Indian Journal of Science and Technology, *Vol 8(S9), 285–291, May 2015*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Combining TCP and UDP for Secure Data Transfer

K. Rajkumar* and P. Swaminathan

School of Computing, SASTRA University, Thanjavur - 613401, Tamil Nadu, India;
rajkumar@cse.sastra.edu, deanpsw@sastra.edu

Abstract

Objective of this paper is to enchance the security while tranfering data between two host through internet. It is very obvious that technology is evolving. As technology evolves the threat for user's data also increases. Enhancement in data security has become a basic necessity. This paper also works towards enhancing the security of data through 6 layers. The data that have to be sent to the receiver will pass 6 layers of security on the sender's side. The first layer comprises of any encryption algorithms. The second layer involves compression techniques. The third layer involves a new technique of key generation. In the fourth layer the data will be traversed in any one of the 9 existing patterns. Fifth layer is to append the data with another text, audio, video or image file. Final layer combines both the two basic networking protocols, Transmission Control Protocol and User Datagram Protocol. The byte array of this appended file will be sent through the User Datagram Protocol. The Generated key will be sent through the transmission Control Protocol. Transmitting data and the key in different protocols improves the security to a greater extent. The working of all these 6 layers will be discussed in this paper. Improvements will be made refer to increasing the performance and security without affecting the complexity level on client and server side. Region based segmentation can be used to hide the data in a particular part of the image. The implementation of these improvements is in process, to enhance the security of the data.

Keywords: Data Security, Data Transfer, Layer, TCP, UDP

1. Introduction

Data can be transmitted through various protocols. Some are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Real-time Transmission Protocol (RTP) etc. This paper focuses on TCP and UDP. TCP is meant for its reliability and UDP is meant for its fastness. To scrutinize them, TCP doesn't impart fastness and UDP doesn't implement security and doesn't guarantee the delivery of packets. The idea implemented in this paper is to combine both the protocols to convey security of the data to be transferred. This combination also increases the performance of the entire system. TCP can also be used as a base connection as explained in "Designing and Comparing Centralized TCP File Server with TCP and UDP based Enhanced Unicasted and Multicasted Multimedia File Servers" by Arun Singh, Ajay K. Sharma and Ashish Kumar. As UDP can transmit only a theoretical limit of 65,535 bytes, the UDP packets have to be segmented into packets if its size exceeds the default limit. By utilizing TCP as base connection, it is considered that the base connection will remain till the entire process gets completed, which makes both the system active till the end of the process so the UDP packets can be received in a faster manner. With this idea as basement, we proceed this paper to implement still more layers to improve the security and maintain the performance level.

2. Experiment

The process starts by selecting the file, or by entering the data that have to be transmitted to the other end. This data has to be encrypted and compressed, which forms the first 2 layers of security to be implemented.

*Author for correspondence

2.1 Encryption and Decryption

There exist a huge number of encryption algorithms. Any preferred algorithm can be used. The implementation of this paper has been done with Affine Cipher for encryption and decryption. As Affine Cipher is a mono alphabetic substitution cipher, which encrypts each alphabet of data to a corresponding alphabet. Affine cipher requires 2 keys for its encryption and decryption process. The formula used for affine cipher is

$E(x) = (ax+b) \bmod m$

Here the 2 keys to be entered by the user is a and b. The key a should be co-prime for the length of the entire data. So if the sender enters a non co-prime value either the sender can be notified to enter a co-prime value if not the code can generate the nearest co-prime value automatically so that the sender also will be unaware of the key used which brings additional security. The index of the plain text element in the character set is represented as x. The Total number of alphabets (i.e) 26 is the value to be given to m. So the value obtained for (ax+b) should be used to find the modulus by 26. For example, the letter a can be encrypted using affine cipher as given below,

x - 1 (as a is the first alphabet)

Let us consider the keys a and b to be 1 and 7 respectively so,

$(ax+b) = (1+7) \bmod 26 = 8$

The alphabet a will be substituted by the alphabet h. Similarly, each and every character of the message will be encrypted to its corresponding cryptographic character. The encrypted string will be then passed to the compression algorithm.

Encryption will be the first step in server side, but decryption will be the last step on the client side. Decryption is done based on the below given formula

$D(x) = a\text{-}1\ (x-b) \bmod m$

The corresponding characters in the decrypted values obtained will append together to get the original message. At the end of the decryption phase in client side the original message transmitted will be obtained.

