WEB SERVER SECURITY ANALYZATION TOOL

Samuel Kopecký
Bachelor Degree Programme (3), FEEC BUT
E-mail: xkopec52@stud.feec.vutbr.cz

Supervised by: David Smékal
E-mail: smekald@feec.vutbr.cz

Abstract: Nowadays it is common practice to have a securely hosted web server. To have a secure web server it is required to ensure that CIA (Confidentiality, Integrity, Availability) triad is in place. CIA triad is fulfilled by using different mechanisms like symmetric cryptography and public key infrastructure. These mechanisms on the other side use a set of algorithms and parameters. This paper focuses on the automation of analyzing the security of these algorithms, parameters and also vulnerability testing.

Keywords: SSL, TLS, Web Server, Automation, Security, Analyze

1 INTRODUCTION

In order to use a websites safely, it is necessary to secure confidentiality (protection against unauthorized access to the communication), integrity (protection against the modification of sent information) and authenticity (identity verification of the communicating entities) of the data being transmitted. The HTTP (Hypertext Transfer Protocol) protocol itself, which is used to transfer the data between a client and a server, doesn’t include any mechanisms to ensure these requirements. That’s why the protocol HTTPS (HTTP secure) was created, to ensure the communication is secured via a “tunnel” created by the protocol SSL/TLS (Secure Socket Layer/Transport Layer Security) [1].

The protocol TLS contains many mechanisms, which it uses to secure confidentiality, integrity and authenticity [2]. In order to use these mechanism, the client and server need to agree on specific parameters and algorithms that will be used during the transmission of messages.

However, before the client and server exchange the required information, every TLS connection needs to start with an exchange of messages using the handshake protocol. If the client wants to establish a secure connection with the server, he first sends a message notifying him that he wants to create a connection. The server then responds to this message. It is during the handshake protocol that the communicating entities exchange the necessary algorithms and parameters to establish a secure connection. [2]

2 PROGRAM FUNCTIONALITY

The program is written in the programming language Python and it is able to analyze the parameters and algorithms that the client and server exchange during the communication establishment. More precisely it analyzes the supported TLS protocol versions, implementations of a web server, server certificates, used cryptographic parameters and algorithms (cipher suite) and a chosen set of vulnerabilities, to which the server may be vulnerable. For the analysis of the cipher suite, supported versions of the TLS protocol and vulnerabilities a custom implementation of the analysis algorithm is used. Certificate analyzation is done by using the python library cryptography [3]. Scanning of the web server implementation is performed using the Nmap tool and by using the information inside the HTTP response headers. The collected data is then printed to the user of the program.
Functionality of the program is divided into several logical parts, that are as a whole responsible for the analyzation of the web server and its parameters. The following chapters describe these logical parts of the program. These logical parts are also illustrated by the figure number 1.

![Program flowchart]

**Figure 1:** Program flowchart

### 2.1 Collection of the Web Server Parameters

One of the first things that the program does is finding out the cipher suite. The cipher suite defines the algorithms that ensure confidentiality, integrity and parameters that determine the key lengths for these algorithms. The values in the cipher suite are separated with the character “_”. Cipher suite is acquired by the program immediately after the creation of the connection with the server and is saved for future processing. Cipher suite example: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

The next object that the program loads is the certificate used for the verification of the web server identity. This certificate contains the server public key with which the client is able to verify information that is digitally signed with the servers private key. The certificate also contains other information such as the algorithm which is used to create the digital signature. The program first loads the certificate in DER binary format and then converts it into a python object. To find out which versions of the TLS protocol the server supports, it creates a connection on the version of the protocol it is currently testing. If the server doesn’t establish a connection, the server doesn’t support that version. If a connection is established, the version is added to a python list and then is sent to the program output.

### 2.2 Parameter Sorting

The next action the program takes is parameter sorting or parameter extraction from the collected objects. Parameters from the certificate can be extracted easily, since the certificate is saved in a simple python object where parameters are saved as properties of the object. Unlike the certificate, the cipher suite is collected in a text format and the parameters positions are not strictly given. For example in one cipher suite the symmetric encryption algorithm is present on the 4th position (position is defined as parts divided with the character “_”). But in a different cipher suite it can be present on the 3rd position, if the cipher suite doesn’t define a public key algorithm. That’s why the program sorts the parameters by dividing them into a list with the “_” character. Then it loops through the list and compares the values with a predefined json file which contains another list of all possible parameter values for each parameter category. When the correct category of the parameter is found the value is saved together with the category into a python object.

