NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AIS CYBERSECURITY SYSTEM FOR REDUCING THE ATTACK SURFACE OF VOYAGE NETWORKS

by

Jorge Vasquez Jr.

December 2021

Thesis Advisor: Chad A. Bollmann
Co-Advisor: Darren J. Rogers

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2021	3. REPORT TYPE AND DATES COVERED Master's thesis

4. TITLE AND SUBTITLE AIS CYBERSECURITY SYSTEM FOR REDUCING THE ATTACK SURFACE OF VOYAGE NETWORKS	5. FUNDING NUMBERS
6. AUTHOR(S) Jorge Vasquez Jr.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Navy Cyber Warfare Development Group, Suitland, MD	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

U.S. Navy and commercial vessels use modern navigation technology consisting of computers and electronic systems that are highly interconnected and create a cyber terrain that is vulnerable to novel cyberattacks. Previous research proved that voyage networks are vulnerable to radio frequency attacks. One especially vulnerable component is the Automatic Identification System (AIS), a navigation and safety tool required on all vessels with a gross weight of 300 tons or greater. Previous security researchers were able to transmit data packets through the AIS receiver. The AIS blindly accepted packets as long as they followed ITU-R M.1371-5 standard protocol.

This work aims to design a low-cost AIS data validation system that will reduce the attack surface of voyage networks. In this work, we leverage the NMEA-0183 and ITU-R M.1371-5 standards to implement two cybersecurity strategies, allow-listing and validating inputs, based on the quality dimensions of the data. The threat models that this security system attempts to address are contact spoofing attacks and arbitrary data injection attacks. We believe that a minimalist security system that is standalone, is not resource intensive, and can handle large volumes of AIS traffic is necessary for an effective design. The system proposed in this work fulfills these objectives. The resulting security system is implemented and validated using Python.

14. SUBJECT TERMS maritime, cyber, cyber-security, cyber warfare, smart security, software-defined radio, radio frequency, automated identification system, NMEA 0183, voyage network, furuno, ITU-R M.1371-5	15. NUMBER OF PAGES 131
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

AIS CYBERSECURITY SYSTEM FOR REDUCING THE ATTACK SURFACE OF VOYAGE NETWORKS

Jorge Vasquez
Ensign, United States Navy
BS, Texas A & M University, 2020

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2021

Approved by: Chad A. Bollmann
Advisor

Darren J. Rogers
Co-Advisor

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

U.S. Navy and commercial vessels use modern navigation technology consisting of computers and electronic systems that are highly interconnected and create a cyber terrain that is vulnerable to novel cyberattacks. Previous research proved that voyage networks are vulnerable to radio frequency attacks. One especially vulnerable component is the Automatic Identification System (AIS), a navigation and safety tool required on all vessels with a gross weight of 300 tons or greater. Previous security researchers were able to transmit data packets through the AIS receiver. The AIS blindly accepted packets as long as they followed ITU-R M.1371-5 standard protocol.

This work aims to design a low-cost AIS data validation system that will reduce the attack surface of voyage networks. In this work, we leverage the NMEA-0183 and ITU-R M.1371-5 standards to implement two cybersecurity strategies, allow-listing and validating inputs, based on the quality dimensions of the data. The threat models that this security system attempts to address are contact spoofing attacks and arbitrary data injection attacks. We believe that a minimalist security system that is standalone, is not resource intensive, and can handle large volumes of AIS traffic is necessary for an effective design. The system proposed in this work fulfills these objectives. The resulting security system is implemented and validated using Python.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AIS	Automatic Identification System
A-SRT	AIS Signal Research Tool
BOEM	The Bureau of Ocean Energy Management
COG	course-over-ground
CPA	closest point of approach
CPU	central processing unit
DOD	Department of Defense
ECDIS	Electronic Chart Display and Information System
EEMS	Engineering Enclave for Maritime Security
GMSK	Gaussian minimum-shift keying
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IMO	International Maritime Organization
ITU	International Telecommunication Union
LWE	Lightweight Ethernet
ML	machine learning
MMSI	maritime mobile service identifier
NMEA	National Marine Electronics Association
NPS	Naval Postgraduate School

OSI	open system interconnection
RAM	random-access-memory
RF	radio-frequency
ROT	rate-of-turn
SDR	software defined radio
SOG	speed-over-ground
SOLAS	Safety Of Life At Sea
SOTDMA	self-organized time division multiple access
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UN	United Nations
USN	U.S. Navy
USRP	Universal Software Radio Peripheral
VHF	very-high frequency

Acknowledgments

I would like to thank my advisors, CDR Chad Bollmann and Darren Rogers, who showed me the ropes for conducting respectable research. They knew when to push me and when to pull me back. I could not have completed this without their resourcefulness and their wisdom.

