Student's name: Matheus Henrique Schaly

Professor's name: Carlisle Adams

Course: CSI 4139 Design of Secure Computer Systems

Due date: 6th October, 2019

Report #2

Program

The programming was written using Javascript programming language along with Node.js framework for writing server-side JavaScript applications (backend). As for the programming environment, the integrated development environment used was the Visual Studio Code. The communication environment used Nexmo SMS API, which enables you to send and receive text messages to and from users worldwide.

The created website enabled the user to log in using a 2-factor authentication both by using a keyboard and a cell phone number. The user had to login in using his/her username and click the "return" button. After that, the program consulted an internal file that kept the cell phone associated with the username. In case the username matched any of the usernames in the file, a SMS was sent to the associated cell phone. On the other hand, if the username was not linked to any cell phone number, the SMS was not sent. Once the SMS was received, the user could enter it along with a password. If both were correct, the website returned a string. The website validated it and, if correct, it returned two string which the user could input to the local application. Finally, the website had to validate these two strings and return success or fail accordingly.

Detailed Analysis of the Implementation

The protocol used to create the secure communication environment includes the Diffie-Hellman (DH) method. Such method is used to securely exchanging cryptographic keys over a public channel. It allows two parties that have no prior knowledge of each other to jointly stablish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher [1].

It is recommended that the order, *p*, of the Diffie-Hellman group should be at least 2048 bits long. The greater the order of *p*, the more secure is your system. However, the greater the order of *p*, the slower your system is [1]. To better decide how many bits are going to be used for *p*, we first need to define a thread model. Let's suppose that the attackers are not experts in hacking, that they don't have governmental resources and we don't have any extremely valued information in our system. Moreover, we would prefer that the user has a better usability and efficiency, rather than an extremely secure environment. Therefore, 2048 bits are good enough security for our system.

A library can be used for encryption and MAC, the bits used to encrypt and Message Authentication Code (MAC) the message can come from the output of the Diffie-Hellman. For the encryption (confidentiality), the Advanced Encryption Standard (AES) algorithm can be used. Such algorithm is a worldwide symmetric-key algorithm, which fits our implementation. There are three different key lengths for AES: 128, 192 and 256 bits [2]. As already mentioned, considering our thread model, we can choose the 128 as it is faster than the others. For the authenticity, the keyed-hash message authentication code (HMAC) can be used. HMAC is a specific type of MAC involving cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authenticity of a message, as with any MAC. Any cryptographic hash function, such as SHA-256 or SHA-3, may be used

in calculation of HMAC; the resulting MAC algorithm is termed HMAC-X, where X is the hash function used [3]. For the hash function, the SHA-256 can be selected. This SHA (Secure Hash Algorithm) algorithm generates an almost-unique, fixed size 256-bit hash. SHA-256 is one of the successor hash functions of SHA-1, and is one of the strongest hash functions available [4]. Therefore, the HMAC-SHA256 also fits our implementation. To get the bits for the encryption key and the HMAC key, let K be the output of the algorithm where $K = g^{ab} \bmod p$. The K is a number between 0 and $p-1$. As we've chosen HMAC-SHA256, we're going to pick the first 256 bits of K to use in our HMAC-SHA256 algorithm. Let k_1 be those 256 bits. Moreover, as we've chose the AES with 128 bits, we are going to pick for the AES the first 128 bits after the 256 bits already selected to the HMAC-SHA256. Let k_2 be those 128 bits. Finally, we can use k_1 and k_2 to execute E_{k2} (m' || HMAC-SHA256$_{k1}$ (m', SHA256)). With that, the protocol is completed and both parties can successfully exchange data with authenticity, integrity and confidentiality.

Protocol Disadvantages

Regarding Diffie-Hellman protocol limitations, we can say that the biggest weakness is that it doesn't (by itself) stablish the identity of the other party, making it susceptible to a man-in-the-middle attack, that is, the Logjam attack. The Logjam attack a man-in-the-middle attack, which allows an attacker to force the negotiation of 512-bit-long keys in order to break encrypted communications. Furthermore, the current best technique for attacking Diffie-Hellman relies on compromising one of the private keys by computing the discrete log of the corresponding public value in the DH group. The parameters to generate the group are previously shared in clear as part of the FCC domain parameters. However, an adversary who performs a large pre-computation for prime *p* can then quickly calculate parameter discrete logs in that group, amortizing the cost over all targets that share this parameter. This attack takes advantage of the fact that an adversary can perform a single enormous computation to crack a particular prime and then easily break any individual connection that uses that prime. This can be done in advance, which leave only the second phase to be done on-the-fly for any particular connection. Websites that use one of a few commonly shared 1024-bit Diffie-Hellman groups may be susceptible to passive eavesdropping from an attacker with nation-state-level resources [5].

References

[1] Wikipedia. (2019). *Diffie–Hellman key exchange*. [online] Available at: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange [Accessed 6 Oct. 2019].

[2] Wikipedia. (2019). *Advanced Encryption Standard*. [online] Available at: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard [Accessed 6 Oct. 2019].

[3] Wikipedia. (2019). *HMAC*. [online] Available at: https://en.wikipedia.org/wiki/HMAC [Accessed 6 Oct. 2019].

[4] Xorbin. (2019). *SHA-256 hash calculator*. [online] Available at: https://www.xorbin.com/tools/sha256-hash-calculator [Accessed 6 Oct. 2019].

[5] Cert.europa.eu. (2019). [online] Available at: https://cert.europa.eu/static/WhitePapers/CERT-EU-SWP_16-002_Weaknesses%20in%20Diffie-Hellman%20Key%20v1_0.pdf [Accessed 6 Oct. 2019].