Master's Thesis
석사 학위논문

A Robot Operating System Framework for Secure UAV Communications

Hyojun Lee(이 효 준 李 孝 濬)

Department of
Information and Communication Engineering

DGIST

2021

A Robot Operating System Framework for Secure UAV Communications

Advisor: Professor Kyung-Joon Park
Co-advisor: Professor Kyoung-Dae Kim

by

Hyojun Lee
Department of Information and Communication Engineering
DGIST

A thesis submitted to the faculty of DGIST in partial fulfillment of the requirements for the degree of Master of Science in the Department of Information and Communication Engineering. The study was conducted in accordance with Code of Research Ethics[1]

12. 24. 2020

Approved by

Professor Kyung-Joon Park (signature)
(Advisor)

Professor Kyoung-Dae Kim (signature)
(Co-Advisor)

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of DGIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

A Robot Operating System Framework for Secure UAV Communications

Hyojun Lee

Accepted in partial fulfillment of the requirements for the degree of Master of Science.

11. 17. 2021

MS/IC
201922036

이 효 준. Hyojun Lee. A Robot Operating System Framework for Secure UAV Communications. Department of Information and Communication Engineering. 2021. 35p. Advisors Prof. Kyung-Joon Park, Co-Advisors Prof. Kyoung-Dae Kim

ABSTRACT

To perform advanced operations with unmanned aerial vehicles (UAVs), it is crucial that components other than the existing ones such as flight controller, network devices, and GCS are used. The feature of obstacle avoidance is added to the pre-existing simple waypoint missions to ensure the commercialization of UAVs. However, this feature requires additional hardware and software to recognize obstacles based on radar or lidar. The inevitable addition of components to accomplish this functionality may lead to security vulnerabilities through various vectors. Hence, we propose a security framework in this study to improve the security of UAS. The proposed framework operates in the ROS (robot operating system) and is designed to focus on several perspectives such as overhead arising from additional security elements and security issues essential for flight missions. The UAS is operated in a non-native and native ROS environment. The performance of the proposed framework in both environments is verified through experiments.

Keywords: Unmanned Aerial Vehicle, Network Attack, Cyber-Physical System, Security

List of Contents

List of Tables

List of Figures

I. INTRODUCTION

Unmanned aerial vehicles (UAVs), commonly known as drones, have been recently deployed in various environments to perform numerous tasks [1-4]. One of the most representative drone services is Amazon's Prime Air, which is currently under development [5]. Other companies are also preparing various drone projects such as storm damage evaluation, property damage assessment, and shale gas asset monitoring. The increase in the use of UAVs in various fields has resulted in growing concerns regarding the security of the unmanned aerial system (UAS). According to the "FAA Aviation Forecast 2019–2029" released by the Federal Aviation Administration along with the example described earlier, the commercial drone market is expected to triple by 2023. As the utilization of UAVs increases, there is a need to manage the security of the UAS. The need to manage the system security is not merely theoretical; it can be illustrated by real-life incidents. The 2011 drone hijacking incident, one of the most widely known cases of UAV cyberattacks, is an event in which Iranian cyber units cut off UAV communications links in the United States and diverted the UAVs into Iranian territory by manipulating the GPS (global positioning system) coordinates. Although there are many arguments about the authenticity of the case, it should be realized that cyberattacks on UAVs are possible. In addition, the most common vector in attack cases is the communication layer provided by the UAV platform.

UAS belongs to a category of cyber-physical systems (CPS). Unlike traditional embedded systems that operate individually, CPS requires the close interaction of computing and physical systems. Ultimately, CPS can be seen as the integration of computational, networking and physical processes. It is a system in which information and software technologies combine with mechanical components to deliver and exchange data as well as monitor or control a device by infrastructure such as the Internet in real time. The UAV network incorporates communications devices, computing functions and control modules to form a single closed loop from data recognition, information exchange, decision making to final execution and can be referred to as a CPS. In these CPSs, studies on system vulnerabilities, one of which is vulnerability in the network, are actively underway [6-10].

Robot operating system (ROS) is the middleware for robot software development. Unlike the operating systems used in computers, ROS is meta operating system and provides services such as hardware abstraction, low-level device control, and message delivery between processes for system operation. It is used in various robot industries and research fields due to its advantages such as development community actively involved in its constant development and efficient development. In UAS, ROS is installed and used on the UAV exterior board for advanced operations such as autonomous and clustered UAV. However, ROS lacks design for system stability. Basic safety tools are provided but these tools focus on system failure, such as time synchronization and program part accuracy; there are no measures for system attacks.

In this paper, we explain the vulnerability in an ROS-based UAV and propose a security framework to solve it. In Section 2, we describe the background of UAVs. Section 3 describes the vulnerabilities in ROS-based UAVs and how attacks are planned using them. Section 4 describes the studies and tools that have been undertaken to address the problem. Section 5 describes the framework proposed for vulnerabilities in ROS. Performance and overhead of the proposed framework are shown in comparison with that of the tools described in Section 4; a low overhead security solution is proposed that can address vulnerabilities in ROS. Section 6 describes the proposed security framework with actual implementation and verification.

II. BACKGROUND

2.1 Unmanned Aerial System (UAS)

UAS is a generic term used to denote the combination of a drone, ground-control system (GCS), and the communication system between the two. A drone refers to an aircraft that flies automatically or semi-automatic without a real pilot on board. It performs its missions by controlling its altitude and position through an internal flight controller. The flight mission is performed either by transmission from the GCS or by built-in algorithms. The conventional media used for communication are RC transmitters, Bluetooth, Wi-Fi, and radio. Drones can send and receive commands and status from the GCS through these media using MAVLink message protocol [11]. MAVLink is a light messaging protocol for on-board communication or components of drones. It can be implemented in 14 languages, including C and C++; various high-level APIs exist for interaction between other systems such as drones and ROS. The protocol can also be used by at least seven GCS software, including QGroundControl and Mission Planner, to communicate with the drone. Figure 2.1 shows the MAVLink protocol message. Figure 2.2 shows QGroundControl, an illustrative GCS in UAS configuration. Figure 2.3 shows the UAV used in this paper.

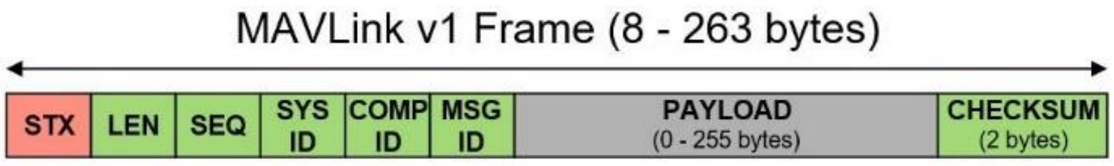


Figure 2. 1. MAVLink protocol message [12].

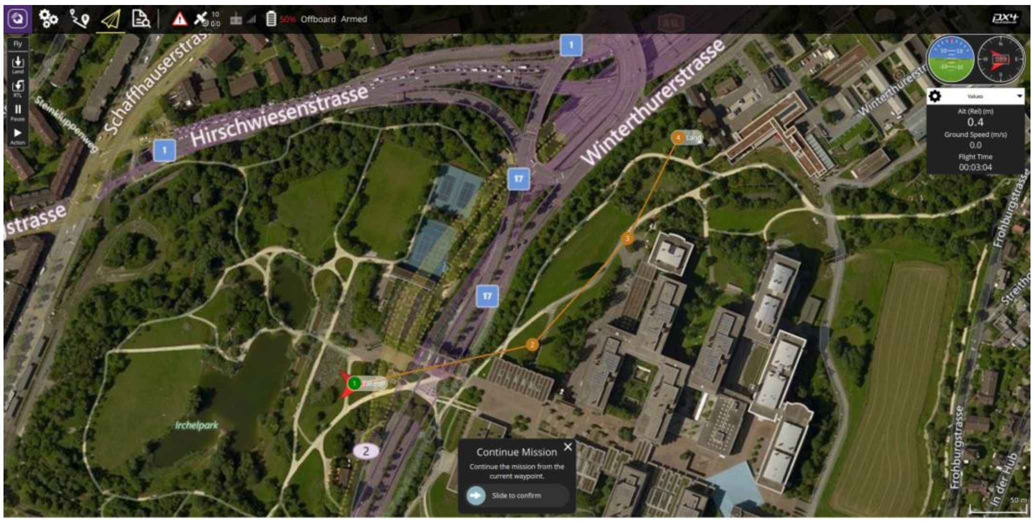


Figure 2.2. QGroundControl as ground control station (GCS).

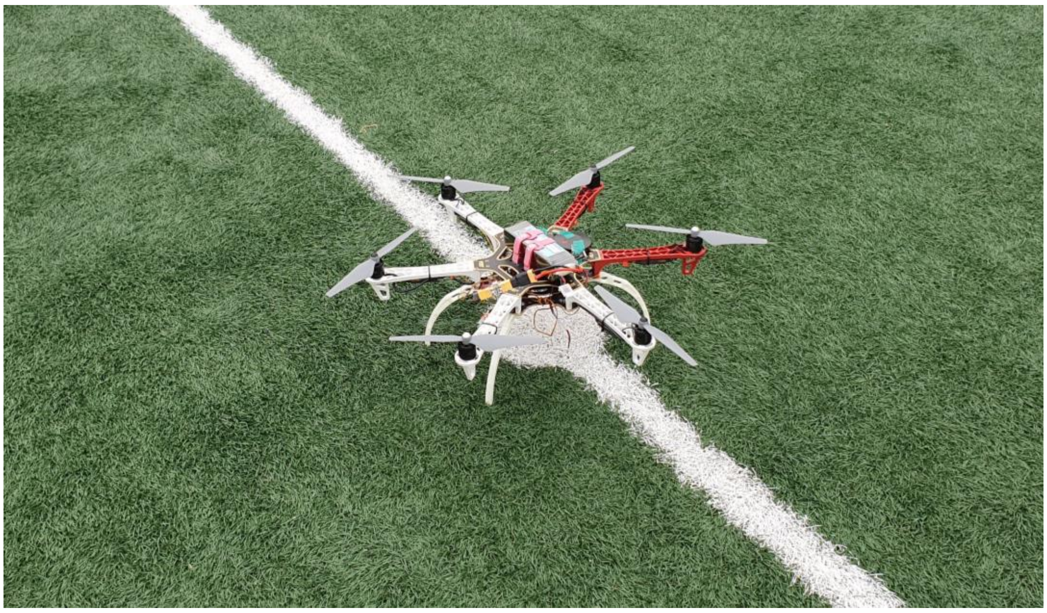


Figure 2. 3. Unmanned aerial vehicle (UAV).

