpd.conf` files. In particular, the following values are defined:

- `Timeout 300` - the number of seconds before receives and sends time out;
- `KeepAlive On` - whether or not to allow persistent connections;
- `MaxKeepAliveRequests 100` - the maximum number of requests to allow during a persistent connection;
- `KeepAliveTimeout 5` - Number of seconds to wait for the next request from the same client on the same connection

So by default, there is no protection at all for the Slow HTTP attack family. Moreover, the line in the configuration file that imports the request timeout module (`mod\reqtimeout.so`) is commented. The Microsoft IIS 10.0 default configuration values are present at `applicationHost.config`. The `executionTimeout` is set to 00:01:50 lower than the Apache webserver. Anyway, no countermeasure are set by default against the slow HTTP attack. Both web servers have been tested with a cellphone (Samsung Galaxy S22 running the PiSecurityCheck application) and a Windows 10 client (running the server to test) connected to the same WiFi connection. The IP address of the cellphone is 192.168.1.221, instead, the server has 192.168.1.242. The default webpage of Apache has been modified in order to support a form field, so the slow body attack feature could be tested. The same field has been created also in the IIS deploying a custom webpage.

A. *PiSecurityCheck usage on Default Apache Web Server*

Executing the Slow HTTP Check, on the Apache webserver has demonstrated how it may be vulnerable to the attack. The confirmation of this has been given by launching an attack, using the default parameters through the Slow HTTP attack feature. After only 1 second the webserver was unable to handle any other requests.

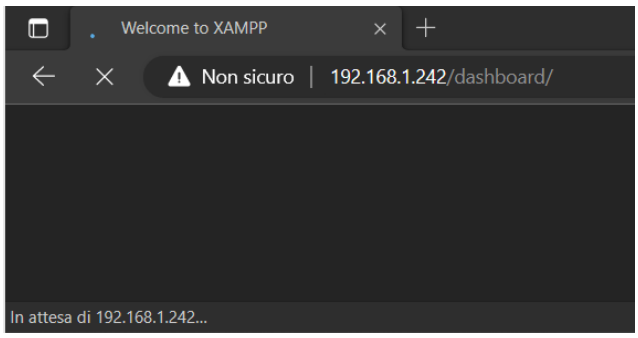

Figure 7. Trying to visit the default page of the Apache server after launching the attack

B. *PiSecurityCheck usage on Apache Web Server properly configured*

On Apache, a possible countermeasure to slow HTTP attack is enabling the `mod\reqtimeout.so` in the `http.conf`. This module is specifically designed to set a minimum rate and a specified timeout for all incoming requests. By default, it defines the following limits:

- `header=10-30`
- `MinRate=500`

Using this setting and launching again the attack, the service is always up and running, because the web server automatically discards incoming malicious requests. Proof

of this can also be seen in the server log file. Multiple 408 HTTP responses are reported, showing how the web server is terminating all the malicious connections, keeping the server up.

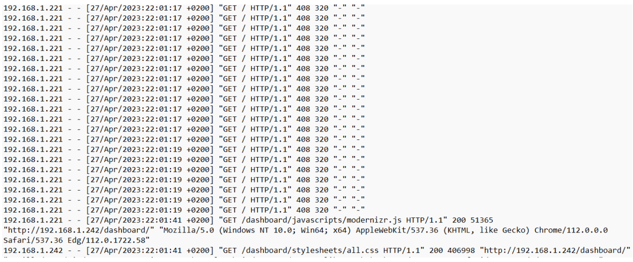

Figure 8. Apache logs applying the proper configuration

C. *PiSecurityCheck usage on Default IIS Web Server*

In order to test IIS version 10.0 web server, a very simple asp.NET page has been developed and deployed at the path `http://192.168.1.242/post`. It will be later used to check if the server is vulnerable to the slow body R.U.D.Y attack, introduced in section III-E2. The following are the default settings:

- `Limit bandwidth usage: disabled` - there are no protections against connections that try to seize the bandwidth;
- `Timeout connections: 120s` - compared to the Apache server the timeout threshold is lower;
- `Limit number of connections: disabled` – a client can open an infinite amount of parallel connections.

1) PiSecurityCheck - Slowloris Attack to IIS 10.0: launching the check feature on the post page, it can be noticed that the server automatically kills the not-closed HTTP request, exactly after 120 seconds. Then the attack feature is used to emulate a Slowloris attack. All default values (so GET method, unencrypted connections) are left unchanged, except for the parallel connections, which have been increased to 1024. The web server is still available and ready to serve connections.