I would also like to thank my beautiful fiancée, Evelyn, for providing me with a sense of peace even during the most stressful of times. Knowing my day would get better as soon as I talked to her encouraged me to perform my best. Thank you for knowing exactly when I needed a meal from Chipotle to boost my mood.

Lastly, I would like to thank my mother, Lorena, who instilled the importance of education and hard work in me. Although I am a first-generation college graduate, and the first in my family to get a graduate degree, she made it possible for me to get an education. She has always truly believed in me, and for that, I am grateful.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1. Maritime Cybersecurity

As the maritime industry becomes more reliant on computer-based information systems, the possibility and impact of cyber-attacks become more likely and more harmful. Maritime shipping transports 53 percent of U.S. imports and nearly 40 percent of U.S. exports; successful cyber-attacks on transport vessels could result in catastrophic damage for U.S. businesses and residents [1]. Furthermore, U.S. Navy vessels are equipped with many of the same technology systems found aboard trade and transportation vessels. As maritime cybersecurity is often the front-line of defense in global trade operations, our area of research has a critical niche in supporting national security and the U.S. economy.

This critical nature of maritime trade makes it necessary to have safety and security measures. The International Maritime Organization (IMO) was created by a convention adopted at the United Nations (UN) Maritime Conference in 1948 to regulate the necessary safety rules and measures in maritime trade [2]. All passenger ships and vessels with a gross tonnage above 300 tons are required by the IMO to be equipped with and use the Automatic Identification System (AIS) [3]. The U.S. implemented regulations that further increase the use of AIS through Title 33 of The Code of Federal Regulations, which limits the number of exceptions for AIS use [4]. These regulations result in most U.S. vessels being equipped with the system.

AIS is an automated vessel tracking system that shares information with vessels in the vicinity [5]. Sharing information such as location, speed, and course supports safety at sea as the crew can more easily avoid collisions. AIS also provides more timely tracking of vessels than an automatic radar plotting display and has the advantage of providing surplus information about each vessel. AIS is a transponder system that operates in the very-high frequency (VHF) mobile maritime band [5]. The protocols it operates on were designed for efficiency, not security. As a result, AIS has become an attack vector for voyage networks.

Voyage networks integrate the various electronic and computer systems found

on-board vessels. Developing cybersecurity measures for maritime devices such as AIS will decrease the attack surface of these networks. The Naval Postgraduate School is well-positioned to explore this area of research as it provides students an environment to apply engineering principles to real-world maritime problems. The Engineering Enclave for Maritime Security (EEMS) laboratory is especially useful because it allows students to immediately test and apply their work on real-world marine electronic systems.

1.2. Engineering Enclave for Maritime Security

The EEMS laboratory is maintained by the Electronic and Computer Engineering Department at the Naval Postgraduate School (NPS). The laboratory provides insight into the cyber and control systems that make global trade through shipping possible. At the EEMS laboratory, researchers are working towards simulating a maritime navigation system by integrating commercial maritime equipment. Figure 1 illustrates the current setup of the voyage network in the laboratory. Through research and scientific experiments conducted in this laboratory, a better understanding of inter-network connections in maritime vessels is attained. The knowledge gained from the research conducted in this lab can be used for hardening the cyber-physical systems found in voyage networks. The goals of the work in this thesis are supported by the resources provided in the EEMS lab. Furthermore, our goals directly align with the EEMS objective. The goals of the EEMS lab include:

- Building a laboratory that accurately depicts the integration of equipment on a Type Approved merchant vessel.
- Exploring the network protocols and devices that enable functionality in maritime environments.
- Examining the sensor and communication network consisting of navigation equipment including Global Positioning System (GPS), AIS, and radar.
- Providing an infrastructure that can be used by researchers to study ship communication systems.