2.2 Robot Operating System (ROS)

As mentioned earlier, ROS is the middleware for robot software development [13]. Unlike the operating systems used in computers, it provides services such as hardware abstraction, low-level device control, and message delivery between processes for system operation. For asynchronous communication in ROS, the publisher–subscriber model is adopted; topic is used for communication between the publisher and the subscriber. Figure 2.4 shows the structured model of ROS. The ROS consists of the master, publisher, and subscriber node. The master node connects the subscriber node to the publisher node that wants access to a specific topic. With the help of the master node, connected publisher and subscriber nodes will be able to send and receive the desired data through topic.

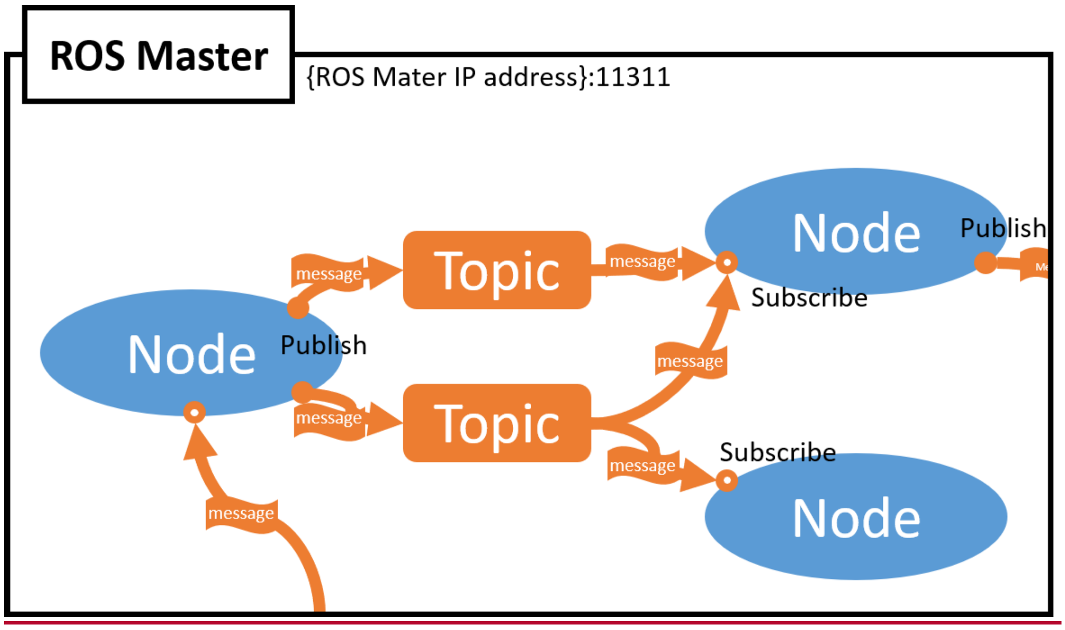


Figure 2. 4 Robot Operating System structure.

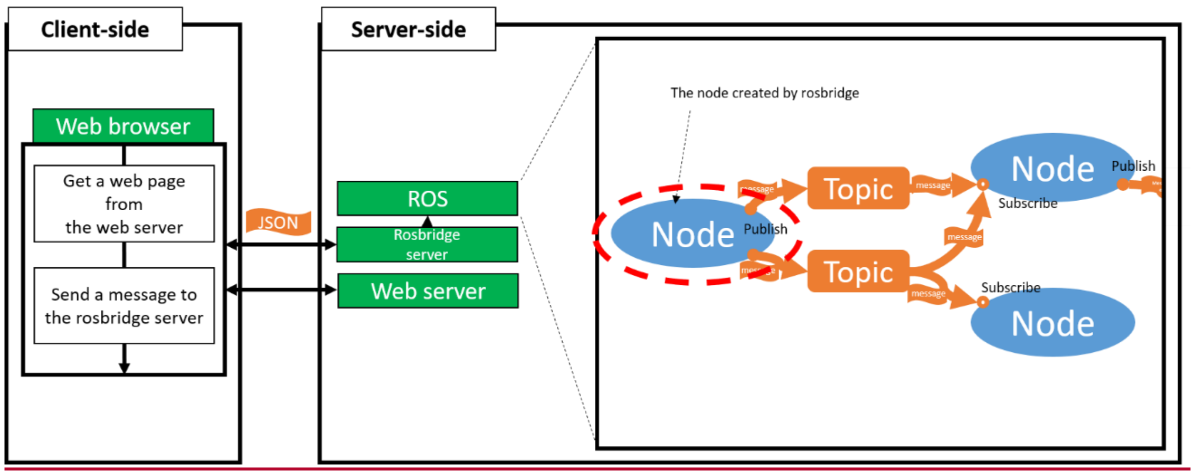


Figure 2. 5 Rosbridge concept

2.3 Rosbridge

Rosbridge is a package of ROS that allows us to use topics and services in ROS even if the client does not have ROS installed. This is possible because JSON-based rosbridge protocol is used on the server with ROS installed. When a rosbridge server that communicates with WebSocket on the ROS server side is executed, it is possible to communicate with the node of the ROS server through various front-end devices such as web browser, and the service is also available. Figure 2.5 describes the concept of rosbridge [14].

2.4 Safety tool of ROS

To ensure the safe operation of ROS, there are several services that are provided by ROS:

1. Network Security [15]

The ROS team is aware that because of the nature of the current system, the system can be attacked by vulnerabilities in the network. To address this, they proposed a method which is not a direct function of ROS, but a part of the configuration of the network used for ROS. They suggest not disclosing the ROS master and restricting access to the network. There are two strategies for achieving this. The first method involves restricting hosts that can access the system. For example, there are ways to create isolated networks or use firewalls. The second method involves giving orders to authenticate users before allowing them access to the system. However, these methods are not implemented within ROS; rather the role of protecting ROS is given to network settings outside ROS.

2. Message filter [16]

A message filter passes only certain messages on a one-on-one basis and blocks other messages. Several functions exist to implement messaging filters. First, there is a subscriber that acts as a top-level filter; it forwards messages from ROS to connected filters. Second, the time synchronization filter serves to synchronize to the same number of channels by referring to the headers of the receiving channels by time stamp. The third function is the time sequencer. The time sequencer filter ensures that callbacks are made in temporal order according to the header timestamp of the message.

By default, message filtering is performed through these three functions. When operating a robot system, the corresponding node may not be able to process the message on time owing to various factors at the time the message was generated. When operating a robot that requires time-sensitive command input, the message filter allows the message to be processed sequentially.

3. Watchdog timer [17]

Watchdog timer is implemented in ROS; it is a tool for high reliability systems. The watchdog timer monitors the CPU and restores the system to normal conditions when abnormal or infinite loops occur. While ROS provides a Watchdog Timer for these functions, it only provides detection function and entrusts developers with way to reconfigure the system.

4. Managed nodes in ROS2 [18]

The subsequent version of ROS, ROS2, introduced the concept of a management node, also called lifecycle node. There are four node states: Unconfigured, inactive, active, and finalized. Seven switching actions can be performed: create, configure, cleanup, activate, deactivate, shutdown, and destroy. When switching operation is performed, it goes through six switching states: configuring, cleaning up, shutting down, activating, deactivating, and error processing. This node state transition is introduced to enhance the overall stability of ROS.

5. ROS 2 DDS-Security integration [19]

Recently, ROS2 was officially released, and the biggest difference and feature of the previous version is the adoption of Data Distribution Service (DDS) as middleware. DDS requires several security requirements, including authentication, access control, and cryptographic. This shows that security is important for mission-critical ROS environments. However, since ROS and ROS2 are incompatible with each other in a native environment, security issues still remain in systems using existing ROS.

III. VULNERABILITY DEFINITION OF UAV USING ROS

This section introduces the types of network attacks occurring in CPS and the vulnerabilities that exist in the communication mechanisms of UAVs currently using ROS.

3.1 Vulnerability in CPS

Unlike traditional embedded systems that operate individually, CPS operates on the close interaction of computing and physical systems. Ultimately, CPS can be seen as an integration of computational, networking and physical processes. Embedded computers and networks monitor and control physical processes through feedback loops; these physical processes affect calculations. A key element in CPS in which this interaction is considered is an information and communication technology (ICT) component as a communication medium, which connects the computing and physical elements by information exchange [20]. CPS is widely used in many areas closely related to industrial control systems, advanced communications, smart power grids, transportation networks, vehicles and social networks. The UAV network incorporates communication devices, computing functions and control modules to form a single closed loop from data recognition, information exchange, decision making to final execution and can be referred to as CPS.

