Collaborative work with Augmented and Virtual Reality - A secure network connection in Unity

Lars Tijsmans
lars.tijsmans@os3.nl

Abstract— **Multiplayer games often prioritize latency over security. The TU-Delft is developing an AR/VR based remote visit system where security is a priority. In this paper a method is researched to setup a secure network channel in Unity using TCP and TLS. The results show that E2E encryption adds around 6ms to the latency, so that it is possible to have real-time communication over a secured channel in Unity.**

I. Introduction

The game industry has boomed in the last decade and a significant portion of the most played games are multiplayer games. [1] Due to the nature of games, using secure networking protocols is not a priority. Games often send non trivial data to other clients where a potential eavesdropper has very little use for. Securing the connection can also create a delay, which can have a huge impact in a real-time game. [2]

There are however cases where security of the data send is a priority. The TU-Delft is developing a AR/VR (Augmented reality/Virual reality) communication system named "Smart-Factory". This system provides a digital environment where a host (using AR), can guide guest around through a factory. The guest uses VR and are in a virtual room where they can see the host and the machinery. The state of the machines is measured by sensors and send to the clients. For many companies, this data is very sensitive. Securing this information is an important requirement, when using this system.

The game engine used for the "Smart-Factory" is Unity. [3] Unity does not have build in encryption in their networking libraries. In this paper I research a method to setup a secured channel in Unity and compare the performance to a standard insecure channel.

II. Related work

A paper written in 2008 researched the security in a CVE (Collaborative virtual environment). [4] They state that confidentiality is important when eavesdropping is a risk. The proposed solution is the use of a secured protocol, HTTPS. Using server-side certificates to provide an encrypted channel, a web-based client can securely send information to the server. Another thread that is introduced in this paper is a DDOS attack. This does not affect confidentiality but it does affect the availability. By using server software that drops non-game related packages, the risk of such an attack will be reduced.

Authenticating a client is also important when sensitive data is involved. The connection can be secure, but if the other party is not who they say that they are. The security is breached. A paper in 2017 researched how a client can authenticate itself by using PIN codes or patterns. [5] The proposed solution is to project an input number pad in front of the user. The user can use the motion controls to aim at the numbers and input a PIN or make a pattern. The user is this situation does not have a physical keyboard to use and projecting a keyboard is difficult use without affecting the user experience.

III. Research questions

The research question is defined as:

How does the performance of a TLS websocket connection in Unity compare to a standard insecure connection?

IV. Background

Unity is a game engine found in 2004. It can be used to create all kind of video games like 2D, 3D and multiplayer games. [6] Although Unity does not offer encryption in their networking libraries by defaults, they do have a method to include your own multiplayer encryption plug-in. [7] If this is used, all data send to the network by Unity will be encrypted and all data received from the network will be decrypted automatically by the provided plug-in. See Figure:1 The plug-in must provide its own encryption/decryption algorithm and has to implement the following mandatory functions: Encrypt Data, Decrypt Data, SafeMaxPacketSize and ConnectionIdAssigned.The first two are self explanatory. The SafeMaxPacketSize function is called when the maximum packet size has to be changed. If you try to send more data than is allowed in a packet, it gets cut off and the receiver will miss data. Setting a maximum forces Unity to split its data over several packets if the limit is reached. The ConnectionIdAssigned function is used when a new connection is accepted by the server. During research I found out the UNet is longer maintained by Unity anymore since 2018 and that they are currently working on a successor. [8] Because UNet is no longer supported I decided to limit the time spend on this solution.