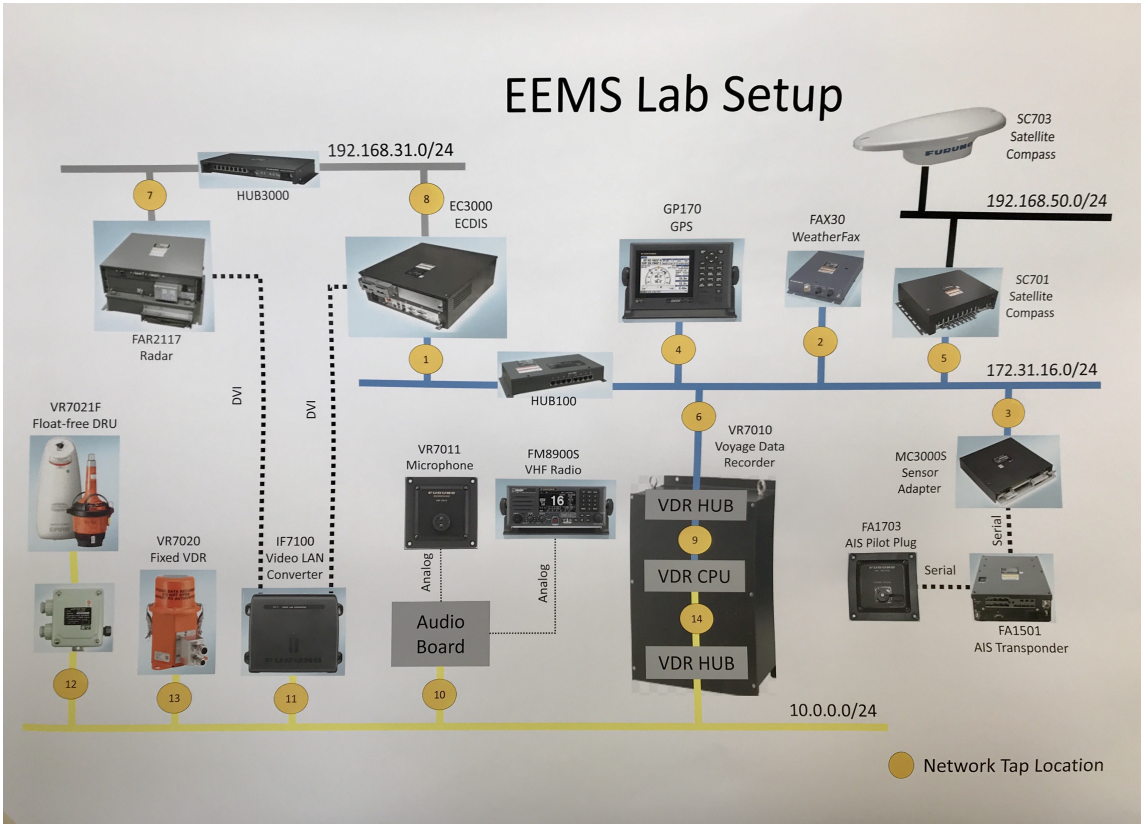


Figure 1. Illustration of the EEMS lab network setup. Source: [6].

1.3. Thesis Goals and Overview

The goals of the work in this thesis are:

1. Assess AIS input sanitization methods for prevention of contact spoofing attacks and arbitrary data injection attacks
2. Design a low-cost AIS security system that is stand-alone, uses minimal resources, and can scale to high traffic areas

We began our work by exploring the literature for tested techniques that could be used to detect falsified AIS messages. From the literature, we expanded and built upon the research that seemed most likely to yield a working detection scheme. We began by examining the protocols and considering which components in AIS messages could be used

effectively for the security design. We then examined algorithms that would use the inner information of a message to identify dubious messages. The result is an algorithm to detect some falsified positional reports.

Second, we explored techniques for mitigating data injection attacks for a variety of information systems. It was necessary to examine what methods could be applied to our specific system. Similarly to our message spoofing work, we began by examining how the protocol and information in AIS messages could be leveraged. Goal one would be satisfied if spoofing detection and injection prevention techniques could be adapted for AIS.

The finalized input data sanitization system is constructed by integrating the security techniques discussed above into a single Python program. Our intention was to create a filter-like software that will take batches of AIS messages being sent over User Datagram Protocol (UDP) as the input, and then will output only the messages that are safe. The resulting security system can scale to high AIS traffic areas and can be implemented without the need for extensive resources, satisfying goal two.

CHAPTER 2:
Background

In this thesis, the AIS protocols are leveraged to build an AIS security system. An understanding of these protocols is necessary to grasp the technical details of this research. This chapter covers the technical basis used to develop the design. Furthermore, the voyage network components, their interconnection, and the message transport process of the network are described. Next, prior related research in AIS cybersecurity will be explored. At the end of the chapter, we state the use case of this research.

2.1. Automatic Identification System

Regulations set by the Safety Of Life At Sea (SOLAS) international convention states that all ships of 300 gross tonnage upwards need to employ an AIS and are required to provide the vessel safety-related information automatically [3], [7]. The use of AIS is globally widespread and millions of AIS messages are sent every day.

The AIS is a shipboard transponder that uses VHF radio signals to broadcast vessel position, maritime mobile service identifier (MMSI), and other useful voyage information. The transmissions are received by other vessels, land stations, or satellites employing an AIS receiver and who are within line-of-sight range. The information being sent over radio-frequency (RF) signals can be received by anybody with the correct tools as it is not encrypted and no license is necessary to transmit or receive data. Over half a million vessels use AIS today making it, "...the backbone of a global ship tracking network [8]."

