Web Authentication

Sigurd Uhre

April 2023

Contents

1 Introduction

The objective in this assignment wis to implement a web application that displays a greeting message to authenticated users. The user credentials provided must persist in a file and the credentials must be protected against network-based replay attacks and network snooping. The password must also resilient to dictionary attacks.

2 Background Theory

2.1 Dictionary attack

The dictionary attack is based on that a short, easy-to remember or common password have a defined set of possibilities. Computer attackers know the most common passwords and keep an dictionary of them, thereby the name dictionary attack. The English language have less than 50,000 common words, where the 1,000 most frequent used words make up around 70 percent of all written and spoken language. In addition, if the attackers also gather other information such as the 1,000 most common human first names, 1,000 typical pet names, 10,000 common last names, and all possible birthdays in the last 90 years. The attacker can put this information together into a directory of all the common passwords and have a file that has fewer than 100,000 entries. [1]

The attacker is armed with this dictionary of common password an they can now use it to attack the password protection for users of web services. With the speed of the modern computer the attacker can attack a password-protected object and brake the protection in just a few seconds [1]. A modern computer can test one password every milliseconds, meaning that it will take 1 minute and 40 seconds (100 seconds) to test the complete dictionary list of 100,000 entries.

2.2 Network-based replay attacks and session hijacking

Session hijacking in an category of attacks that takes over a TCP or HTTP session in an attack. These attacks require that the attacker intercepts communication between a web server and web client, in addition the attack also require the atacker to impersonate whaterver measures are being used to maintain the HTTP session [2].

Servers also need to defend against possible replay attacks. Replay attacks are based on reusing old credentials to performer false authentication or authorization [2]. In a case such as this, an attacker would use an old, previously valid session token to perform an attempted HTTP session hijacking attack. By incorporating random number into both client and server side tokens, combined with frequently changing session tokens, a server can protect against such attacks [2].

2.3 Network snooping

Also known as network sniffing, this kind of attack is the practice of analyzing and intercepting network traffic for the purpose gathering information from monitoring network traffic. Trough specialized software and hardware, packages that are sent over the network can be captured, allowing the data inside to be examined by an unauthorized attacker. To protect against network snooping, technologies (protocols) such as SSL/TLS or VPNs are used to secure network traffic.

2.4 Hashing and salt generation

In this assignment we use bcrypt algorithm that generates a one-way function to the password, meaning that there is no way to reverse the process back to the original password. This results on a fixed-length string of characters known as the hash-value of the the original password. Instead of the password being stores in the database, visible for authorized users, the hash value representing the password is stored in the database. Salt generation is generally happens before the hashing when the user creates a password, the salt is a random string of characters that is appended to the password before its hashed. The reason for using the salt method is to prevent attackers from using recomputed hash tables to solve the password.

After the salt is generated and added to the password, the resulting string is then hashed. Hashing is often referred to as the process of converting any data of variable size into a fixed-size output. The process itself is done using a mathematical function called a hash function. When a user creates a password, the random salt-value is generated and combined with the password and the resulting string is then hatched and stored in a database. When the user logs in at a later time, the same salt-value is retrieved from the database and and then combined with the password during login. The resulting string is then hashed again, and the resulting string is compared to with all the stored hash in the database. Hashing and slat generation protects user credentials against dictionary attacks, rainbow table attacks, MITMA-attacks, brute force attacks and insider attacks.

2.5 Secure transport layer security (TLS/SSL)

HTTP request and responses communicate via TCP over port 80. The Standard HTTP protocol does not provide any means of encrypting its data, meaning that content of the communication between the two sides is sent "in the open". The lack of encryption in this communication makes the HTTP protocol vulnerable to attackers who can interact with a packet that is sent between a web page and a web browser. An attacker in an man-in-the-middle attack would gain full access to all the information that the user of the web site was transmitting, including the password and username, as we will discuss in section 4 [3].

To solve the problems of HTTP, an alternative protocol is available called HTTPS (Hypertext transfer protocol over secure socket layer). The difference between the two protocols is that HTTPS hav an additional security layer called SLL (secure socket layer), or the newer version TLS (transport layer security). These technologies rely on the usage of certificates as an validation. To ensure security, it is necessary to authenticate the servers identity and establish a secure and encrypted communication channel between the web server and the web browser [3]. There are three key benefits that SSL or TCP brings to the HTTPS protocol, being confidentiality, integrity and authentication. SSL encryption provides confidentiality by ensuring that data in transit between the server and client is protected against eavesdropping an third parties attacks. It ensures integrity by not allowing any tampering of data in ttransmission between client and server. SSL also provides authentication of the server to the client, which helps to prevent phishing and other attacks that rely on impersonating a legitimate website.