2.2 Compression and Decompression

The compression technique adopted in this paper reduces the length of the text file to exactly half of the original length. Compression is based on the average value concept. The ASCII values of all characters are obtained, then each obtained value will be subtracted from a constant value 97. Average of first 2 values will be calculated and the corresponding character will be replaced from the character set. Similarly, all characters will be replaced by the character corresponding to its average value. All the substituted characters are appended together to get the final compressed text. So, the compressed text obtain will be half in number of the original text. If the length of the original text is odd then the last left character can be appended directly with the compressed string. For example, consider a word "compress" which will be compressed to "iols". The complete flow of the process is defined with the same example.

Step 1. The ASCII value of each alphabet in the word compress

c=99 o=111 m=109 p=112 r=114 e=101 s=115 s=115

Step 2. Subtract the ASCII value with a constant 97. So the obtained values will be

2,4,12,15,17,4,18,18

Step 3. Calculate the average of the neighboring 2 alphabets. If the length is an odd value then use the last alphabet directly.

2+14 = 16/2=8

12+15 = ceil (27/2) = 14

17+4 = ceil (21/2) = 11

18+18=36/2 = 18

Step 4. Substitute the character corresponding to the obtained value

I – 8, o – 14, l – 11 and s - 18

So the compressed text is "iols".

Compressed text not only reduces the size of the file to be transmitted but it also provides security for the data.

Decompression will be done on the client side, it happens prior to decryption. Decompression undergoes the elimination method which is one of the basic mathematical concepts. The formula for elimination method can be quoted as $A + B = C$ and $A - B = C$. By solving these two equations value of A and B can be obtained. During compression process the difference between the neighboring characters will be obtained. Those values will be used to perform decompression. Consider the above example compressed text "iols". During Compression the difference between neighboring values are as below

C – O =-12 = abs(-12) = 12

M – P =- 3= abs(-3) = 3

R – E = 13

S – S = 0

On the client side, the respective number for each alphabet in the received compressed text will be multiplied by 2. This manipulation is similar to finding the sum of the neighboring characters. The received text will be "iols" in our case. So,

i – 8 * 2 = 16
o – 14 * 2 = 28
l – 11 * 2 = 22
s – 18 * 2 = 36

above obtained results will be similar to

o + c = 16
p + m = 28
r + e = 22
s + s =18

The differences will be used to generate key (will see in detail at key generation). So these differences can be extracted from the key in client side and can be used to find the original characters. In this example on client side values 12, 3, 13 and 0 will be extracted and values 16, 28, 22 and 36 will be calculated.

So by elimination method, A + B = 12 and A – B = 16 on solving we will get A= 14 and B = 2 which gives the characters o and c. similarly other characters are manipulated and appended to get the decompressed text which has to be sent for decryption.

2.3 Key Generation

This module is used to generate a key based on the key entered by the user for the encryption and the original data. The output of this phase will be the key that will be sent in TCP connection. The key generated will be the combination of the key and the difference between the characters which was obtained in compression phase. The key entered by the user should be considered in a 5 bit representation. Consider the original data to be of single word. Divide the constant 10 with the length of the compressed cipher. The quotient will be the number of parts present in key to be generated. This number of parts represents the number of segmentation to be done in the key entered by the user. Let us consider the same "compress" example in which length of the compressed cipher (i.e.) "iols" is 4.

So the key should be segmented into 3 partitions.

10/4 = 3(Number of partitions)

10/3 = 4(Number of bits from key in each partition)

Consider akey and bkey as two keys entered by user. Here we consider 2 keys as affine cipher requires both the keys for encryption. If other encryption techniques are preferred then one key can be represented in 10bits or the same process can be pursued with 5 bits. Let a key and bkey be 11 and 7 respectively. Representing the keys in 5 bit format like a key = 01011 and b key = 00111. These two keys should be appended together, so it will look like 0101100111. As the number of bits in each partition is calculated to be 4, the above appended key should be split up into 3 partitions with maximum 4 bits each. So the segmented key will be as given below

0101 1001 11. In between each segmented key the difference calculated in the compression part should be embedded. The differences calculated in compression part are 12,3,13 and 0. The binary representation of all these values should be embedded in between each partition. In some cases the differences may be a negative value. So if the value is negative then add 1 prior to its binary representation else add 0. In the above case 12 and 3 will be negative values. The final binary representation of differences will be 101100, 100011, 001101, and 000000.