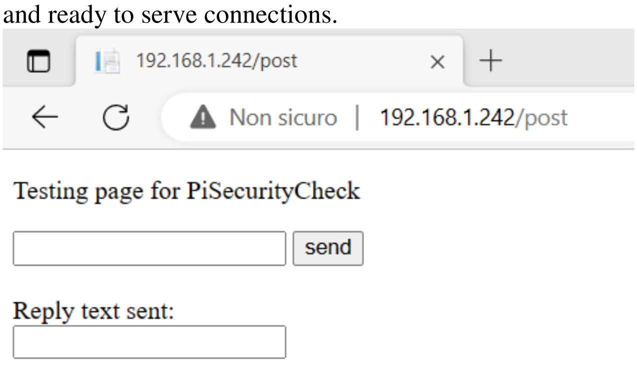

Figure 9. The IIS server is still available, after a Slowloris HTTP attack with 1024 connections

2) PiSecurityCheck – Slow Body attack to IIS: using the Slow HTTP attack feature, the method has been changed to POST, and the number of parallel connections has been set to 1024. All the other values have been left unchanged. This time the webserver was not available anymore, in fact trying to access the post page, the output was a `Error 503 - Service Unavailabe`. The same not-working result has been obtained also by testing IIS version 9.0.

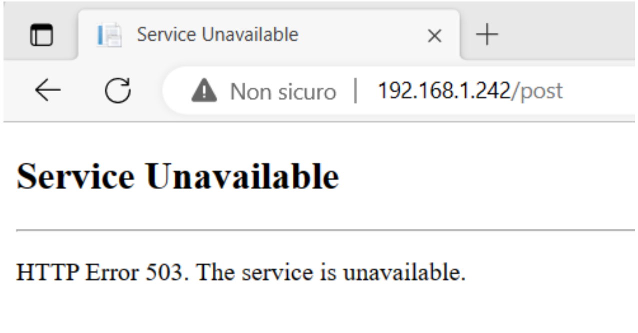

Figure 10. The IIS server is not available, after a slow body attack with 1024 connections.

D. *Slowhttptest usage on Default IIS Web Server*

slowhttptest [12] is a well-known HTTP tool specifically designed to test if a service is vulnerable to slow HTTP attacks. It is automatically included in an ethical hacking Linux-based distribution named “Kali Linux” [19]. For this test, a pre-configured kali linux virtual machine has been installed and used to conduct the attacks against the IIS web server. It is useful to conduct the very same slow body R.U.D.Y attack implemented by a different tool, so it is possible to compare the behavior of the server. The string launched to attack is shown in “Fig. 11”, where the most important parameters are: -B means slow body HTTP attack; -r specifies that 1024 parallel connections will be sent.

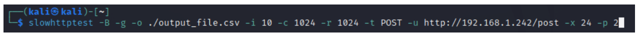

Figure 11. Slowhttptest string line with attack parameters set.

E. *Outcome Analysis*

Looking at the output of slowhttptest (“Fig. 12”), it can be seen how IIS is vulnerable to slow body HTTP attacks. This outcome confirms what was previously seen with PiSecurityCheck.

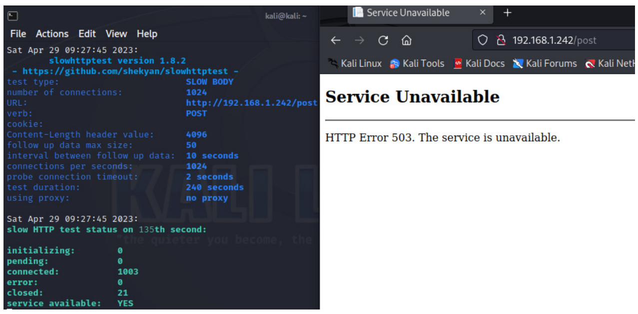

Figure 12. slowhttptest attacking IIS and unavailable post page

It is also noticeable how slowhttptest is wrongly assuming that the webserver is UP. In fact, accessing directly the post page, the result is always an HTTP 503 error code. This behavior can be justified by the fact that

IIS responds with an error code, so the tool assumes that the service is up due to the response. Probably slowhttptest would not check the content of the response. In fact, in slow HTTP attacks, the normal outcome for a request to a busy server is a pending connection that eventually expires with a timeout. This bug is also visible looking at the html file that outputs the tool. “Fig. 13” depicts the graph with all the connections that were sent during the test, colored by the closed and the connected ones. The green instead shows the availability of the service. As said before slowhttptest consider mistakenly the web server as up and correctly running.

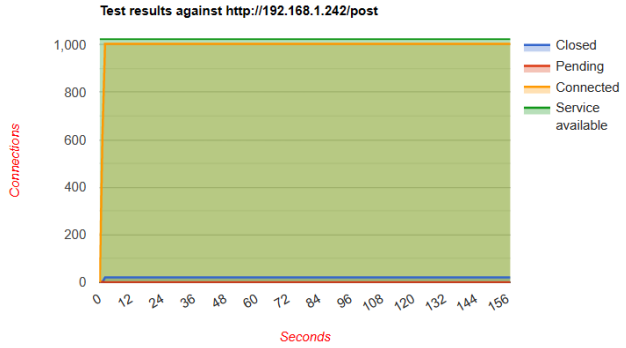

Figure 13. slowhttptest graph result.