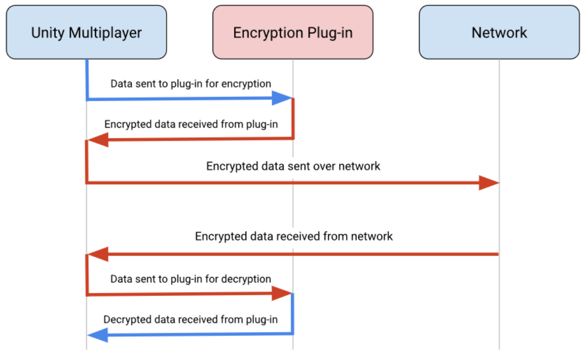
Unity Multiplayer
Encryption Plug-in
Network
Data sent to plug-in for encryption
Encrypted data received from plug-in
Encrypted data sent over network
Encrypted data received from network
Data sent to plug-in for decryption
Decrypted data received from plug-in

Fig. 1. Diagram describing the plug-in setup [9]

To prevent any eavesdropper from intercepting data, E2EE (end-to-end encryption) can be used. [10] In E2EE the sender encrypts data with an encryption key, which can only be decrypted by the receiver with the corresponding decryption key. Before E2EE is possible the sender and receiver need to have the proper keys. One way to do this is by using pre-shared keys. [11] The upside is that it is easy to implement. The downside of this is that the the keys need to be shared before the system is deployed. Also creating connections with new clients becomes more difficult since the same pre-shared key is reused for all connections. Which is not a safe option. If the pre-shared keys are leaked the system becomes as safe as not using encryption at all, since anyone possessing the key could decrypt the data.

A better alternative is using the Diffie Hellman key exchange. [12] Using Diffie Hellman eliminates the need of using pre-shared keys by being able to negotiate a key-pair in a secure way for every new connection that is established. By using the discrete logarithm problem, which is computationally infeasible to solve, the server and the client are able to exchange keys without having to send the literal keys to each other. An eavesdropper is not able to intercept any key, and therefore all the encrypted messages that follow can not be tapped.

There are several encryption algorithms. A very well known one is AES (Advanced Encryption Standard). Which is a symmetric encryption algorithm. [13] Since AES is open source and extensively tested, it is considered to be a very secure encryption standard. [14] Although the block length is fixed, 128 bit. The key length can vary in 128, 192, 256 bit respectively. Increasing the key size, increases the amount of round processed. DES or 3DES is the predecessor of AES and is inferior in both terms of security and speed. [15]

V. Methodology

In this section the approach is presented for performing the research. It will detail the steps that are taken to create the test implementation and run the experiment.

A. Secure websocket

The implementation is made in unity using the mirror networking library in Unity. From the GitHub repository the test implementation can be downloaded. [16] In the Assets/Scene folder the "TLSTestScene.unity" can be found. You can build this scene and run it to use a secured websocket over TLS. When the network manager is inspected from the hierarchy, a component named "Websocket TLS" is shown. All traffic that is handled by the network manager is send and received using this script. There are three properties you can setup for this component. The first is the port. This value is both used by the client and the server to establish a connection. Make sure that both the client and the server are running on a version with the same port number. Otherwise setting up a network channel will fail. The second value is the path to a SSL certificate. The format has to be ".pfx". The last value is the password that corresponds with the certificate. If the certificate is created without a password, leave this field empty. The "Websocket TLS" can be added to every mirror example project. During the research I tested using the pong game and the basic game.

B. Certificate

There are two methods to get a certificate. The easiest method is to get a certificate from a verified certificate Authority (CA). Certificates that are issued by verified CA's will be trusted by most devices. For local development and testing purposes I created my own local CA and used it to issue a self-signed certificate. [17] Make sure that the root certificate is both trusted by the machine that host the server and the machine that runs the client. If something is wrong with the certificates, the client trying to connect with the server will log a TLS handshake error when run in debug mode.

C. Collect data

When everything is setup properly you can launch instances of the scene. There needs to be at least one host and a maximum of two client can join. The host can also be one of the clients. Connect to the host by typing the hosts IP address. The port number does not need to be specified since it is set in the network manager component. when the ball is spawned, the two clients have connected successfully and the secure channel is now used to send player and ball information back on forth form the server to the clients. To inspect the packets that are send and determine the latency Wireshark was used. [18] The packets were send over TCP so all the packets were acknowledged by the other party. By using the timestamps of the packets the latency could be determined.

VI. Results

In this section I will present the results that were gathered while conducting the experiment.