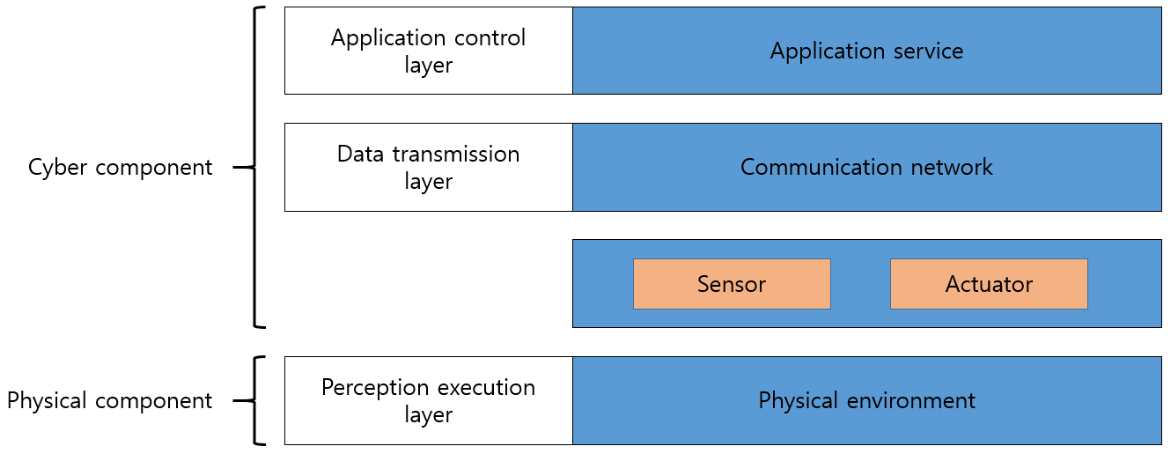


Figure 3. 1 Three-layer model of CPS

The CPS can be divided into three layers as shown in Figure 3.1 [21]. The recognition execution layer consists of physical elements such as sensors and actuators. The data obtained from the sensor is passed to the application through the data transmission layer, and the feedback is applied to the physical environment. Based on the data received from the data transmission layer, the application control layer calculates and assesses the information required by the system and then gives feedback to the data transmission layer. The data transmission layer connects the recognition execution layer and the application control layer, and performs information delivery.

The data transmission layer transmits the data necessary for system operation over the communication network. However, when using a communication network, an attacker may destroy the system with malicious behavior using the network. The CPS is made vulnerable due to the accessibility of various communication networks, including network access methods, equipment and software, and depending on the time and location of the CPS application. Successful exploitation of the data transmission layer would allow an attacker to tamper with the system and eavesdrop on information in that system; the attacker may even paralyze the system's function. For this reason, the data transmission layer is also used as a path for attacks from other layers. Network attacks on the data transmission layer use the following methods: man-in-the-middle attack, masquerade, and denial of service [22-24].

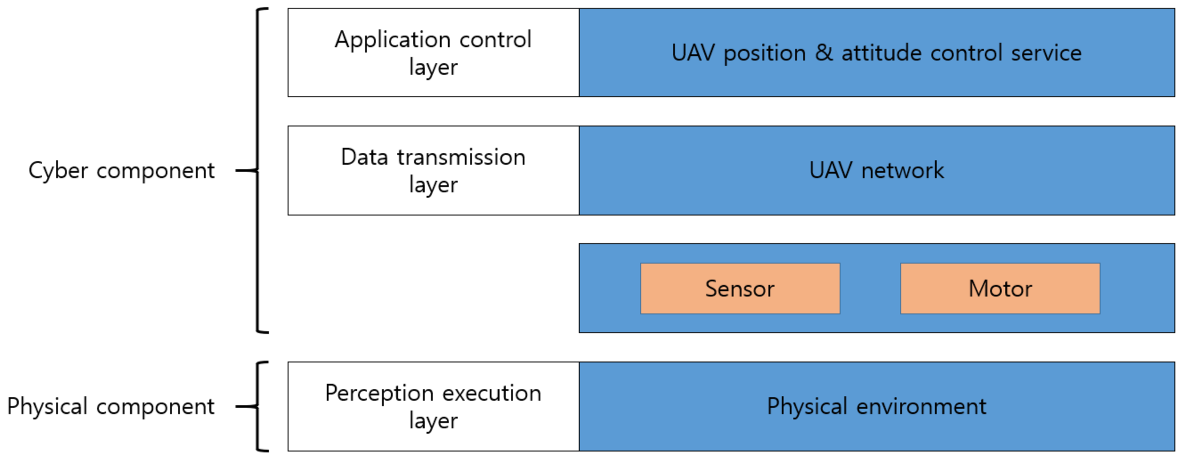


Figure 3. 2 Three-layer UAS framework in CPS perspective

3.2 UAS 3-layer model in CPS perspective

The components and structure of the UAS can be designed from a CPS perspective. Figure 3.2 is a three-layer framework of CPS for computing, communication and control of UAS. UAV sensors are hardware such as GPS and gyroscope that are used for measuring UAV flight status. An actuator in UAV is a motor that physically gives UAV movement in a three-dimensional environment according to the controller signal. A communication network is a wireless network environment in which UAVs send and receive UAV status and commands during their flight. The component corresponding to the application control layer is the flight position and altitude controller, which calculates the flight control of the next loop based on the status of the UAV received from the sensor.

UAS, where each component consists of these frameworks, is as vulnerable to system attacks through the data transmission layer as CPS. Successful exploitation could result in eavesdropping and tampering with the data of the system to neutralize it.

3.3 Model of ROS-based UAS

Advanced operations in UAVs, such as autonomous and clustered flights, require hardware and software capable of additional computing functions as well as flight controllers. The ability to provide additional assistance to the flight using information and computing power outside the flight controller is called offboard mode. The UAS assumed in this paper has a structure in which offboard computers are connected to the flight controller and communicate with each other. In addition, offboard computers support communication between flight controllers, external sensors and off-board computers via ROS. Figure 3.3 schematizes the UAS with the aforementioned communication architecture from a CPS perspective. As with CPS, this figure incorporates plant, sensor, controller, and actuator to form a closed loop from data recognition, information exchange, decision making to final execution. The red box indicates the part to which ROS is applied.

ROS is middleware for the development of robot software. This allows the configuration of UAS. The ROS adopted a pub-sub model for communication between each component that forms the robot. It is a structure in which two nodes, which exchanged node information with the help of Master node, send messages through the publishing and subscribing functions as needed. ROS provides MAVROS, a MAVLink expandable communication node. This allows the UAV to receive the data needed for the flight over the ROS.

Figure 3.4 shows the system model of UAV with ROS through the aforementioned procedure. The /sensor node present in the external sensor publishes the message to the /process node in the offboard computer using /topic. /Process nodes deliver command messages for UAV control to /MAVROS node based on sensor data. /MAVROS node forwards the data to the flight controller via MAVLink.

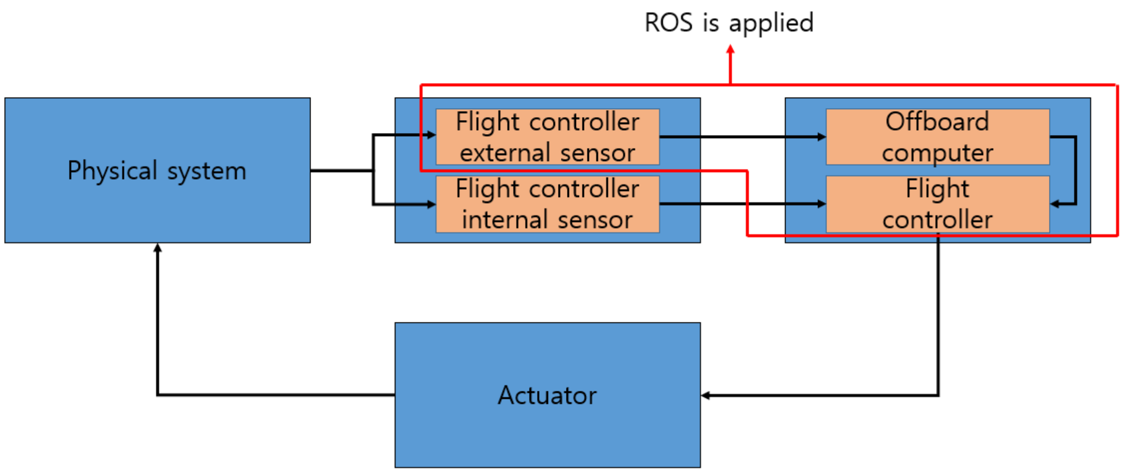


Figure 3. 3 Feedback loop in UAS

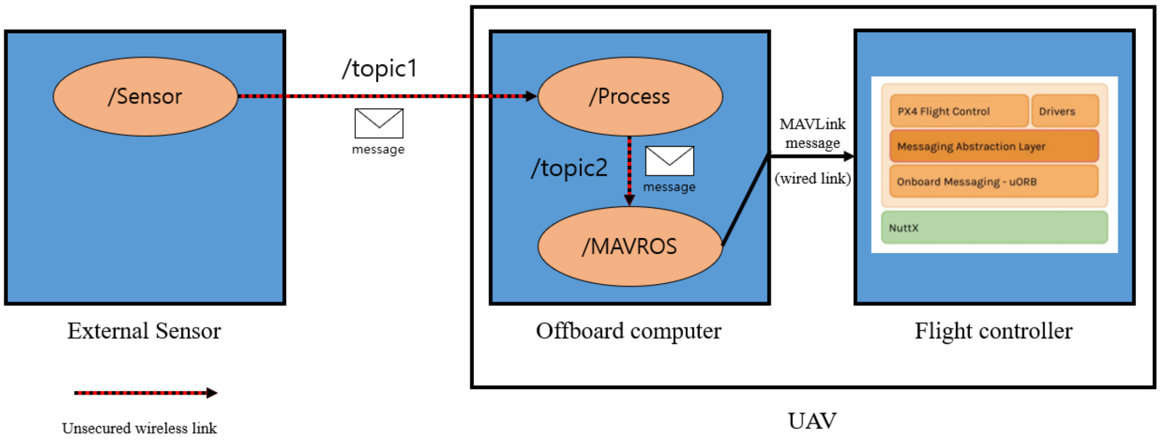


Figure 3. 4 System model of UAV with ROS

Table 1. The terms used in the ROS.

Term	Concept
T	Topic
P	Publisher to send information about a particular topic
S	Subscriber to receive information about a particular topic
A	Attacker Node
msg_{PST}	Message between P and S for T
msg_{P_AST}	Message between P_A and S for T
msg_{PS_AT}	Message between P and S_A for T

3.4 Vulnerability of ROS-based UAS

Table 3.1 defines the terms for each component in the ROS.

