International Journal of Advancements in Technology

OPEN ACCESS Freely available online

Research Article

Zero-Trust Based Ad Hoc Surveillance Robot

Mahantesh Salimath*, Rahul Sharma

Department of Electrical and Computer Engineering, Cornell University, Ithaca, New York, USA

ABSTRACT

In an era of growing security concerns, with the context of ad hoc wireless communication which is common in the domain of Internet of Things (IoT) or remote areas where Internet is not a viable option, establishing trust between two entities is challenging. Not only the protocol should be difficult to deny connection for untrusted structures but also cost-effective and practically viable. This paper demonstrates how zero-trust framework can be employed in ad hoc wireless communications through a surveillance robot with remote control and live video stream. We achieve zero-trust connection using mutual-Transport Layer Security (mTLS) protocol between the endpoints. This secure connection is then used to exchange control messages for robot movement and for the live video stream. The endpoints communicate over ad hoc Wi-Fi and hence there is no need for router or Wi-Fi access points or internet. Since Wi-Fi has limited range, this work can be further extended over long-distance communication where a network of ad hoc peers employs a routing protocol for transfer of packets between two endpoints within their network.

Keywords: Ad hoc wireless communication; mTLS; Surveillance robot; Zero-trust

INTRODUCTION

We are living in a world where smart devices co-exist along with human beings, they assist us in several ways and function autonomously without needing help or human intervention. These smart devices can talk to each other and be able to decide on possible courses of action with the advent of Artificial Intelligence (AI). It is common that these devices talk to each other wirelessly and directly peer-to-peer (ad hoc) without the need of a router or access point. These types of devices are especially useful in remote areas without internet access or where a private network of smart devices is created for security purposes, but this type of security is barely enough. AI can also be adversely used to disadvantage. AI can create or predict several things; they can create a picture or predict the next word in a sentence. Internally, AI engines use Large Language Models (LLM) which are trained with an immense number of data points to support in prediction/creation. For example: If given a history of old passwords, AI can predict future passwords.

The ways in which AI can be employed to use vulnerabilities to hackers' advantage are innumerable. Hence, with the growing concern of security, it becomes very vital to use strategies like zero-trust. Zero-trust means no user, device, or component is trustworthy and requires strict identity verification and authorization for access. It is also important to use security protocols or algorithms which are proven and well tested.

Because often a new protocol which seems very strong at surface level might have vulnerabilities under which are not discovered yet and in this era of technology, vulnerabilities won't stay hidden for long. Hence, it becomes important to address the security concerns in ad hoc wireless communication mediums by employing zero-trust model using proven and well tested protocols.

MATERIALS AND METHODS

An overview of model

To demonstrate zero-trust model in ad hoc wireless communication, we built a robot with wheels and camera using Raspberry Pi 2 model B, we implemented a web server on the Raspberry Pi using flask python module to gain remote access [1,2]. To connect to the robot, we installed a Wi-Fi module in ad hoc mode for peer-to-peer communication. We then employed mTLS protocol to achieve zero-trust [3]. Steps are explained in detail over the upcoming sections. Figure 1, shows the overview block diagram with different components involved.

Correspondence to: Mahantesh Salimath, Department of Electrical and Computer Engineering, Cornell University, Ithaca, New York, USA, E-mail: mahantesh.salimath130@gmail.com

Received: 21-Nov-2024, Manuscript No. IJOAT-24-35328; **Editor assigned:** 25-Nov-2024, PreQC No. IJOAT-24-35328 (PQ); **Reviewed:** 09-Dec-2024, QC No. IJOAT-24-35328; **Revised:** 16-Dec-2024, Manuscript No. IJOAT-24-35328 (R); **Published:** 23-Dec-2024, DOI:10.35841/0976-4860.24.15.313

Citation: Salimath M, Sharma R (2024). Zero-Trust Based Ad Hoc Surveillance Robot. Int J Adv Technol.15:313.

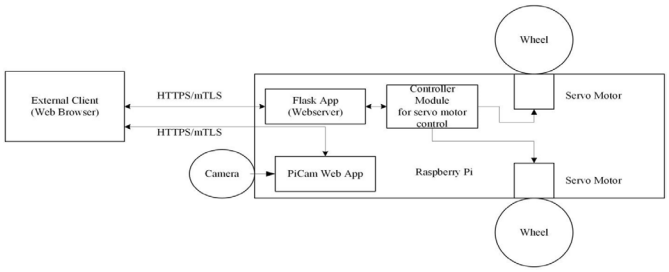

Figure 1: Overview of block diagram.

