**WEB SERVICES**

420-511-VA

#### **ASSIGNMENT-2-**

In this assignment students apply their acquired knowledge about HTTP, client / web server communication, hashing, and encryption.

When done, you need to demo the assignment and discuss it with the teacher.

PART-1

In this part we will simulate an HTTP Request and HTTP Response between a web browser and web server.

1. Using the programs implemented in LAB-1, we will consider program-1 as a web browser and program-2 as a web server. We will send an HTTP Request from program-1 to Program-2 and an HTTP Response from program-2 to program-1.
   * Implement the below code where necessary to achieve our goal.
   * Code 1:

|  |
| --- |
| $request = "GET /page.html HTTP/1.1\r\n";  $request .= "Host: localhost\r\n";  $request .= "Connection: Close\r\n\r\n"; |

* + Code remarks:
    - Note that the \r\n are for carriage return and line feed (CRLF)
    - Note the double \r\n at the end of "Connection: Close\r\n\r\n", since after the header there should be an empty line before a possible body as per the HTTP specifications: <https://datatracker.ietf.org/doc/html/rfc2616#section-4.1>
    - Note that we broke down the response value into multiple code lines for clarity, but it is effectively equivalent to: $request = "GET /page.html HTTP/1.1\r\nHost: localhost\r\nConnection: Close\r\n\r\n";
  + Code 2, write a custom response:

|  |
| --- |
| $response = "Hello";  $response .= "World";  $response .= "Goodbye"; |

* + Note that we could make program-2 loops and waits for other requests and sends corresponding responses. You do not need to implement this.
  + You could test the program-1 code with a program-2 running on a colleagues machine in the LAB, instead of your own program-2.

1. Communicating with Apache.
   * If we want program-1 to send the HTTP Request to a real web server, Apache what should be done? modify the code to do that. **Change the port number to 80.**
   * Create the html page page.html, and put in it a simple message, otherwise you will get the html for a 404-error page.
   * Test the code and write the response from Apache below.

|  |
| --- |
| HTTP/1.1 200 OK Date: Wed, 16 Feb 2022 04:21:06 GMT Server: Apache/2.4.51 (Win64) OpenSSL/1.1.1l PHP/8.0.12 Last-Modified: Wed, 16 Feb 2022 04:19:05 GMT ETag: "8b-5d81af34320ad" Accept-Ranges: bytes Content-Length: 139 Connection: close Content-Type: text/html |

1. What happens if we remove one of the \n from the end of the request?

|  |
| --- |
| This causes the web browser to load infinitely. In result, no change in the browser will appear. |

1. What happens if we remove "Connection: Close\r\n\r\n" from the end of the request?

|  |
| --- |
| This is the same result as question 3. The browser loads an infinite amount of times, leading to no change in the browser. |

1. What happens if, using the original request, we replace GET with DUMMY and make the request to Apache? What are the status codes and messages?

|  |
| --- |
| Reading response: HTTP/1.1 501 Not Implemented Date: Wed, 16 Feb 2022 04:29:08 GMT Server: Apache/2.4.51 (Win64) OpenSSL/1.1.1l PHP/8.0.12 Allow: HEAD,GET,POST,OPTIONS,TRACE Content-Length: 304 Connection: close Content-Type: text/html; charset=iso-8859-1  Not Implemented  DUMMY not supported for current URL.  The page doesn’t get loaded. The webpage is not returned. The status code is 501, followed by the message above. |

PART-2

In this part students will modify the communication between program-1 and program-2 in order to use hashing for signing the data and encryption.

Signing the data using a Hash

1. Considering program-1 and program2, and the data in the JSON format:

'{"name":"John", "age":30}'

that needs to be sent from program-1 to program-2. We will not use a full HTTP Request and Response for simplicity.

Use the PHP hash\_hmac() function, with the sha256 Algorithm, to Implement the following:

* Hash the data and include a signature with the data before sending it to program-2
* When received by program-2 verify the signature and send back to program-1 a feedback, or acknowledgement, with an appropriate message stating whether the data is valid or invalid.

|  |
| --- |
| Submit the code files. |

Encryption and Signature

1. Considering program-1 and program2, and the data in the JSON format:

'{"name":"John", "age":30}'

that needs to be sent from program-1 to program-2.