As explained in Section 2.4, ROS does not have fundamental security elements; hence, malicious nodes other than normal nodes can easily be connected. It is easy to break into the network; moreover, there is a possibility of masquerade attacks and false data injection. Once P_A is able to configure communication with S about T through the master node, the false data, msg_{P_AST}, can be sent to S. If the attack is made, the flight controller will not be able to control the exact position and altitude in a given environment, causing the system to be destroyed. There are three reasons why such attacks are possible.

First, the master node does not check whether it is a normal node or a malicious node for the node that makes the request. This allows an attacker to gain unauthorized access to ROS. Hence, this enables attacks such as eavesdropping or masquerade.

Second, the master node does not check whether data from connected nodes through monitoring is within the acceptable range of the system. In any command, dropping data that exceeds a user-defined threshold protects the system from malicious false data injection. However, when an attack is authorized within the scope, it cannot be defended.

Third, it does not guarantee the integrity of messages transmitted in ROS.

If the system ensures that the data is authorized and not changed, it can protect itself against active attacks such as masquerade and injection attacks. The second problem can be covered if the system is satisfied with the data integrity. Checking data integrity can also protect the system against masquerade attacks.

The most effective attacks on the system in UAS operation are those of active attacks that change the system: masquerade, injection, replay, etc. Active attacks hurt integrity and availability and can directly affect the flight of an UAV in a short time. To protect the system against this, we need a solution that can solve the aforementioned problems. In addition, the method should not have large overheads and not obstruct the flight. We propose a security framework that addresses vulnerabilities in ROS-based UAS and has less overhead.

IV. RELATED WORK

This section discusses the studies that have been conducted to ensure the stability of ROS-based systems. For each study, the method and direction of security application for authentication, authorization, and message verification areas are discussed.

Jeff Huang et al. [25] proposed ROSRV, a runtime verification framework for ROS-based robot applications. A node called ROSRV is placed under the master node. The node that needs to be registered as a publisher or subscriber node is identified and is connected to the other node. The second function then places the monitoring node between all publishers and subscribers within the ROS to drop commands or messages outside the user-specified range. The two functions satisfy the authorization and message verification. Thus, this can address the first and second vulnerabilities described in Section 3.4 at present. However, there are some limitations to this solution. First, for monitoring purposes, the monitoring node between the publisher and the subscriber verifies the message. This will take twice the transmission time in the existing publisher–subscriber model. The second is that if a large number of nodes are connected to a centralized ROSRV, there will be a delay in the monitoring node. In addition, if the data are modulated within the monitoring range, it will not be detected. The absence of two overheads and data integrity makes it difficult to apply ROSRV to UAS.

Russell Toris et al. [26] proposes rosauth, an authentication service to enhance the security of the connection of non-native clients in ROS. There is a package called rosbridge in ROS. This allows clients to communicate synchronously and asynchronously with ROS, even if not in an ROS environment. The author proposes a method of authenticating whether the client accessing the ROS server using the message authentication code (MAC) is an authorized node. This project can solve the first vulnerability mentioned in Section 3.4. However, the nodes are verified using MAC only at the point of client connection. It does not guarantee security for message tampering that occurs after the connection. In other words, it does not guarantee data integrity, the most important vulnerability in ROS-based UAS.

Bernhard Dieber et al. [27] treated ROS as a black box and used an authentication server (AS) to ensure communication between authorized nodes. Publisher receives a key from AS, encrypts and signs it with the message, and forwards it to the subscriber. The subscriber can decode and verify whether the message has been tampered with. However, every time we send a message, we have two encryption overheads and a decryption overhead. Furthermore, RSA signatures are slow.

Roland Dóczi et al. [28] claims to protect the system through certification and authorization on medical surgical robots subject to ROS. The author uses authorization and authentication (AA) to eliminate security problems arising from ROS. They implemented AA node for AA function. The node receives its name and password from the connection request node, checks the DB, and passes the key if it is the correct information. The node then requests the master node to connect with the other node along with the key; the master node sends the key to AA to verify that it is a valid node. However, methods of authenticating using names and passwords can easily be overridden by attackers.

V. PROPOSED METHOD

The work is carried out in an environment with MAVROS, an extension package for UAV in ROS (see Section 3 for details). We have found that the vulnerability of ROS makes ROS-based UAS vulnerable. To solve this, we implement security measures in the master, publisher, and subscriber. This does not address all security issues in the system, but it ensures that the following security issues that are key to UAS operation are dealt with(see Section 4 for details):

- Unauthorized users register nodes on the system without permission
- Unauthorized registered node infuses incorrect data and affects drone flight

Message transmission in the current ROS has the following procedure. There is node S that receives information about a particular topic T. P trying to transmit information about T attempts to connect with the node that receives the T through the master node. Master connects P to S. P broadcasts msg_{PST} and delivers it to S. S receives the information and performs calculations to control the UAV. The procedure unconditionally trusts the node and operates the robot. Thus, if the P_A, the node that publishes the wrong data, accesses the system, the following occurs. P_A requests a connection to the master node with a node that receives information for a specific topic T. The P_A connected to the system injects the wrong data, msg_{PAST}, into the S at a faster rate than P. S does not recognize that the data is incorrect and uses that data to control UAVs. The current procedure cannot determine whether the node that requests registration to the master is an authorized node. It is also not known whether the data that is being transmitted is modulated or is from an accredited node. For this reason, we would like to propose a security framework for ROS-based UAS to implement a UAS that is safe from such intrusions. ROS with frameworks can be schematized as shown in Figure 5.1. Table 5.1 defines the terms for each component of the proposed framework.

Table 2. The terms used in the framework.

Term	Concept
AC_T	Access list for P and S accessing specific topic T
d(x)	Digest for x
H(k, msg)	Getting a hash of msg using a key k.
P_{name}, S_{name}	Node name of P and S

5.1 Registration of a new node

Access control is the function of allowing or denying someone use of a resource. We apply access control to ROS, preventing unauthorized system registration of nodes.

AC_T means a list of access rights for nodes accessing a particular topic T. This includes nodes with access to T and can be expressed as AC_T = [x, y, z]. d(P) and d(S) mean digests for P and S respectively, and d(P) can be expressed in H(k, Pname||T). The ROS with access control registers the node using the following procedure:

① All publishers and subscribers accessing specific Topic T before ROS operation are listed and recorded in AC_T. The information recorded in AC_T is d(P) and d(S), which are digests of P and S. The reason for recording Digest is to make it impossible for an attacker to masquerade itself as a node that sees digest and has authority over T.

② P requests the master node to master node as a publisher of T.

③ The master node obtains d(P), which is the digest for P.

④ Master checks if the digest is in AC_T. If there is digest in AC_T, P is allowed to publish to T.

Figure 5.1 is a diagram showing the registration procedures of the ROS with added access control.

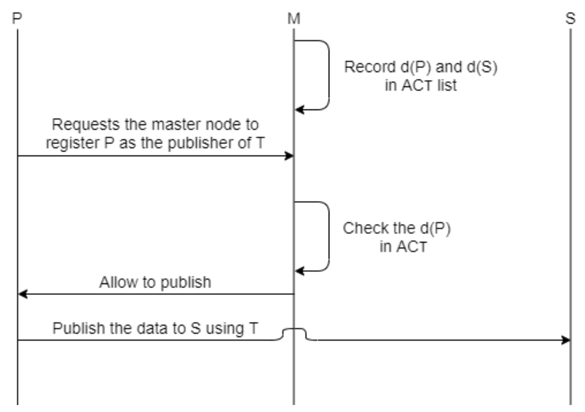

P
M
S
Record d(P) and d(S) in ACT list
Requests the master node to register P as the publisher of T
Check the d(P) in ACT
Allow to publish
Publish the data to S using T

Figure 5. 1 Proposed access control procedures

5.2 Signature with HMAC

A digital signature is a security tool that uses encryption for data integrity, authentication and non-repudiation. Generally, when key sharing is not possible, we use a digital signature using RSA. This means that when a message is signed and sent by the private key, the receiver verifies the message with the public key. In addition to RSA, there is also a signature method using the hash-based message authentication code (HMAC). This signature is characterized by sharing and using the same key between trusted parties. Although there are limitations in operating it in a network that is commonly used by the public, in a network that operates UAS, the signature method can be used if the keys are shared between nodes in advance. The overhead of signing using HMAC is significantly less than that of signing using RSA. For this reason, we will ensure data integrity using HMAC to sign messages delivered from ROS.

H(k, msg) obtains digests for msg using the key k. At this time, k should be exchanged between transceivers in advance. There are several types of hash functions, but SHA-256 was used in the solution. The hash function can be used as the agreed function between the sender and receiver. a||b means the connection between the letters x and y. For example, the result of x||y is "ab". Sign(k,msg) means digitally signing for message m using the key k. verify(s) means the verification process for signed data s.

The subscriber and publisher of the ROS, where the verification process has been added, sends and receives data through the following procedure:

① S and P for a specific topic T make a registration request to the Master node.

② P performs a signature on msg_{PST}. Sign(k, msg_{PST}), the signature procedure, means H(k, msg_{PST})||msg_{PST}.

③ P, which carried out the signature, sends s = Sign(k, msg_{PST}) to S.

④ S performs a verify(s) on the received s.