Zero-trust using mTLS

To achieve zero-trust, we are employing mTLS protocol. TLS protocol is commonly used in HyperText Transfer Protocol Secure (HTTPS) connections on the internet. Note that HTTPS connections are not necessarily qualified as zero-trust since mutual verification need not necessarily be done [4]. Partial verification such as verification of server by client or client verification by server are also accepted in HTTPS connections. HTTPS connection over the web make use of several registered Certificate Authorities (CA's) who sign the certificates that are already inserted in our browsers (client) and partial verification (verification of server by client and not *vice versa*) is done to allow access to different websites [5]. Web servers don't verify the client in most of the cases to allow any client to connect to the web server and access the website. From a security and privacy perspective of our network, we decided to create our own self-signed root CA private key and certificate, server root key, server certificate, client root key and client certificate. We used Open Secure Sockets Layer 3.0 (OpenSSL 3.0) which is the latest version of OpenSSL, a software library for applications that provide secure communications over computer networks, to create different root keys, certificates and to support in mutual verification and authentication on server and client [6]. We then used root CA to sign the server certificate and client certificate [7]. Figure 2, shows the different keys, certificates involved along with their association. The root CA certificate will then be installed on server and client which will be used during TLS handshake process for mutual verification and authentication of server and client.

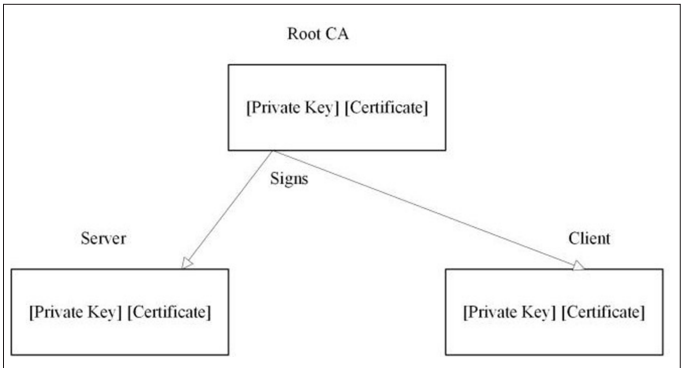

Figure 2: Root CA private key signs the server and client certificates.

Morgan et al., explains the mTLS mechanism in detail during TLS handshake, but briefly, during TLS handshake server provides its certificate and public key to client, client verifies the server certificate using root CA certificate [3]. Client then sends its certificate and public key to the server so that server can verify and authenticate the client certificate using root CA certificate. Once this mutual authentication is done, both server and client will possess each other's public key which is then used to exchange session token/keys. If the authentication fails, then the connection will be terminated immediately. Since, a client needs to be in possession of client certificate that is signed by the root CA, not any client can connect to the server. And since server also needs to be in possession of the server certificate that is signed by the root CA, a hacker can't spoof and pretend to be a server and *vice versa*. This establishes the zero-trust framework upon which the client and server connection is based upon. Further communication and message exchange uses the secure connection thus established.

Webserver using flask

We used Python 3 programming language for implementing the software components involved since it has modules handy to be used in the development [8]. One such module is flask which provides easy and quick development of web server. We used SSL module for configuration of root CA certificate, server key and certificate created [9]. The configured certificates and keys, using SSL module, can be linked and used in flask web server. This will enable web server to use the keys and certificates during TLS handshake process and hence in secure communication. As part of the webserver a control page as shown in Figure 3. is provided to the client on successful connection. User can control the robot by clicking on the different control buttons available on the control web page. User actions are sent to the server using HTTP POST requests and are then serviced accordingly by server based on the button clicked.

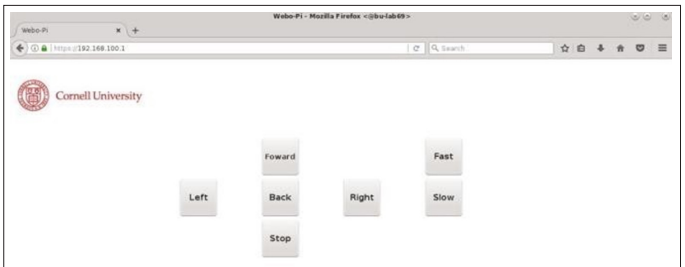

Figure 3: Control page on successful connection.