### 2.3 Parameter Rating

When all the parameters are sorted and saved in python objects, the program rates them by their security. Of course only the parameters, that can be rated are rated, for example a certificate version
can’t be rated. To rate the parameters, another json file is used, which contains 4 levels of security for each parameter category. Every level contains a list of parameter values. When the program rates a parameter it loads it and loops through all of the levels, until it finds a match with value of the rated parameter and one of the security levels. The parameter values of the security levels are created with the inspiration from the NIST recommendations, more precisely NIST SP 800-52 [4] and NIST SP 800-131A [5]. The number 1 denotes the highest security and number 4 the lowest.

Numeric parameters such as the key lengths, are rated similarly. They too have 4 security levels defined in the json file. Since these parameters can’t be rated alone and depend on the algorithm for which they determine the length of the key, each numeric parameter has an assigned algorithm. Because of that, the security levels in the json file are defined differently. One value is defined for each key size and algorithm pair, which consists of the algorithm name, a sign of comparison and the key size. For example the value for the algorithm RSA with a key size of 2048 bites would like this: “1: RSA,>=2048”. This means, that the algorithm RSA with a key size bigger or equal to 2048 has the security level of 1. After the acquisition of the security level of the parameter the program saves this value into a dictionary, where the key is the key size and value is the security level.

2.4 Vulnerability Analysis

Finally the program analyzes the vulnerabilities of the web server. The implementation of the vulnerability tests depend on the nature of the vulnerability. Program supports several tests which are performed using custom implementations. Every test tests the presence of the vulnerability. If more vulnerability tests are ran in one instance, every test is ran in one thread, to speed up the testing process. Vulnerability tests that the program supports are: Heartbleed, ZombiePOODLE/GOLDENDOODLE, attacks using renegotiation and CCS (Change Cipher Spec) Injection. These tests were chosen because their exploitation results in the voidance of TLS protocol security. Heartbleed allows for the collection of the web server private keys. POODLE attacks and CCS injection void the confidentiality ensured by the symmetric cryptography. Renegotiation attack compromises the integrity and in some cases even the confidentiality of the messages being sent [2].

3 Tool Usage

The program is ran as a python script and defines some optional arguments. It requires one mandatory argument, that is -u, which is used to enter the url of the web server to scan. Vulnerability tests can be triggered using the the test number and the -t argument. Some other arguments and test numbers can be displayed using the -h argument. In the standard mode the program output is specified as a text output (stdout) to the environment in which the program is running. However, if the -j argument is specified together with output file name, the program outputs all information about the web server to the given file.

In picture number 2 a shortened program output is shown. The output consists of 4 sections and each part has its own name and its elements are intended. The first sections holds the cryptographic parameters that are rated using numbers, which denote the security level. Parameters of the first section are extracted from the cipher suite and from the the certificate. The final rating of the parameters is at the end of this section also.

The next section lists the TLS protocols that the web server supports and their ratings same as in the first section. The third section consists of the un-ratable certificate parameters, for example the serial number. In the last section the specific implementations of the web servers that are used by the server are listed. The program will also list the the results of the vulnerability tests after they are implemented.
**Cryptographic parameters:**
- Key exchange algorithm: RSA->2
- Symmetric encryption algorithm: AES->1
- Symmetric encryption algorithm length: 128->1
- Symmetric encryption algorithm block mode: CBC->2
- Hash function: SHA->2
- Public key algorithm: RSA->1
- Public key length: 2048->1
- rating: 2

**Certificate information:**
- Valid from: 2020-04-21
- Valid until: 2021-04-26
- Subject: localityName=Brno
- commonName=www.vutbr.cz
- Issuer: commonName=TERENA SSL CA 3
- Alternative names: vut.cz

**Protocol support:**
- TLSv1->2
- TLSv1.2->1
- rating: 2

**Web server versions:**
- http_header: Apache
- nmap: Apache httpd

---

**4 CONCLUSION**

During the development of the program, its functionality was tested on Linux distributions such as Linux, Mint and Debian using Docker containers. The program was working correctly in these environments. Next the program was tested on WSL (Windows Subsystem for Linux) but during the analyzation of the web server vutbr.cz an error occurred. The web server vutbr.cz supports TLS versions from 1.0 to 1.2, but the program listed only version TLS 1.2. This error happened because the implementation of the OpenSSL library was compiled only with the support for protocol TLS 1.2 and higher. Since the Python interpreter, which interprets the program uses the same instance of the OpenSSL library, it was unable to create a connection even if the server supported it. However, the program works correctly because if the operating system of the user has the OpenSSL library compiled such that it doesn’t support versions 1.0 and 1.1 the attacker is unable to force the user to use a lower version protocol. There are many other TLS scanning tools freely available on the internet, such as the website ssllabs.com. However, these tools run on 3rd party servers. This means if many users utilize this service the scanning may be slowed down. The advantage of this tool is that the user can run the program on his own computer or run multiple instances of the program to scan multiple servers parallely using his own computing power.

**REFERENCES**