Use the PHP hashing and encryption functions to Implement the following:

* Hash the data and include a signature with the data before sending it to program-2.
* Encrypt the data before sending it to program-2, use a seed.
* When received by program-2, decrypt the data.
* Verify the signature and send back to program-1 a feedback, or acknowledgement, with an appropriate message stating whether the data is valid or invalid, and the original decrypted data.
* In order to implement the above go through the following example from PHP Manual line by line and explain each line:
  + **EXAMPLE #2**: https://www.php.net/manual/en/function.openssl-encrypt.php

|  |
| --- |
| * <?php * // The plain text to encrypt * $plaintext = "message to be encrypted"; * // The length of the A non-NULL Initialization Vector. * // The openssl\_cipher\_iv\_length() function returns the length of the IV on success and FALSE on error. * // It returns an E\_WARNING level error when the cipher alorithm is unknown * // The parameter that it accepts is the cipher algorithm, which in this case is is AES-128-CBC. * // A cipher method is a way of hiding characters with an encryption algorithm, by replacing or substituting characters with a different character according to the alorithm. * $ivlen = openssl\_cipher\_iv\_length($cipher = "AES-128-CBC"); * // The openssl\_random\_pseudo\_bytes generates a string of pseudo-random bytes according to the length parameter, which in this case is the ivlength, created from before. * $iv = openssl\_random\_pseudo\_bytes($ivlen); * // The openssl\_encrypt function encrypts the plaintext using the method, key, and returns a raw or base64 encoded string. * // The parameter that it accepts is the plaintext, which in this case is the message to be encrypted. * // The second parameter is the cipher method, which in this case is AES-128-CBC. * // The third parameter is the encryption key, which in this case is the passphrase. * // The fourth parameter is the options, which in this case is OPENSSL\_RAW\_DATA. * // OPENSSL\_RAW\_DATA is a constant that is used to specify that the data should be returned as a raw string, not base64 (which is by default). * // The fifth parameter is the Initialization Vector, which in this case is the variable created previously. * $ciphertext\_raw = openssl\_encrypt($plaintext, $cipher, $key, $options = OPENSSL\_RAW\_DATA, $iv); * // hash\_hmac() function returns a HMAC (hash-based message authentication code) of the given data using the given hash algorithm. * // According to PHP doc, it returns "lowercase hexits unless binary is set to true in which case the raw binary representation of the message digest is returned." * // The first parameter is the hash algorithm, which in this case is sha256. * // The second parameter is the data, which in this case is the ciphertext. * // The third parameter is the key, which is the shared secret key used for generating the HMAC variant * // The fourth parameter is an options parameter, which if set true, returns raw binary data. * $hmac = hash\_hmac('sha256', $ciphertext\_raw, $key, $as\_binary = true); * // Base64 encode the make binary data survive transport through transport layers that are not 8-bit clean * // If this step is skipped, the data set will involve ? characters * $ciphertext = base64\_encode($iv . $hmac . $ciphertext\_raw); * //decrypt later.... * $c = base64\_decode($ciphertext); * // Same thing as line 10. * $ivlen = openssl\_cipher\_iv\_length($cipher = "AES-128-CBC"); * // Extract the IV from the first 16 bytes of the ciphertext * $iv = substr($c, 0, $ivlen); * // Extract the HMAC from the 16 bytes after the IV * $hmac = substr($c, $ivlen, $sha2len = 32); * // Extract the ciphertext from the 16 bytes after the IV and the HMAC * $ciphertext\_raw = substr($c, $ivlen + $sha2len); * // Decrypt the ciphertext using the same key and the same method * $original\_plaintext = openssl\_decrypt($ciphertext\_raw, $cipher, $key, $options = OPENSSL\_RAW\_DATA, $iv); * // Hashing the original plaintext. Method explained above. * $calcmac = hash\_hmac('sha256', $ciphertext\_raw, $key, $as\_binary = true); * // Comparing the two hashes together. If both are equal, then the text was not manipulated by a man in the middle * if (hash\_equals($hmac, $calcmac)) // timing attack safe comparison * { * echo $original\_plaintext . "\n"; * } |

|  |
| --- |
| Submit the code files. |