When a connection between a web browser and a server starts, initially, the web browser sends a request to establish a HTTPS session with a "client hello" message to the server. The message presents a catalogue of cryptographic ciphers and hash functions that are supported by the client along with other information. The server responds by a "Server hello" message, including a SSL/TLS version, choosing the strongest combination of the encryption algorithm and informs the browser, before sending back the certificate which contains the servers public encrypted key. The next step is for the browser to verify the authenticity of the certificate. Lastly, the browser employs the public key of the server to encrypt a randomly generated number that can solely be decrypted with the server's private key [3].

3 Design and implementation

3.1 Protect against dictionary attacks

To protect against dictionary attacks the implementation of flask limither has been done. This function is called before every request is processed, it returns the IP address of the client and the number of requests that have been made from that IP address in the last hour, minute or second, depending on what is desired. In this implementation, we have a limit of three attempts per minute and an overall limit of 10 attempts per hour. As we discussed in TB, a dictionary can check for 100,000 passwords in 1 minute and 40 seconds. With this restriction implemented, an dictionary attack can not check more than 10 passwords in one hour, thereby eliminating one of the key functions of the dictionary attack which is the testing of many passwords in a very short time.

3.2 Hashed password

In this implementation the password are encoded and stored using bcrypt hash function. The password is stored in a file with one username and hashed password per line, representing the credentials for one specific user. The user provided credentials registered during registration will be stored here. The password will first have a salt generated and the resulting string will then be hashed and stored in the password file along with the username. The username is not hashed and is visible in the "passwords" file while the password itself will be hashed and located next to the corresponding username. It is important to specify that salt generation and hashing first take place at the server. These methods therefore provide no protection in the transmission between the browser and the server when a user first enters his user credentials, an encrypted network communication must be established to ensure this. However, the hashed password add additional security in the way that it is difficult for an attacker to access and obtain the original password, even if the attacker have access to the password file.

3.3 Encrypted network communication

To protect the stored credentials against network based replay attacks and network snooping we have implemented OpenSSL and Let's Encrypt. First we created a self-signed SSL certificate which generate an certificate and a encrypted key to the selected files (cert.pem and key.pem). In the main function of the application the path to this key and certificate is established before OpenSSL is used to load the certificate and private key on the server. An SSLContext object is then created where the desired protocol (HTTPS) is specified along with the certificate and private key.

3.4 Server API

The server API returns HTML pages located in templates, starting with the home "/" endpoint. The homepage endpoint runs a if-check to see if the user i logged inn or not, displaying the username of the logged-in user if the check returns positive. The GET HTTP request method display the HTML files for login and registration, displaying interfaces for registration or login of username and password. The POST HTTP request method for /register gets the username and password from the registration form, the password is then salted before the resulting string is hashed and written to the passwords -file together with the username.

The POST HTTP request for the /login page function somewhat the same as the registration above. In login, the input username and password is taken and a check is done to compare the input against a username and password in the password file. Here an if-check is done to compare the username input against the "plain-text" username stored. We use bcrypt to test if the password input, which is salted and hashed, returns a true match between the hashed passwords in the database with the corresponding username. If the credentials match a user in the database, the endpoint will be directed to the /loggedin endpoint, displaying the name of the user.

The server identify authenticated users after login by using the session cookie. The username is displayed along with a welcoming message when the user successfully is logged inn. The session can be explained as the server-side storage of information that is desired to persist throughout the session between the server and browser. Session id is a unique identifier that is stored on the client browser side, and sent to the server every time that specific user (with a unique session id) makes an HTTP(s) request. The server web application pairs the session id sent from the browser with its internal database and retrieved the stored variables for it to be used in the requested page [4].

4 Evaluation / Discussions

4.1 HTTP vs HTTPS

Wireshark was used to test for differences in HTTP and HTTPS. Before implementing the HTTPS method the we ran the server on HTTP to see if it was possible to retrieve some information from the packets

that were sent between the browser and the server. Plain text information is sent from the browser to the server when we use HTTP, making it possible for an attacker to retrieve critical user cridentials and preform network snooping or replay attacks among other. The figure below shows a wireshark window that have located a packet counting the username and password in plain text. The protocols are easy to detect since they are labeled as HTTP protocols.