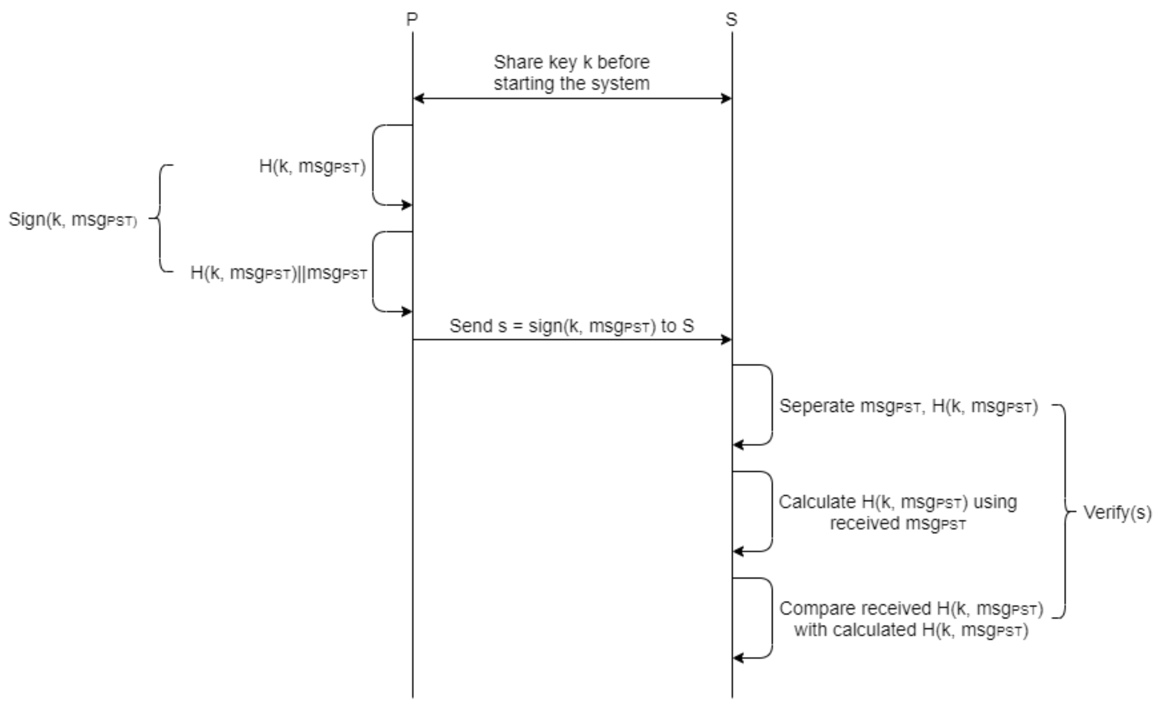
⑤ Separate s = H(k, msg_{PST})||msg_{PST} from H(k, msg_{PST}) and msg_{PST}.

⑥ For separated msg_{PST}, perform H(k, msg_{PST}) using a pre-shared key k. k is the same symmetric key used by P.

⑦ Compare the two H(k, msg_{PST}) and inspect them for the same value.

⑧ If there is no problem with msg_{PST}, use that data.

Figure 5.2 is a diagram showing the data transmission procedures of the ROS with the added verification.



P
S
Share key k before starting the system
H(k, msgPST)
Sign(k, msgPST)
H(k, msgPST)||msgPST
Send s = sign(k, msgPST) to S
Seperate msgPST, H(k, msgPST)
Calculate H(k, msgPST) using received msgPST
Compare received H(k, msgPST) with calculated H(k, msgPST)
Verify(s)

Figure 5. 2 Proposed signature procedures

5.3 Performance and Conceptual Comparison

A. Overhead

The proposed security framework has additional features to address vulnerabilities in existing ROS. The function results in additional overhead of data size and computation. First, the computational overhead that occurs during the execution of access control does not affect UAV operation because it occurs once upon initial connection. However, for signatures using HMAC, data overhead and computational overhead exist for each transmission. We implemented HMAC signatures using SHA-256 for demonstration. For example, the existing 69 bytes size geometry_msgs/PoseStamped message is 32 bytes due to HMAC signatures. Conversely, the overhead would be 256 bytes if RSA-2048 signatures were used, as shown in Section 4, related study [26]. Computation overhead refers to the time it takes to compute SHA-256. This was benchmarked using a crypto++ library and can handle 290 MiB per second when a 128bit key is used. This means processing 101 bytes of data, including data overhead, would take 0.0026 ms. For RSA-2048, it takes 0.061 ms for encryption and 1.225 ms for decryption. From these results, it can be observed that the overhead of the proposed security framework will have little impact on the performance of existing ROSs.

B. ROS with MAC

We ensured integrity through verification of the data transmitted within the system using MAC. Similarly, rosauth [25] in Section 4 wanted to use MAC to improve the security of ROS. However, its use is different from this study. In the previous work, MAC is used in a non-native environment early in the connection to enable clients to authenticate themselves with the server as validated clients. However, as there is no solution for the integrity of the data transmitted, the system will be breached if an attacker attempts an MITM attack on an already connected channel. Conversely, the framework proposed in this study uses MAC to ensure data integrity and authentication with each transmission since the beginning of the connection. In addition, only nodes authorized through access control can be registered. Furthermore, the proposed framework can be secured in a non-native environment, similar to their study. This is demonstrated in Section 6 with an experiment.

VI. TEST

This section describes an experiment that studies the consequences that can be caused by the vulnerabilities in existing ROS-based UAS and the impact on UAS after applying the security framework. First, we describe the experimental environment of the drones that make up UAS for the experiment and the arrangement of components. The results are presented with an explanation about the operation of the proposed security framework in a native ROS environment and a non-native ROS environment.

6.1 Experiment environment of UAS

Figure 6.1 shows the UAS environment configured for experimentation. Pixhawk is an industry standard autopilot developed and jointly developed by 3DR Robotics and Ardupilot Group. Various robots such as RC cars, airplanes, and multi-copters can be made and firmware is provided for them using Pixhawk. We made quadcopters which belong to a class of multi-copters and used them for the experiment. Pixhawk typically uses two firmware, ardupilot and PX4. We used PX4 firmware that supports offboard mode in the experiment

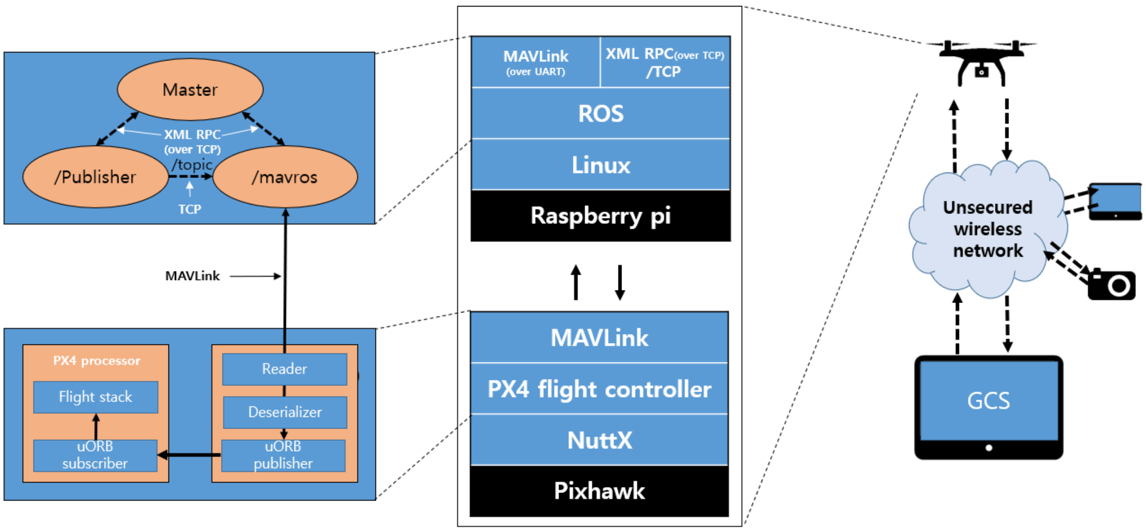


Figure 6. 1 Experiment environment

because we assumed UAS to operate advanced drones such as autonomous driving and cluster flight using offboard mode. Pixhawk uses the MAVLink protocol for communication. MAVLink is a light messaging protocol for on-board communication or components of drones. This can be implemented in 14 languages, including C and C++, and various high-level APIs exist for interaction between other systems such as drones and ROS. In this experiment, communication is made using MAVLink to the component computer for offboard mode. The companion computer used Raspberry pi which is an embedded Linux-based development small computer and Ubuntu MATE which is a Linux based OS was used in the computer. This has better compatibility than Raspberry pi on a variety of issues, such as packages and kernels, to use ROS. ROS stands for robot operating system, which is not similar to the conventional operating systems used in computers. It is a middleware concept for robot development that is installed on an OS such as Linux or Windows. We installed ROS Kinetic for the experiment. ROS supports node-to-node communication using XML-RPC and TCP. XML-RPC is an XML-based distributed system communication method that is simple and portable RPC protocol over HTTP. This is used in ROS by the Publisher and the Subscriber to communicate with the master node to connect with each other. When the publisher sends data for a particular topic after the connection, it serializes the data and sends it to TCP payload. The Subscriber receives the packet and receives the data by deserializing it. We use the MAVROS package which is an ROS expansion package. This package enables MAVLink communication between Raspberry pi and Pixhawk where ROS runs. Hence, the /mavros node, which receives data related to the flight from the publisher, forwards it to Pixhawk through the MAVLink protocol. Upon receiving this, Pixhawk calculates flight control from the flight stack based on the corresponding data.

The overall experimental environment is described above. We conducted the experiment in this experimental environment by considering two situations. In Figure 6.1, two computers and one sensor that are authorized can be found attached to the ROS through the wireless network. These devices can access the ROS in a native environment or, depending on the intention of the user, the ROS can be accessed in a non-native environment. In these two environments, the approach to ROS is as follows First of all, if the client is in a native ROS environment, the client has ROS installed and by running the launch file the client accesses the ROS server and creates a node. If the client is in a non-native ROS environment, the client does not have an ROS installed and requests the master to connect to the communication for a particular topic on the front end implemented with the roslibjs library. After connection, the client encodes the data in JSON and sends it to the rosbridge server. On the server side, rosbridge is run, which transmits data received by clients to nodes that subscribe to the topic.

6.2 Experiment on native ROS attack

This section describes the modes of attacks in native ROS and the ways that can be used to defend the native ROS through the proposed framework. The attacks in the environment are shown in Figure 6.2. First, the accredited device sends the data and commands necessary for the flight to /mavros via a specific topic. The drone performs normal flights based on their data. At this time, a malicious computer breaks into the network with the aim of sabotaging the system and then register the publisher with the ROS that transmits the /mavros/local_position/pose topic. An attacker could then influence the flight path of the drone by means of the corresponding topic. This experiment can be found in [29, 30].