These differences will be embedded between the key partitions. The final key generated by this algorithm – 10110 0010110001110010011011110000000. This key will be sent to the user so that even the sender cannot predict what will be the key generated. If the data is to be in larger size then the key can be generated with any random 6 or 7 letter word from the original message. This paper has been tested with the first 6 letter word in the original data. On the client side, this key will be received. Then the code has to extract the differences and the original key from the received data. With the help of the differences extracted decompression can be executed. With the original key extracted decryption can be done to get the original message.

2.4 Traversals

The 4th layer of security is traversing the data through any particular shape. There exist 9 different shapes of traversals which are known as snake lake horizontal, snake lake vertical, raster horizontal, raster vertical, z-horizontal, z-vertical, spiral, zigzag and diagonal. Data can be read in either of the above traversal methodology. So the data will be arranged in a different manner from the original manner. This paper has been tested with the horizontal snake lake pattern. The common direction of the traversals is listed here.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- Raster Horizontal: 1,2,3,4,5,6,7,8,9,10,11,12
- Raster Vertical: 1,5,9,13,2,6,10,14,3,7,11,15,4,8,12,16
- Snake Horizontal: 1,2,3,4,8,7,6,5,9,10,11,12,16,15,14,13
- Snake Vertical: 1,5,9,13,14,10,6,2,3,7,11,15,16,12,8,4
- Z-Horizontal: 1,2,5,6,3,4,7,8,9,10,13,14,11,12,15,16
- Z-Vertical: 1,5,2,6,9,13,10,14,3,7,4,8,11,15,12,16
- Zig-Zag: 1,2,5,9,6,3,4,7,10,13,14,11,8,12,15,16
- Diagonal: 1,5,2,9,6,3,13,10,7,4,14,11,8,15,12,16
- Spiral: 1,5,9,13,14,15,16,12,8,4,3,2,6,10,11,7

By changing the starting point and the direction of traversal various different patterns can be obtained. To make still more complex and secure random selection of traversal can be chosen and the respective reverse traversal can be executed on the other end. This random selection will make even the sender unknown about the traversal done and it will not be easy to predict the traversal undergone by any attacker as the data will be misarranged, encrypted and compressed. Consider the word "Cryptography". We can consider 2*2 matrix so compress in 2*2 matrix will be arranged like

C	r
y	p
t	o
g	r
a	p
h	y

After snake horizontal traversal the data will look like "Crpytorgaphy". The dimension of the matrix can be decided by the sender. If that too is made to be random then the increase in security will be directly proportional to the complexity to perform cryptanalysis. On the client side this data should be arranged back to the original order in prior to Key extraction, decompress and decryption.

2.5 Hiding Data

The next layer will be appending the data with the contents of text file, image or audio. After appending, the multimedia file will be sent as data to the other end. On the client side the file is received but the text file, image or audio to which the data was appended need not be saved as the client end has no use with that file. So the appended text can just be extracted and the remaining contents of the file can be ignored. The data will be read from the text/multimedia file and will be passed to traversal module in the client side. As the UDP can support only 60KB (approximately) large multimedia files need to be segmented into packets on server side and integrated on client side. We append the data into a multimedia file so obviously the size of the file to be transmitted will be above 60KB which leads to segmentation. In general, Even if one of the packets get lost or damaged the integration of file wouldn't be successful. But in the case of this paper, the file somehow going to be ignored and only the message has to be extracted so loss of packets will not affect the flow of the process.

2.6 TCP and UDP Together

This technique is the final layer to implement security for the data. The data to be passed will be passed through the UDP and generated key will be passed through the TCP. Here, the data is the multimedia file or text file to which the traversed data is appended. This file will be segmented into packets and each packet will be transferred through Datagram Sockets. The generated key will be sent through TCP socket. For example the key generated in above key generation technique is 10110 001011000111001001101111000000 will be transferred through TCP connection. This technique of combining both the connections will make the data secure.

Let us consider 3 cases

Case 1: If a cryptanalyst or an attacker attacks the TCP socket, the generated key will alone be obtained by the cryptanalyst. To get the original key from this generated key the differences should be extracted. To extract the differences, length of the compressed text should be known from which the number of partitions and number of bits in each partition has to be calculated. But the compressed text is traversed and hidden into a multimedia file which is further being transmitted in UDP (i.e.) in a different connection. So with just the generated key it becomes very complicated to get the original key.

Case 2: If a cryptanalyst or an attacker attacks the UDP socket, the multimedia file contents will be obtained by the cryptanalyst. Here, in this paper the byte array of the multimedia file is only transferred by the UDP socket so as the contents will look like combination of special characters, characters and numerals it will be difficult to analyze that only the appended data will be the original data. If the original data has to be obtained by the cryptanalyst then the position of the appended data should be identified, then it should be traversed in a particular pattern used (pattern will be unknown), decompression should be performed, original key has to be obtained to perform encryption but the key will be passed through TCP connection which is combined with the binary representation of a 6 or 7 letter word in the message. So with just the message from the socket without the key it is impossible for the cryptanalyst to read the message.