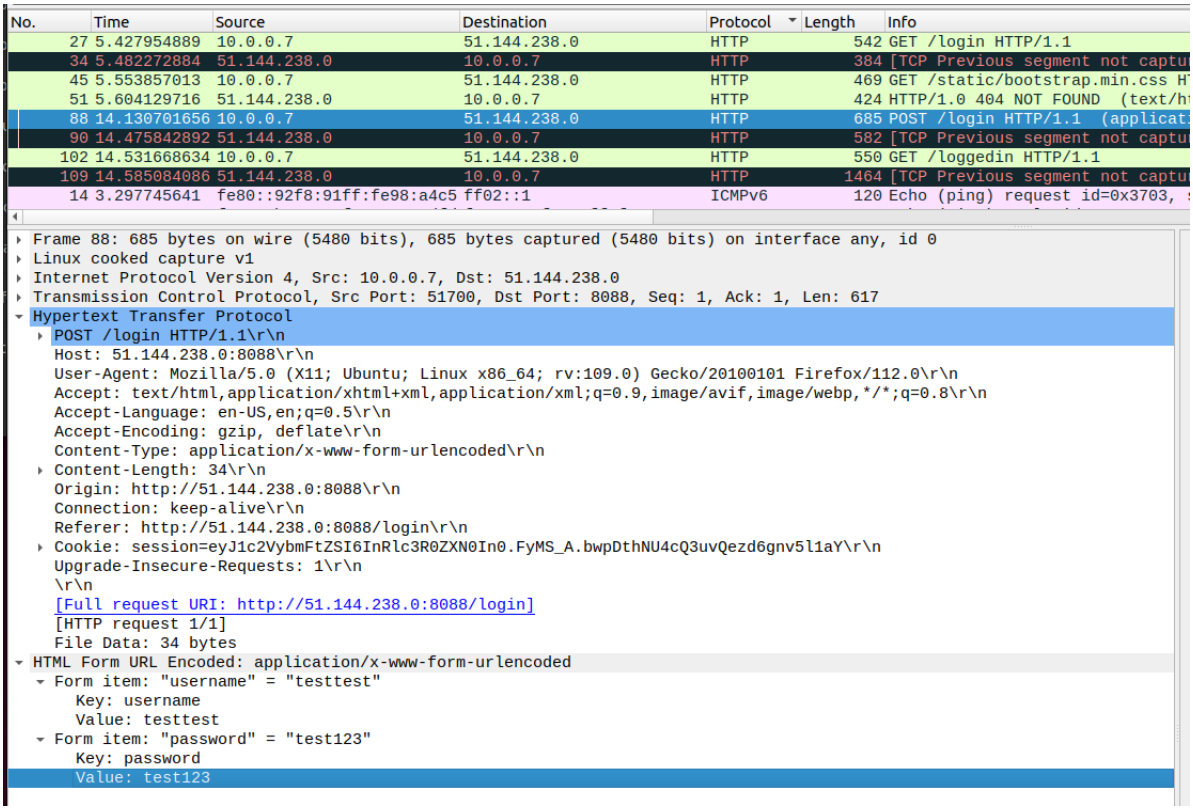

Figure 1: Wireshark window showing plain-text password (test123) and username (testtest) in a package.

The HTTPS package transactions looks different compared to the HTTP transaction. It uses transmission control protocol TCP for ensuring reliable and ordered delivery of data between the browser and server. HTTPS uses SSL or TLS encryption to protect the data against network snooping and from an attacker to recive critical user credentials from it. As suspected, we did not find any un-encrypted user credentials in the packages sent in transmission, below is a wireshark image showing how TCP-protocol packages look like.