Motor controller module

Robot movement controlled by actuating the servo motors is implemented as a different python module, but the web server module invokes the respective action interface based on the input received from the client. Servo motors are connected to the General-Purpose Input Output (GPIO) ports of the Raspberry Pi. The safe way to connect servo motors to raspberry pi is to power the motors through a 6V battery or a regulated power supply of 6V. In this project, we are using 4 double-A batteries to power the motors and a 5V portable rechargeable battery for Raspberry Pi. The motors should not be connected to an unregulated power supply as the varying voltage can damage them. A generic connection diagram is as shown in Figure 4. In this project,

we used two servos that are connected to the General Purpose Input/Output (GPIO) pins 19 and 26.

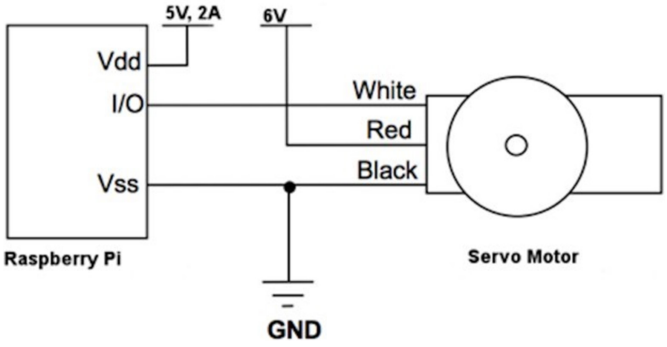

Figure 4: Servo motor interface with Raspberry Pi.

A regular servo motor is designed for 180 degrees swing i.e., 90 degrees to left of neutral position and 90 degrees to right. Continuous Rotation (CR) servos are a modified version in which the shaft can rotate continuously by essentially removing the physical hard stops. CR servos behave like compact Direct Current (DC) gear motor with built in H-Bridge driver and can be controlled through a pulsed signal like Pulse Width Modulation (PWM).

For calibration, we send the servo a 1.5 ms pulse refreshed every 20 ms. Python Raspberry Pi modules have GPIO interfaces with PWM functionality which are used to send the actuating signal. The GPIO interface in Raspberry Pi module takes frequency and duty cycle as an input. The values for frequency and duty cycle are given below.

Frequency=1/(1.5+20)=46.511 Hz

Duty cycle=1.5/(1.5+20)=6.97%

As the length of the pulse decreases from 1.5 ms to 1.3 ms, the servo will gradually rotate faster in the clockwise direction, pulse width modulation is as shown in Figure 5. To express full speed clockwise rotation in terms of Raspberry Pi GPIO module, we define our duty cycle as below.

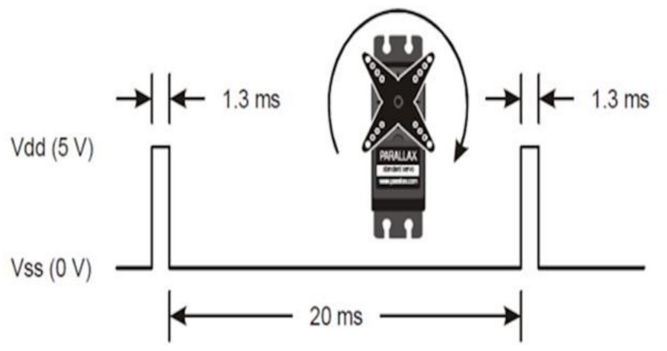

Figure 5: Actuating signal for CR servo motor.

Frequency=1/(1.3+20)=46.948 Hz

Duty cycle=1.3/(1.3+20)=6.10%

Similarly, as we increase the pulse width from 1.5 ms to 1.7 ms, the servo attains it maximum speed in the counter clockwise direction and we get,

Frequency=1/(1.7+20)=46.082 Hz

Duty cycle=1.7/(1.7+20)=7.83%

Raspberry Pi (Pi) camera

Python module Pi camera 2 was used to generate stream from Pi camera and the stream was hosted on a web server running on a different ad hoc Internet Protocol (IP) address than the one used for accessing the control page [10]. This was done to simplify the implementation and avoid dwelling into complications of developing a web page to host control and video stream on a single Uniform Resource Locator (URL). Again, flask module was used to implement the web server and employed the same root CA, server key and server certificates to be used during TLS handshake for establishing a secure connection. Figure 6, shows a snapshot from the live video stream hosted.

Figure 6: Live video stream from Pi camera.