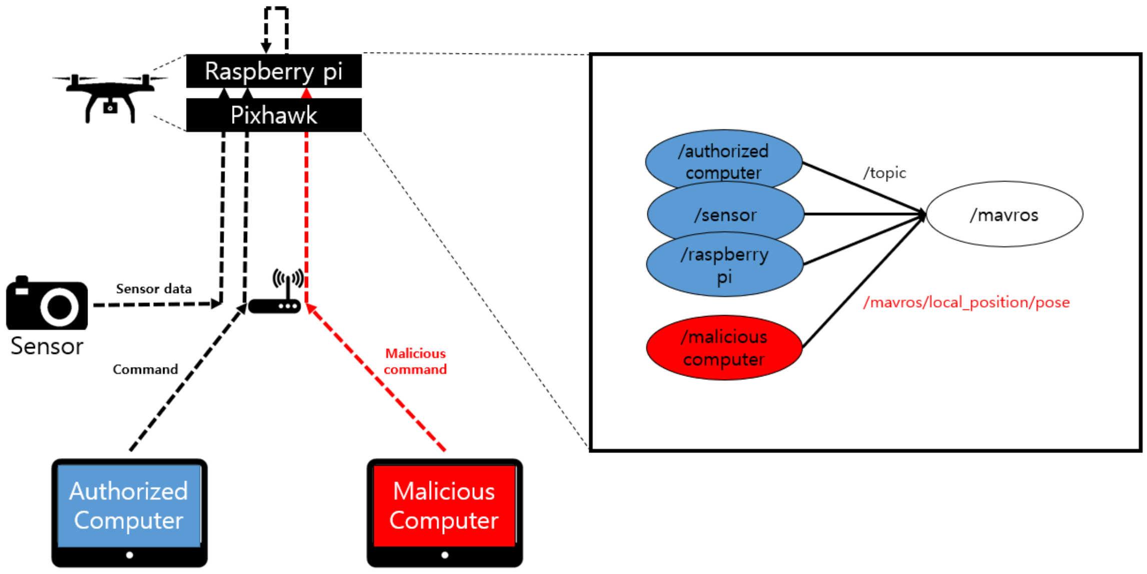


Figure 6. 2 Attack in native ROS

In the actual experiment, the experiment was conducted by flying a normal drone driving at a height of 2 m considering drone, property, and human casualties, and by returning the drone to its starting point. Figure 6.3 shows the state of UAV during an attack. The X and Y axis in Figure 6.3 denote time and the altitude of the drone, respectively. Up to the 30 second point in the figure, only the accredited node approaches the ROS and transmits the /mavros/local_position/pose topic, resulting in a 2 m high UAV flight. After that, the attacker node can then approach the ROS and inject itself into the UAV to fly the drone at an altitude of 0 m to confirm that the altitude of the UAV is slowly converging at 0 m.

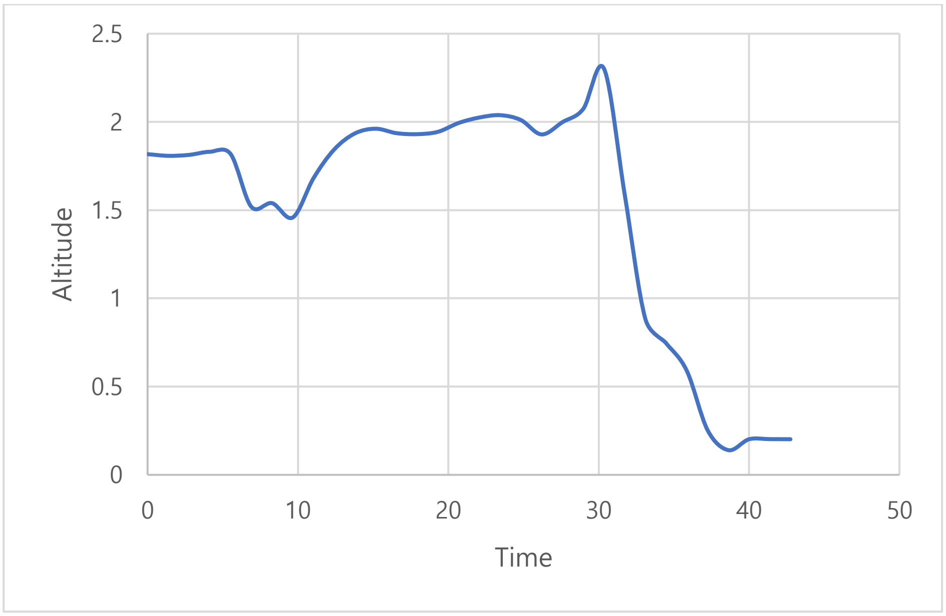


Figure 6. 3 UAV flight without security framework in native ROS

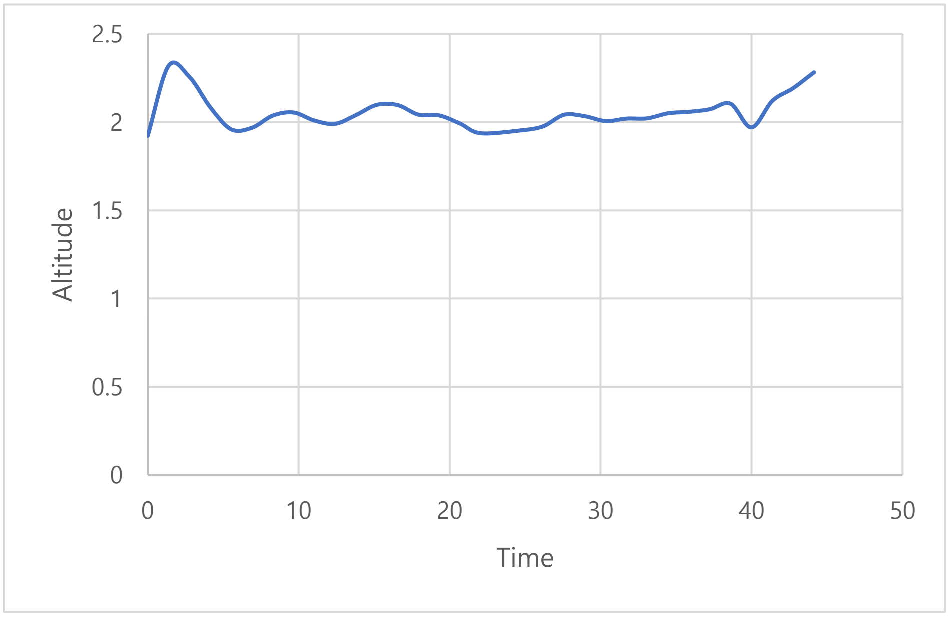


Figure 6. 4 UAV flight with security framework in native ROS

The reason for this attack is the lack of verification and data integrity for newly registered nodes. We apply a security framework to existing ROS to defend against such attacks. Figure 6.5 shows how HMAC is applied to ROS to send and receive data. The framework is applied to each computer running publisher and to the computer running MAVROS. We demonstrate through experiments that UAV with these security frameworks have no impact on existing methods of attack. Figure 6.4 shows the state of UAV during an attack in the same scenario as the above. An attack was made near 30 seconds, but it can be confirmed that the UAV flies at an altitude of 2 m until the experiment is over.

6.3 Experiment on non-native ROS attack

Rosbridge is a package that enables synchronous and asynchronous communication of ROS in an environment where ROS is not installed. We have implemented a web client that can use ROS using the rolibjs library for experiments. At this time, the server side should run a rosbridge to create a node that sends data to the mavros. Figure 6.6 briefly describes the composition of the rosbridge. For practical use of rosbridge, three applications must be run on the server-side. The first is the ROS, and the second is the rosbridge server. When the rosbridge server receives a message from the client from which topic to send, it attempts to connect with the subscriber receiving the topic. In Figure 6.6, the corresponding subscriber