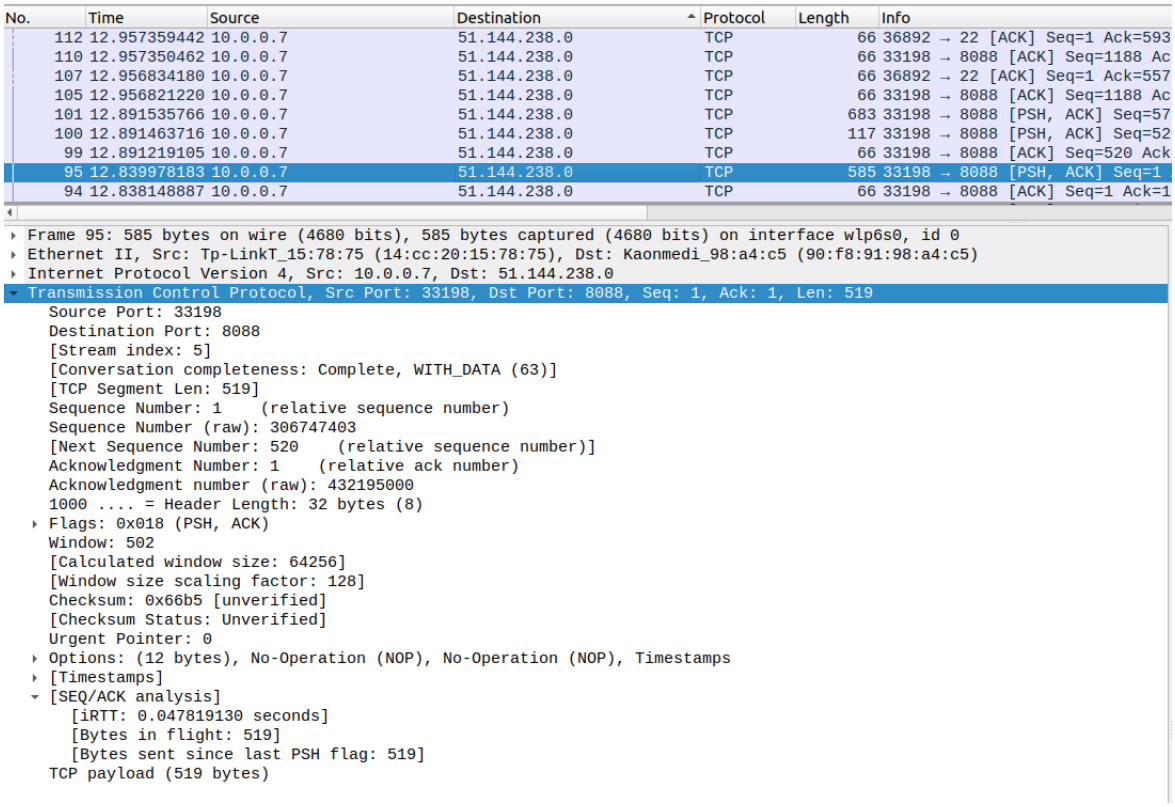

Figure 2: Wireshark window showing https transfer between server and browser.

4.2 Dictionary attacks

As explained earlier, a flask limiter was added to prevent an attacker from preforming dictionary attacks efficient. In addition to this implementation, specific criteria could have been given for how long the password should be, whether it must contain lowercase or uppercase letters and whether it should contain special characters. These criteria have not been added but would contribute an extra layer of security against dictionary attacks, since the most used passwords do not contain random special characters, upper and lower case letters or are over a certain length.

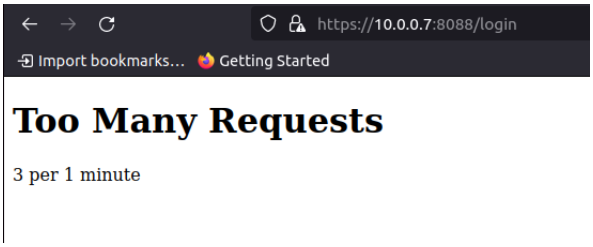

Figure 3: One minute timeout from three password attempts

4.3 Session cookies

Below, the web application (server) is deployed on the virtual machine provided to each student (see URL). The figure illustrates the greeting message to authenticated users that have completed a registration and successfully logged inn afterwards. This is done by using session cookies and session id to display the desired user credentials of that user.

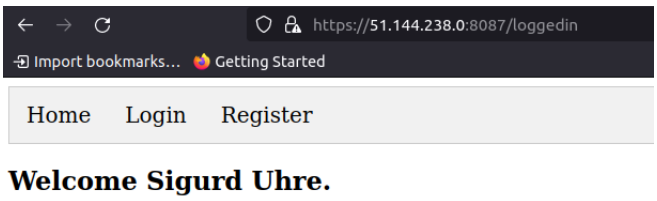

Figure 4: Welcome message to authenticated user.

5 Conclusions

The objective in this assignment was to implement a web application that displays a greeting message to authenticated users. The user credentials provided must persist in a file and the credentials must be protected against network-based replay attacks and network snooping. The password must also resilient to dictonary attacks.

The web application endpoint infrastructure works as indented and the html-templates correspond with the navigation of the frontend. During registrations the password and username (user credentials) are sent and kept in file. The password undergoes a salt generation before the resulting string is hashed and stored in the file. To set up encrypted network communication and make the application resilient to network-based replay attacks and network snooping, HTTPS with SSL encryption is used. The important difference between HTTP and HTTPS is illustrated and discussed in this report. Flask limiter is implemented to make the application resilient to dictionary attacks, further implementations to strengthen this defence would be to make some requirements about what the password must contain (such as e.g. special characters). The web application displays a greeting message to authenticated users, the server does this by storing the users username in a session cookie. The session id is used to identify the user and display the name while the user credentials are kept safe in a file.

6 Reference:

1. Goodrich MT, Tamassia R. Web Security. In: Introduction to computer security. Upper Saddle River: Pearson; 2014. p. 41.

2. Goodrich MT, Tamassia R. Web Security. In: Introduction to computer security. Upper Saddle River: Pearson; 2014. p. 347–8.

3. Goodrich MT, Tamassia R. Web Security. In: Introduction to computer security. Upper Saddle River: Pearson; 2014. p. 333-5.

4. Patel S. How to use cookies and session in python web scraping - worthweb [Internet]. Worth Web Scraping. 2021 [cited 2023Apr25]. Available from: https://www.worthwebscraping.com/how-to-use-cookies-and-session-in-python-web-scraping/