Ad hoc Wi-Fi configuration

To provide wireless capabilities to the bot, we used the Edimax Wi-Fi module. The main advantage of ad hoc is that it does not require any Wi-Fi access point or router to connect from the client. The client can simply connect and access bot controls. The configuration of an ad hoc network on Raspberry Pi can be done by editing the network interfaces file as show in Figure 7(a) [11]. On Raspberry Pi running a Linux based operating system, the network interfaces file is located at/etc/network/interfaces. Once done, re-enable the interface as shown in Figure 7(b). to activate the configuration. The ad hoc host also needs to host a Dynamic Host Configuration Protocol (DHCP) server to allow client connection [12]. We also set up a DHCP server on the Raspberry Pi to allow client connections. We used the standard dhcpd utility provided in Raspbian (Linux), a Debian Linux distribution.

```
auto wlan0
iface wlan0 inet static
address 192.168.100.1
netmask 255.255.255.0
wireless-channel 1
wireless-essid RPiAdHocNetwork
wireless-mode ad-hoc
```

(a)

```
$ sudo ifdown wlan0
$ sudo ifup wlan0
```

(b)

Figure 7: (a) Ad hoc network configuration and (b) Re-enable Wi-Fi interface.

RESULTS

The robot built is as shown in the Figure 8. It is functional and capable of movement using CR servo motors and the wheels attached. It also holds a camera and batteries to power the raspberry pi and servo motors. At the center of the chassis is a pseudo wheel which helps in balance and movement.

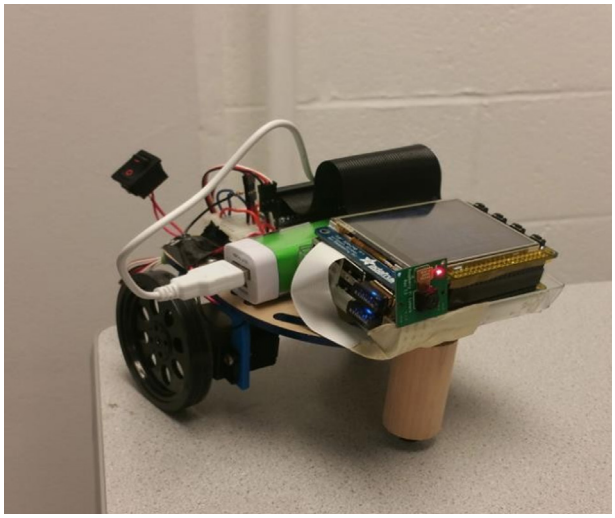

Figure 8: Surveillance robot with remote control.

To test the mutual verification done as part of the mTLS protocol, we attempted to connect to the server using Mozilla Firefox (version 52.7.0) without any root CA certificate, client certificate and client key enrolled. The secure connection failed. In the next step, we enrolled the root CA certificate but didn't enroll the client certificate and client key, in this case as well the secure connection failed as shown in Figure 9. Finally, we enrolled the root CA certificate, client key and certificate and this time the secure connection was established as shown in Figure 10, and we could access the control page to manage the robot. In a final verification test, we removed the server certificate and root CA certificate from the web server and as expected the secure connection couldn't be established by client as the server couldn't be verified. These tests verified the mTLS protocol functionality and thus mTLS established zero- trust.

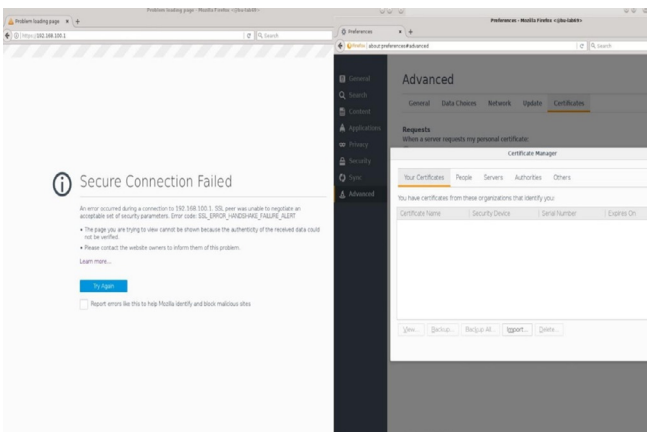

Figure 9: Client certificate and key missing in browser.