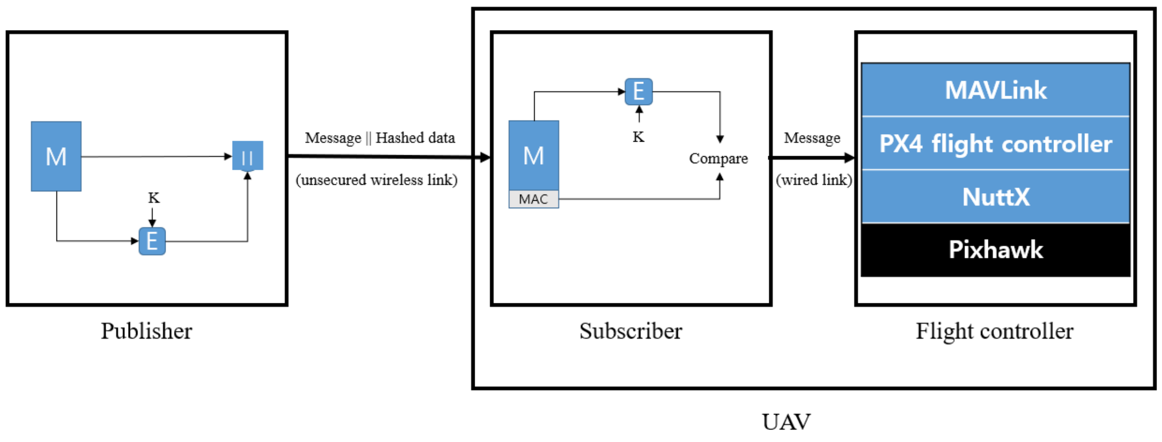


Figure 6. 5 Security framework for ROS

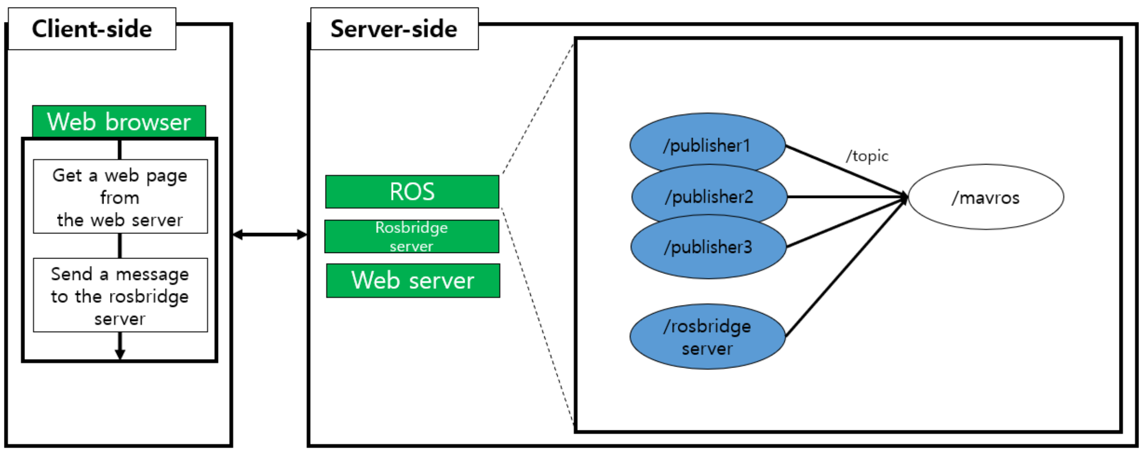


Figure 6. 6 Rosbridge composition diagram

is /mavros. The third application is web server. The Web server provides roslibjs services to clients via web pages and helps them communicate indirectly with ROS through rosbridge. These three applications do not necessarily have to run on one computer, and they can also run on multiple server computers. This will allow the client without ROS to communicate with the ROS installed computers. This experiment can be found in [31, 32].

The attack and defense experiments in non-native environments, as in previous experiments, proceeded with an attack that lowered UAV flying at certain altitudes to 0 m and a scenario that defended them. Figure 6.7 shows the method of attack in a non-native ROS environment. An attacker can execute an unauthorized web server to execute a malicious node on the ROS through the rosbridge server. These nodes can break UAS by injecting incorrect data into the system, such as malicious nodes in a native ROS environment. Figure 6.8 is a web page received through a Web server, and when an altitude is entered in a textbox, the UAV has constructed an experimental environment that flies at that altitude. Figure 6.9 shows the status of the UAV affected by the corresponding attack method. Upon receiving /mavros/local_position/pose topic data from normal web clients, the UAV flies at an altitude of 1-2 m during 10 seconds. After 10 seconds, an attacker uses rosbridge to connect a malicious node to the ROS and inject false data to lower the altitude of the UAV. We performed data integrity and node verification by applying the HMAC-based security framework to Web servers and MAVROS. Experiments show that existing attack methods have no impact on UAVs with that method. Figure 6.10 shows the state of UAV when an attack is made in the same scenario as earlier. An attack was made near 10 seconds, but UAV can confirm that it performs a highly normal flight of 1–2 m until the experiment is over.

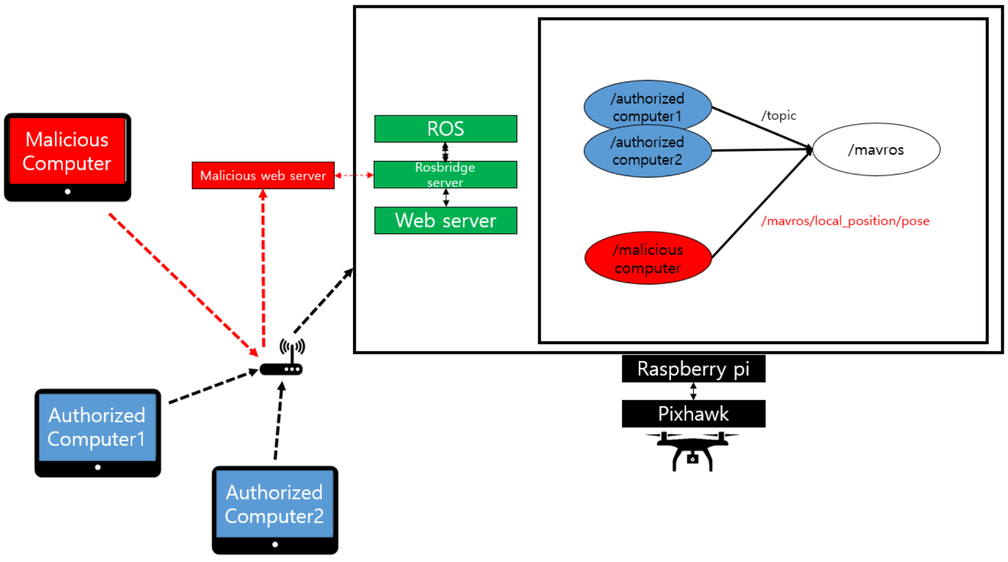
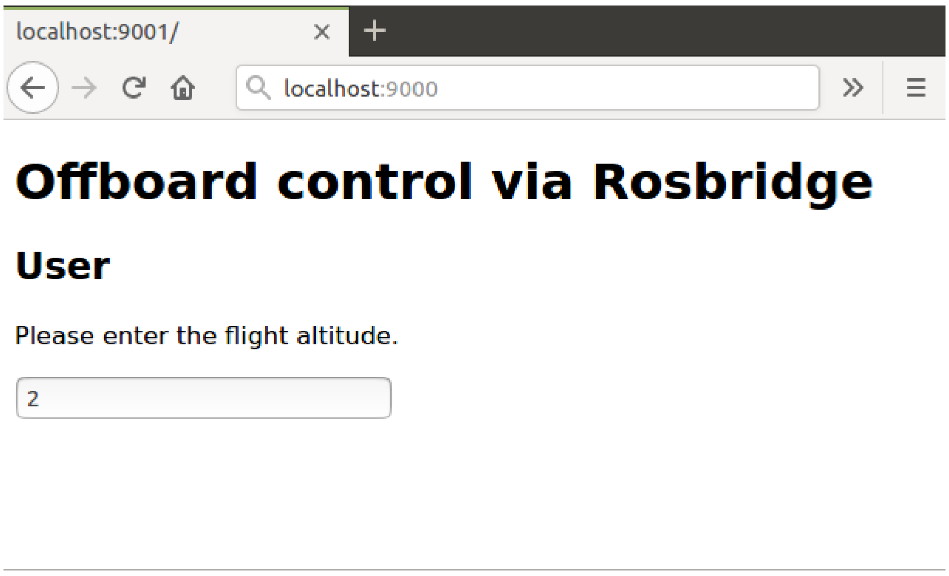


Figure 6. 7 Attack in native ROS



Offboard control via Rosbridge

User

Please enter the flight altitude.

2

Figure 6. 8 Web client

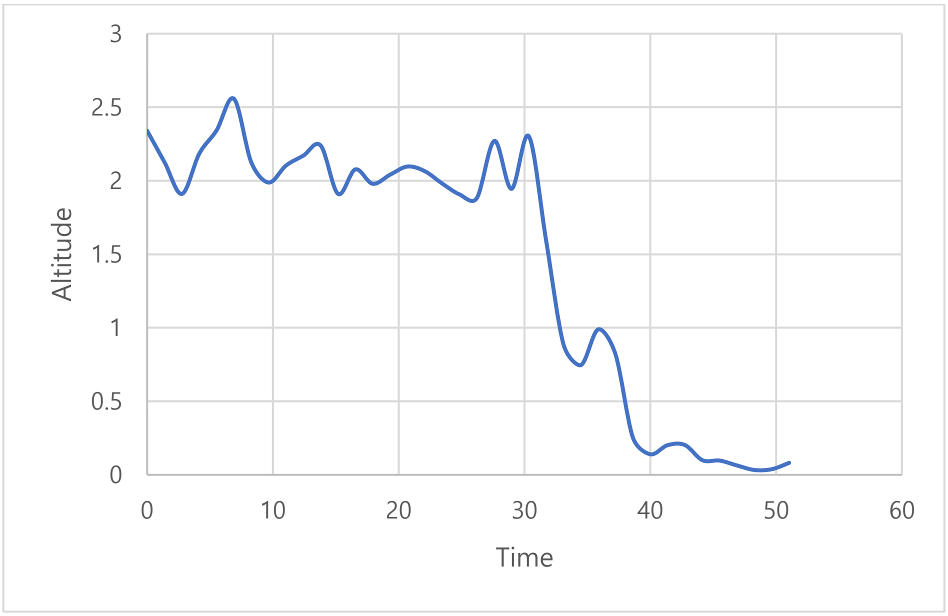


Figure 6. 9 UAV flight without security framework in non-native ROS

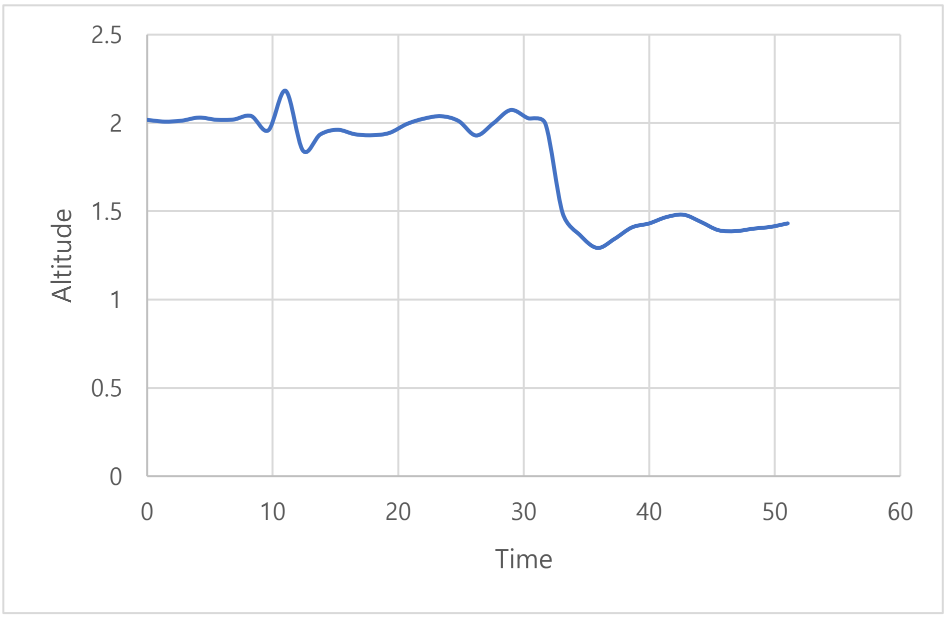


Figure 6. 10 UAV flight with security framework in non-native ROS

VI. CONCLUSION

With the noticeable growth in the use of UAV, the stability of the system has become a major concern in recent years. Due to the absence of system stability, UAVs that are applied in diverse places are exposed to potential risks. Therefore, it is necessary to be aware of this fact and study the stability of the system of UAVs.