Case 3: Even if both the channels are being observed then it would be difficult for the cryptanalyst to extract the key from the duplicate key and even all the 6 layers has to be performed in reverse order on the entire data to get back the original data. This reduces the probability level to obtain the real message getting transferred.

From these 3 cases, it can be understood that by combining TCP and UDP the data can be passed in a secure manner and even if any packet loss occur it does not affect the flow or execution of the process as only the appended data is required.

2.6.1 Algorithm

2.6.1.1 Sender' Side

Step 1: Select a file and read the necessary contents.
Step 2: Pass the data to the encryption module.
Step 3: Get two keys from the user and verify for the co-prime feature of first key.
Step 4: Encrypt the data with the key and corresponding techniques.
Step 5: Pass the encrypted contents to the compression module
Step 6: In compression module, get the ASCII values for each character and subtract it with 97(to get the position of the character in the character set).
Step 7: Find the average of the neighboring character and save the difference between the characters in a new array.
Step 8: Replace the average value obtained with its respective character.
Step 9: If the encrypted string length is odd just append the last character with the compressed string.
Step 10: Pass the compressed string to the traversal module.
Step 11: Read the data in the required pattern.
Step 12: Select the multimedia file to which the data has to be hidden.
Step 13: Write the data into the selected file.
Step 14: Find a 6 letter word from the original data contents.
Step 15: Make use of the differences of neighboring characters in this particular word and the keys entered by the user to generate the key that has to be traversed.
Step 16: Establish the TCP Multicast Connection and send the data to the clients who are ready to receive.
Step 17: Segment the final data into packets of 60000 bytes each and send those packets through UDP multicast connection.

2.6.1.2 Client's Side

The flow of processes on the client side will be directly opposite to the server side. The psuedocode for client side will be:
Step 1: Receive the data packet from UDP and get the contents of buffer array
Step 2: Extract the original data from the buffer array and just ignore the remaining part of the array.
Step 3: Pass the extracted data to traversal module, output should be properly arranged data (i.e.) reverse traversal should take place.
Step 4: Receive the key from TCP connection, segregate the original key and the difference between the characters.
Step 5: In terms of using substring function the original key can be segregated.
Step 6: Pass the traversed data to the decompression module that decompresses the obtained data into double its length using the basic elimination methodology.
Step 7: In decompression module, multiply the responding value of each alphabet in a character set by 2. That gives the answer for $A + B = C$.
Step 8: By getting the differences from sender the equation $A - B = C$ can be formed.
Step 9: Evaluate both the equations to get the characters, append all the characters and obtain the decompressed text.
Step 10: Using the extracted original key decrypt the decompressed data and get back the original message.
Step 11: This original message should be saved in the client's system at the desired directory.
Thus, the message is delivered to all client's that are active and requested to receive from the server.

2.6.2 Performance Analysis

2.6.3 Comparison Analysis

As the compression algorithm is implemented with a new technique, it has been compared with the existing techniques

Table 1. Performance Analysis

Module	700 characters	1400 characters
Encryption	15 ms	15 ms
Compression	1 ms	1 ms
Decompression	1 ms	1 ms
Decryption	15 ms	15 ms
Key Generation	1 ms	1 ms
UDP Send	4 seconds(approx)	5 seconds(approx)

Table 2. Comparison analysis

Algorithm	700 characters	1400 characters
Average value concept	1ms	1ms
Using Deflator class	2ms	3ms
Huffman algorithm	31ms	46ms
LZW algorithm	12ms	17ms

2.6.4 Comparison Graph

From the graph it is proved clearly that average concept algorithm works better than other existing algorithm and even the performance don't decrease even if the input size increases.