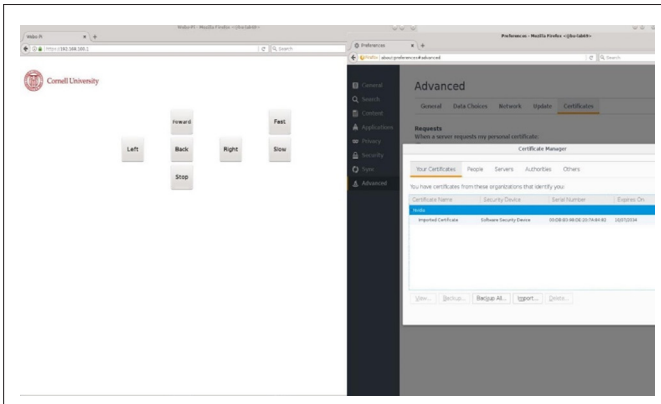

Figure 10: Client and root CA certificate, client key enrolled in browser.

Once the connection was established, we were able to click different buttons on the web page and the robot moved accordingly. On a different tab and URL, we were also able to get a live video stream from the Pi camera. Thus, allowing surveillance of the robot location remotely. With ad hoc mode we tested that the Wi-Fi connection was intact for around 100 m in an open space, and it was limited to around 15 m in the presence of obstructions (walls).

CONCLUSION

Using mTLS we demonstrated the zero-trust framework in action for ad hoc wireless communication channel. Ad hoc wireless communication is particularly helpful in remote areas where this is no internet or any area outside of the Wi-Fi or wired connectivity zone. We focused on ad hoc wireless communication channel as it poses the most challenges. Once proven to work in this mode, our findings could be applied to other less challenging mediums as well. Since Wi-Fi is used as the wireless communication channel, like every other channel it has its limitations. The main limitation is the range/distance over which the message can transmitted successfully.

To overcome this limitation, we usually locate a network consisting of routers or access points. The mTLS approach used in this paper can be extended to such a network and hence zero-trust can be used in long distance communication as well. Irrespective of the medium, mTLS can be used if the application layer protocol (HTTP in this case) is capable of or depends on transport layer security protocol.

We employed a simple strategy to have a single layer of complexity when signing the server and client certificate by signing them directly with root CA. This can be enhanced to several layers by introducing a chain of intermediate CA's. The root CA will sign an intermediate CA certificate which in turn signs another intermediate CA and so on. The last intermediate CA signs the server and client certificate.

During TLS handshake, the server and client are required to provide all the intermediate CA certificates up to the one that was signed by root CA for verification and authorization. This boosts the security framework significantly and in fact is what's recommended for usage in a finished product.

REFERENCES

1. Scott R. Beginners guide to Raspberry Pi 2. 2015.
2. Miguel G. Flask web development. Sebastopol: Meghan Blanchette and Rachel Roumeliotis. 2014.
3. Morgan J, Flynn. Linkerd: Up and running: A guide to operationalizing a Kubernetes-native service mesh. 2024.
4. Brinkmann M, Dresen C, Merget R, Poddebniak D, Muller J, Somorovsky J, et al. ALPACA: Application layer protocol confusion-analyzing and mitigating cracks in TLS authentication. In 30th USENIX Security Symposium. 2021:4293-4310.
5. Kim D, Cho H, Kwon Y, Doupe A, Son S, Ahn GJ, et al. Security analysis on practices of certificate authorities in the HTTPS phishing ecosystem. In Proceedings of the 2021 ACM Asia CCS.2021:407-420.
6. Khlebnikov A. Demystifying cryptography with OpenSSL 3.0: Discover the best techniques to enhance your network security with OpenSSL 3.0. Part 4. 2022.
7. Khlebnikov A. Demystifying cryptography with OpenSSL 3.0: Discover the best techniques to enhance your network security with OpenSSL 3.0. Part 5. 2022.
8. Monk S. Programming the Raspberry Pi, second edition: Getting started with python 2nd edition. 2015.
9. Ortega JM, Sarker MF, Washington S. Learning python networking: A complete guide to build and deploy strong networking capabilities using Python 3.7 and Ansible. Packt Publishing Ltd. 2019.
10. Norbom H. Raspberry Pi camera controls using python 3.2. 3: For Windows and Debian-Linux. CreateSpace. 2013.
11. Liu A, Chen H. Configuration of WLAN and ad hoc network access point and research on internet topology control. Int J Futur Gener Commun Netw. 2016;9(9):313-320.
12. Perkins CE. Ad hoc networking. Pearson Education India; 2008.