In figure 2 the results from the test implementation are plotted. Initially the secure and insecure connections seem to behave the same. The latency for both implementations

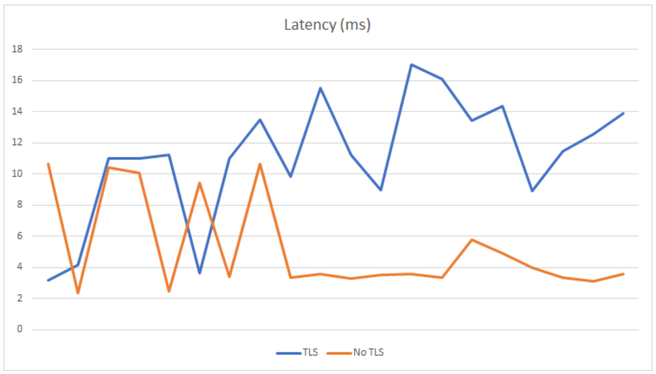

Fig. 2. Plot showing the results comparing the secured channel with an insecure channel

	Min(ms)	Max(ms)	Average(ms)
TLS	3,14	16,9	11,09
No TLS	2,49	10,65	5,24

is similar. However, after the initial phase there is a clear distinction visible between the channel over TLS and the channel that is not using TLS.

Table VI shows the lowest and highest latency that was measured and shows the average latency that was measured while conducting the experiment. Both the minimum and maximum latency are higher when using TLS compared to the non encrypted equivalent. The average latency using the insecure channel was 5,24ms. This is very low and allows for real-time communication between the client and the server. For the connection over TLS, the average latency was 11,09ms. Although this is almost double the latency, it is still low. Using TLS adds approximately 6ms to the latency.

VII. Discussion

As the results show the latency measured during the experiment is higher than using a insecure connection but not by a huge margin.

A. Latency

According to research done in 2006, the fastest type of games can have a latency to up to 100ms without impacting the user experience. [19] This suggests that having a real-time conversation with a ghost in a virtual environment should be possible with a latency of 11,09ms. The latency can be affected by other factors. It should be noted that the remote client the experiment was conducted with, was geographically close to the host machine. They were about 20km apart. If the client and host are physically separated even more, the latency will naturally go up. According to Global Ping Statistics the average ping from Amsterdam to Washington (across the Atlantic) is 84ms. The average ping to Tokyo is 242ms. This is significantly higher. If one were to host a remote visit in Amsterdam with a visitor from Tokyo, the user experience will be affected by the latency. It should be noted that being geographically further apart affects both the secured an insecure channel equally. Using TLS still only adds around 6ms to the latency.

B. Authentication

Setting up end to end encryption between the server and the clients is an important step for creating a secure multiplayer game. However, authenticating the user before letting it establish this connection is also important for maintaining integrity. Due to time constraints there is no authentication system in the test implementation. Anyone that knows the address and port of the server is able to connect. A form of authentication should be created before using it in a production environment.

C. Mixed reality

The conducted experiment is done using desktops rather than AR or VR glassed. Since Unity is a cross platform game engine it is possible to build the application to a wide range of devices. Therefore one could develop a game and build it to both a computer and the Microsoft Hololens. For this reason the focus is on setting up a secure connection in Unity rather than doing it specifically for mixed reality. Many VR glasses are not stand alone. They are connected like a display to a computer. All the processing is done by the PC, including encryption and decryption.

VIII. Conclusion

In this paper I analyzed a method to setup a secure TLS connection in Unity. The relevance from the research comes from the fact that game engine multiplayer libraries offer very little security functionalities. The results show that it is possible to use the secured TLS connection and still maintain a playable environment. The latency is low enough to allow for real-time communication. By using AES-256 for end-to-end encryption, a potential eavesdropper can not do an anything use full with the intercepted data.

To complete the setup of a secure multiplayer game, the client also need to authenticate itself. According to research the most convenient method is to use either a PIN or a patter, since this has the right balance of safety and it does not harm the user experience like a complete projected keyboard would.

The performance over an end-to-end encrypted channel is lower than that of a standard insecure channel, but real-time gaming over a secure channel is possible.

IX. Future work

Due to time constraints, there are still aspects left to be researched.