For advanced operation of UAVs that make up UAVs, such as autonomous and clustered flights, computers that can be operated and communicated are required in addition to the flight controller, which is referred to as offboard systems. UAS is a generic term for controls, communications equipment, etc. to operate UAVs, including UAVs, and falls under the category of CPS. We explain the vulnerability of UAS flying offboard in terms of CPS and present a security framework to address it. The framework ensures the integrity of the data transmitted in the system through digital signatures and prevents unauthorized nodes from accessing the system without authorization, hiding their identities. By measuring overhead for computations, data, and transmission speeds as the framework's functions are added, it is shown to be an appropriate framework for UAVs.

In this study, the real time experiment shows that the UAS fails to function properly through cyber-attacks using the vulnerability of the ROS and installing ROS in the offboard computer. This can establish stability by addressing the problem of access control to ROS and integrity problems. To address this, the proposed security framework is applied to the system to demonstrate system stability through practical experimentation.

References

[1] Khan, M. A.; Ectors, W.; Bellemans, T.; Janssens, D.; Wets, G. UAV-Based Traffic Analysis: A Universal Guiding Framework Based on Literature Survey. Transportation Research Procedia 2017, 22, 541–550.

[2] Vacca, G.; Dessì, A.; Sacco, A. The Use of Nadir and Oblique UAV Images for Building Knowledge. ISPRS International Journal of Geo-Information 2017, 6, 393.

[3] Kang, J.-H.; Kwon, Y.-M.; Park, K.-J. Cooperative Spatial Retreat for Resilient Drone Networks. Sensors 2017, 17, 1018.

[4] Bithas, P. S.; Michailidis, E. T.; Nomikos, N.; Vouyioukas, D.; Kanatas, A. G. A Survey on Machine-Learning Techniques for UAV-Based Communications. Sensors 2019, 19, 5170.

[5] Handwerk, B. surprising drone uses (besides Amazon delivery). National Geographic. 2013.

[6] Wang, E. K.; Ye, Y.; Xu, X.; Yiu, S. M.; Hui, L. C. K.; Chow, K. P. Security Issues and Challenges for Cyber Physical System. In 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing; IEEE, 2010.

[7] Kwon, Y.-M.; Yu, J.; Cho, B.-M.; Eun, Y.; Park, K.-J. Empirical Analysis of MAVLink Protocol Vulnerability for Attacking Unmanned Aerial Vehicles. IEEE Access 2018, 6, 43203–43212.

[8] Zhao, N.; Li, Y.; Zhang, S.; Chen, Y.; Lu, W.; Wang, J.; Wang, X. Security Enhancement for NOMA-UAV Networks. IEEE Transactions on Vehicular Technology 2020, 69, 3994–4005.

[9] Hartmann, K., & Steup, C. The vulnerability of UAVs to cyber attacks-An approach to the risk assessment. In 2013 5th international conference on cyber conflict (CYCON 2013), 2013, 1-23.

[10] Yoon, K., Park, D., Yim, Y., Kim, K., Yang, S. K., & Robinson, M. Security authentication system using encrypted channel on uav network. In 2017 First IEEE International Conference on Robotic Computing (IRC), 2017, 393-398.

[11] MAVLink. 2020. Available online: https://mavlink.io/en/ (accessed on 17 December 2020)

[12] MAVLink format. 2020. Available online: https://mavlink.io/en/guide/serialization.html (accessed on 17 December 2020)

[13] ROS. 2020. Available online: https://www.ros.org/core-components/ (accessed on 17 December 2020)

[14] Rosbridge. 2020. Available online: http://wiki.ros.org/rosbridge (accessed on 17 December 2020)

[15] ROS security. 2020. Available online: http://wiki.ros.org/Security (accessed on 17 December 2020)

[16] ROS message filter. 2020. Available online: http://wiki.ros.org/message_filters (accessed on 17 December 2020)

[17] ROS watchdog timer. 2020. Available online:

http://library.isr.ist.utl.pt/docs/roswiki/watchdog_timer.html (accessed on 17 December 2020)

[18] ROS2 lifecycle. 2020. Available online: https://design.ros2.org/articles/node_lifecycle.html (accessed on 17 December 2020)

[19] ROS 2 DDS-Security integration. 2020. Available online: https://design.ros2.org/articles/ros2_dds_security.html (accessed on 17 December 2020)

[20] Park, K. J., Zheng, R., & Liu, X. Cyber-physical systems: Milestones and research challenges. 2012

[21] Cao, L.; Jiang, X.; Zhao, Y.; Wang, S.; You, D.; Xu, X. A Survey of Network Attacks on Cyber-Physical Systems. IEEE Access 2020, 8, 44219–44227.

[22] Sztipanovits, J.; Koutsoukos, X.; Karsai, G.; Kottenstette, N.; Antsaklis, P.; Gupta, V.; Goodwine, B.; Baras, J.; Shige Wang. Toward a Science of Cyber–Physical System Integration. Proceedings of the IEEE 2012, 100, 29–44.

[23] Humayed, A.; Lin, J.; Li, F.; Luo, B. Cyber-Physical Systems Security—A Survey. IEEE Internet of Things Journal 2017, 4, 1802–1831.

[24] Abdi, F.; Chen, C.-Y.; Hasan, M.; Liu, S.; Mohan, S.; Caccamo, M. Preserving Physical Safety Under Cyber Attacks. IEEE Internet of Things Journal 2019, 6, 6285–6300.

[25] Huang, J.; Erdogan, C.; Zhang, Y.; Moore, B.; Luo, Q.; Sundaresan, A.; Rosu, G. ROSRV: Runtime Verification for Robots. In Runtime Verification; Springer International Publishing, 2014; pp. 247–254.

[26] Toris, R.; Shue, C.; Chernova, S. Message Authentication Codes for Secure Remote Non-Native Client Connections to ROS Enabled Robots. In 2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA); IEEE, 2014.

[27] Dieber, B.; Kacianka, S.; Rass, S.; Schartner, P. Application-Level Security for ROS-Based Applications. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); IEEE, 2016.

[28] Doczi, R.; Kis, F.; Suto, B.; Poser, V.; Kronreif, G.; Josvai, E.; Kozlovszky, M. Increasing ROS 1.x Communication Security for Medical Surgery Robot. In 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC); IEEE, 2016.

[29] ROS-based UAV Attack Experiment in Native Environment. Accessed: Dec. 24, 2020. [Online]. Available: https://youtu.be/m6oT---Y36Q

[30] ROS-based UAV Framework Experiment in Native Environment. Accessed: Dec. 24, 2020. [Online]. Available: https://youtu.be/MUmTsNmxMsM

[31] ROS-based UAV Attack Experiment in Non-Native Environment. Accessed: Dec. 24, 2020. [Online]. Available: https://youtu.be/ODzQ1fQpUwE

[32] ROS-based UAV Framework Experiment in Non-Native Environment. Accessed: Dec. 24, 2020. [Online]. Available: https://youtu.be/NgvpGi9mzhI

요 약 문

보안된 UAV 통신을 위한 로봇 운영 체제 프레임워크

본 논문은 ROS가 적용된 UAS에서의 취약점을 분석하고, 이를 방어할 수 있는 보안 프레임 워크를 제시한다.

최근 여러 상업매체 또는 군에서 동작하는 UAV 는 개별적으로 동작하는 전통적인 임베디드와 달리 컴퓨팅 시스템과 물리시스템이 밀접한 상호작용을 하며 동작한다. 이러한 특성을 갖는 UAV 를 운용하는 시스템인 UAS 는 CPS 의 한 범주에 속한다. CPS 는 계산, 네트워킹, 물리적 프로세스가 하나의 피드백 루프로 통합된 시스템을 의미한다. 이 중 네트워킹을 담당하는 데이터 전송 계층은 장소, 상황, 하드웨어, 소프트웨어 등 다양한 통신 네트워크의 접근성을 갖는 CPS 의 특성으로 인해 취약한 계층이다. 이러한 사실은 CPS 의 일부인 UAS 도 같은 취약점을 갖고 있음을 의미한다. 최근 UAV 의 사용은 단순한 비행 미션 수행뿐만 아니라 군집비행, 자율비행 등의 특수한 미션을 수행한다. 해당 미션을 위해서는 컴퓨팅 능력이 있는 하드웨어 및 소프트웨어가 UAS 에 추가 되어야한다. 그 소프트웨어 중 하나로 ROS 가 사용되며, 이는 보안적 요소의 부재로 공격자가 로봇 시스템을 망가뜨리는 것을 허용한다. 우리는 ROS 의 보안을 위해 기존에 연구된 내용을 조사하였으며, 해당 방법이 UAS 에 적용 가능한지 여부를 분석하였다. 우리는 시간에 민감한 UAS 에 적용할 수 있는 가벼운 오버헤드를 갖는 보안 프레임워크를 제안한다. 또한 ROS 를 활용할 수 있는 두 가지 접근법인 native ROS 와 non-native ROS 환경에서 실제 실험을 통해 보안 프레임워크의 성능을 검증한다.

핵심어: Network attack, Unmanned Aerial Vehicle (UAV), Security, ROS, MAVROS,