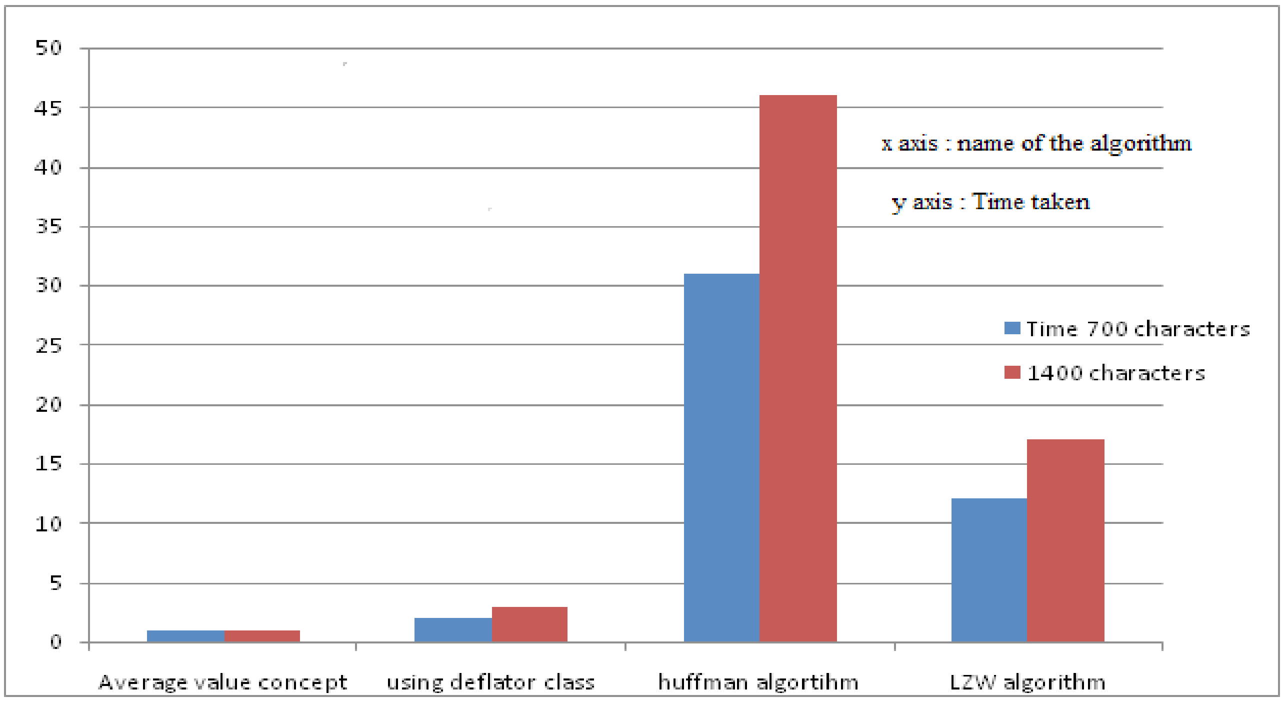

Figure 1. Comparison graph.

3. Future Work

This paper has been implemented with snake horizontal traversal and proves to get successful results. Still, this technique can be extended by implementing all 9 varieties of traversals in such a way that the traversal to be performed has to be selected at random as the security increases. Similarly, this paper has been implemented by using the first 6 letter word for key generation. Future works involve using still longer word with unfixed sizes. UDP can be replaced by RTP as it does not suit for Multimedia transfers. Still better developments can also be implemented to enhance the efficiency of this idea. Hiding the data into an image can be done through region based segmentation to improve the complexity in retaining the data during transfer.

4. Conclusion

The above explained flow has been implemented successfully and has even obtained satisfying results. Each layer individually performs in an efficient manner. After integration, the performance level has no degradation and in fact, it imparts better security too. This idea is being enhanced to meet further requirements too. Improvements will be made refer to increasing the performance and security without affecting the complexity level on client and server side. Region based segmentation can be used to hide the data in a particular part of the image. The implementation of these improvements is in process, to enhance the security of the data.

5. References

1. Anbar M, Ramadass S, Manickam S, Al-Wardi A. Connection Failure Message-based Approach for Detecting Sequential and Random TCP Scanning. Indian Journal of Science and Technology. 2014 May; 7(5):628–36.
2. Sarma MP. Performance Measurement of TCP and UDP Using Different Queuing Algorithm in High Speed Local Area Network. Technical report.
3. Coates GM, Hopkinson KM, Graham SR, Kurkowski SH. Collaborative, trust-based security mechanisms for a regional utility intranet. IEEE Trans Power Syst. 2008; 23(3):831–44.
4. Zengzhi L, Zeng-zhi L, Yan-ping C. A control theoretic analysis of mixed TCP and UDP traffic under RED based on nonlinear dynamic model. IEEE Third International Conference on Information Technology and Applications, ICITA; 2005.
5. Naqvi AN, Abbas AM, Chouhan TA. A Performance Evaluation Of IEEE 802.16 e Networks For TCP And UDP Traffics. International Journal of Engineering Research and Technology. 2012.