The results are gathered by using high-end desktop computers. Conducting the same experiment on less powerful machines like AR/VR glasses might give different results. However, most modern CPU's are optimized to perform AES encryption and decryption, so really diverging results are not expected. Testing on different machines is still desired.

TLS supports many different cipher suites. In the experiment TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA was used. AES-256 is a widely used and proven safe way for encrypting data. However other cipher suites could lead to different and maybe better results.

In the future a UNet successor will be released. Using the native Unity multiplayer library could be a better solution than using a third party alternative in regard to security and support. When the successor is released it will be interesting to create a similar test setup in native Unity and compare the results.

Using TLS does prevent eavesdropper from being able to make sense of the data. However if anyone is able to setup a secure connection with the server without authentication, the protection of sensitive data is not guaranteed. Therefore a authentication system has to be used were a client that wants to connect to the server, needs to provide credentials in the form of a PIN or pattern. Combining this authentication system with the a TLS channel provides a complete playable and secure environment.

References

[1] *Most played video games.* URL: https://en.wikipedia.org/wiki/List_of_most-played_video_games_by_player_count (visited on 06/05/2020).

[2] Lothar Pantel and Lars C. Wolf. "On the Impact of Delay on Real-Time Multiplayer Games". In: *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '02. Miami, Florida, USA: Association for Computing Machinery, 2002, 23–29. ISBN: 1581135122. DOI: 10.1145/507670.507674. URL: https://doi.org/10.1145/507670.507674.

[3] *Unity.* URL: https://unity.com/ (visited on 06/05/2020).

[4] Yuseung Sohn Miguel Fernandes Seunglim Yong Hyun-Yi Moon. "A Survey of Security issues in Collaborative Virtual Environment". In: *International Journal of Computer Science and Network Security* 8 (Jan. 2008), pp. 14–19. DOI: 10.1.1.115.5260.

[5] Ceenu George et al. "Seamless and Secure VR: Adapting and Evaluating Established Authentication Systems for Virtual Reality". In: Feb. 2017. DOI: 10.14722/usec.2017.23028.

[6] *Unity.* URL: https://en.wikipedia.org/wiki/Unity_Technologies (visited on 06/05/2020).

[7] *Multiplayer Encryption Plug-ins.* URL: https://docs.unity3d.com/Manual/UNetEncryptionPlugins.html (visited on 06/05/2020).

[8] *UNET deprecated.* URL: https://support.unity3d.com/hc/en-us/articles/360001252086-UNet-Deprecation-FAQ?_ga=2.118709003.309166409.1593878819-394192428.1591272201.

[9] *Plugin diagram.* URL: https://docs.unity3d.com/uploads/Main/UNetMultiplayerEncryption.svg (visited on 06/05/2020).

[10] *end-to-end encryption (E2EE).* URL: https://searchsecurity.techtarget.com/definition/end-to-end-encryption-E2EE (visited on 06/05/2020).

[11] *Pre-shared key.* URL: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director3.1/topics/concept/wireless-wpa-psk-authentication.html.

[12] *Diffie-Hellman Protocol.* URL: https://mathworld.wolfram.com/Diffie-HellmanProtocol.html (visited on 06/05/2020).

[13] Joan Daemen and Vincent Rijmen. *AES Proposal: Rijndael*. 1999.

[14] Gurpreet Singh and Supriya Kinger. "A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security". In: *International Journal of Computer Applications* 67 (Apr. 2013), pp. 33–38. DOI: 10.5120/11507-7224.

[15] Hamdan Alanazi et al. "New Comparative Study Between DES, 3DES and AES within Nine Factors". In: (Mar. 2010).

[16] *Test implementaion Gitlab.* URL: https://gitlab.com/lars.tijsmans/rp1.

[17] *Create Certificate.* URL: https://letsencrypt.org/docs/certificates-for-localhost/.

[18] *Wirehark.* URL: https://www.wireshark.org/.

[19] Mark Claypool and Kajal Claypool. "On latency and player actions in online games". In: (2006). URL: https://digitalcommons.wpi.edu/cgi/viewcontent.cgi?article=1053&context=computerscience-pubs.