V. Conclusion and Future Work

In this paper, a new Android application has been presented. It has been shown how fast and simple is, to set it up. Its main features have been used to demonstrate how useful it can be to web server administrators. Some tests have been conducted on Apache and IIS webserver and the results compared with another very known tool. All the results lead to the goodness of the application and its flexibility in the configuration. During the research, a bug of slowhttptest was also noticed (the tool presented the server available, even though it was not responsive). After analyzing the HTTP responses, a possible explanation for the malfunctioning has been given.

In the future PiSecurityCheck can be further implemented, adding new functionalities, thanks to the modularity of the application. The attack module, in particular, can be extended with more parameters. The wait timeout could be introduced as a field to be set by the user. Also the total duration of the attack could be a parameter of interest. This way an administrator could check both the behavior of the web server, minimize the downtime in case of vulnerability and fine tune the configuration related to the timeout settings of HTTP requests.

References

[1] *G. McKeever,"The Imperva Global DDoS Threat Landscape Report 2023".* Accessed *on 12.29.2023* [Online]. Available: https://www.imperva.com/blog/imperva-releases-its-global-ddos-threat-landscape-report-2023

[2] C. Pallardy, "Understanding DDoS Attacks on US Airport Websites and Escalating Critical Infrastructure Cyberattacks". Accessed on 12.29.2023 [Online]. Available: https://www.informationweek.com/security-and-risk-strategy/understanding-ddos-attacks-on-us-airport-websites-and-escalating-critical-infrastructure-cyberattacks

[3] Foreign, Commonwealth & Development Office and National Cyber Security Centre, "UK assesses Russian involvement in cyber attacks on Ukraine". Accessed on 12.29.2023 [Online]. Available: https://www.gov.uk/government/news/uk-assess-russian-involvement-in-cyber-attacks-on-ukraine

[4] BfV (2022), “Cyber-Brief Nr. 01/2022 – Hinweis auf aktuelle Angriffskampagne”. Accessed on 12.29.2023 [Online]. Available: https://www.verfassungsschutz.de/SharedDocs/publikationen/DE/cyberabwehr/2022-01-bfv-cyber-brief.pdf?__blob=publicationFile&v=10

[5] M. Bagwe, "Pro-Russian Killnet Group in DDoS Attacks on Czech Entities". Accessed on 12.29.2023 [Online]. Available: https://www.bankinfosecurity.com/pro-russian-killnet-group-in-ddos-attacks-on-czech-entities-a-18949

[6] E. Cambiaso, G. Papaleo, and M. Aiello, “Taxonomy of Slow DoS Attacks to Web Applications”, Communications in Computer and Information Science 335; October 2012.

[7] E. Cambiaso, G. Papaleo, G. Chiola, M. Aiello, “Slow DoS attacks: definition and categorisation, ”International Journal of Trust Management in Computing and Communications, vol. 1, no. 3, pp. 300–319, 2013.

[8] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee, ”Hypertext Transfer Protocol – HTTP/1.1”, RFC 2068, January 1997, Accessed on 12.29.23 [Online]. Available at https://datatracker.ietf.org/doc/html/rfc2068

[9] Wikipedia, “Slowloris”. Accessed on 12.29.23 [Online]. Available at http://en.wikipedia.org/wiki/Slowloris

[10] Pio Treglia, “PiSecurityCheck, an Android tool for your security”- Accessed on 12.29.23 [Online]. Available at https://pisecuritycheck.000webhostapp.com/.

[11] E. Cambiaso, G. Papaleo, G. Chiola and M. Aiello, ”Mobile executions of Slow DoS Attacks”, Logic Journal of the IGPL, Vol. 24, Issue 1, pp. 54–67, Feb 2016.

[12] Google-Code, “slowhttptest - Application Layer DoS attack simulator”. Accessed on 12.29.2023 [Online]. Available: https://code.google.com/p/slowhttptest/

[13] Google LLC, *ReciclerView*. Accessed on 12.29.2023 [Online]. Available: https://developer.android.com/develop/ui/views/layout/recyclerview

[14] Google LLC, *Material design*. Accessed on 12.29.2023 [Online]. Available: https://m2.material.io/develop/android

[15] Suroto, ”A Review of Defense Against Slow HTTP Attack”, International Journal on Informatics Visualization, Vol.1 no.4, 2017.

[16] S. Shekyan, “Are you ready for slow reading?”. Accessed on 12.29.2023 [Online]. Available: https://blog.qualys.com/vulnerabilities-threatresearch/2011/07/07/identifying-slow-http-attack-vulnerabilities-onweb-applications”

[17] Apache foundation, *Apache HTTP Server project*, Accessed on 12.29.2023 [Online]. Available: https://www.apachefriends.org/it/index.html

[18] Microsoft, *IIS*. Accessed on 12.29.2023 [Online]. Available: https://www.iis.net/

[19] OffSec Services, *Kali linux*. Accessed on 12.29.2023 [Online]. Available: https://www.kali.org/get-kali/#kali-virtual-machines

[20] Nmap, *Script http-slowloris-check*. Accessed on 04.01.2024 [Online]. Available: https://nmap.org/nsedoc/scripts/http-slowloris-check.html