The intended purpose of this navigation aid tool is to prevent accidents and collisions at sea by increasing maritime awareness. AIS is capable of increasing situational awareness beyond other means, such as radar, as it is able to detect vessels that were previously undetectable thanks to a longer wavelength and one way transmission of RF signals. On occasion, you will have the vessel on radar first before it shows up in AIS. Where radar is merely a return, AIS enhances the radar picture with an overlay of information on the radar contact. The system is capable of sharing information with other components found on a voyage network such as the Electronic Chart Display and Information System (ECDIS),

therefore allowing the crew to more easily digest the information provided.

The operational use of AIS is described by the International Telecommunication Union (ITU)-R M.1371-5 standard. Four frequencies have been designated for AIS use worldwide. These four frequencies are: 161.975MHz, 162.025MHz, 156.775MHz, and 156.825MHz. The first two, named AIS 1 and AIS 2, are the default for AIS operation, while the last two are reserved for message 27 transmission only. Furthermore, AIS uses self-organized time division multiple access (SOTDMA) to allow for more efficient use of the spectrum by all users [9].

2.2. AIS Protocols

AIS operates on two protocols. First, the ITU-R M.1371-5 standard specifies the data format on the external side of the ship when it is being transmitted as RF signals [9]. Second, when the data is on the network side inside the ship, the National Marine Electronics Association (NMEA)-0183 interface standard permits data communication between electronic marine instruments, navigation equipment, and communications equipment [10].

2.2.1. ITU-R M.1371-5

In general, this protocol allows the AIS to efficiently perform its role of automatically broadcasting ship dynamic and static information to all other AIS in the area. The ITU-R M.1371-5 specifies the technical characteristics for an automatic identification system using time division multiple access in the VHF maritime mobile frequency band. This standard outlines the first four layers (physical layer, link layer, network layer, and transport layer) of the open system interconnection (OSI) model of an AIS station [9]. Three components of this standard are the most relevant to this research: SOTDMA, message types, and message formats.

2.2.1.1. SOTDMA

The implementation of SOTDMA is relevant to this research as it controls how many messages a network of synchronized AIS stations can send over unit time. This transmission timing scheme uses the concept of frames and slots; each frame equals one minute, and it is divided into 2250 slots. AIS stations share these slots and use them as

access tokens to be able to transmit in an organized manner. This implies that 2250 messages can be sent in each channel every minute, 4500 messages for both AIS 1 and AIS 2 [9]. 4500 messages per minute is equivalent to 75 messages per second, meaning the security system described in this research should be able to process 75 messages per second in order for it to scale well.

2.2.1.2. Message Types

There are 27 total message types defined in the standard. Each message type is unique, has its own purpose, and provides different information (except for 1, 2, and 3 which are equally structured). The 27 message types are listed and described in Figure 2. In this research, the message types of interest are 1, 2, 3, and 18 (position reports).

Message ID	Name	Description
1	Position report	Scheduled position report
2	Position report	Assigned schedule position report
3	Position Report	Special position report, response to interrogation
4	Base station report	Position, UTC, date and current slot number of base station
5	Static and voyage related data	Scheduled static and voyage related vessel data report
6	Binary addressed message	Binary data for addressed communication
7	Binary acknowledgement	Acknowledgement of received addressed binary data
8	Binary broadcast message	Binary data for broadcast communication
9	Standard SAR aircraft position report	Position report for airborne stations involved in SAR operations
10	UTC\date inquiry	Request UTC and date
11	UTC\date response	Current UTC and date
12	Addressed safety related message	Safety related data for addressed communication
13	Safety related acknowledgement	Acknowledgement of received addressed safety related messages
14	Safety related broadcast message	Safety related data for broadcast communication
15	Interrogation	Request for specific message type
16	Assignment mode command	Assignment of specific report behavior by competent authority using Base station
17	DGNSS broadcast binary message	DGNSS broadcast binary message
18	Standard Class B equipment position report	Standard position report for Class B shipborne mobile equipment
19	Extended Class B equipment position report	No longer required. Extended position report for class B shipborne equipment
20	Data link management message	Reserve slots for Base stations
21	Aids-to-navigation report	Position and status report for aids-to-navigation
22	Channel Management	Management of channels and transceiver modes by Base station
23	Group assignment command	Assignment of specific report behavior by competent authority using Base station
24	Static data report	Additional data assigned to an MMSI
25	Single slot binary message	Short unscheduled binary data transmission
26	Multiple slot binary message	Scheduled binary data transmission
27	Position Report for long-range applications	Class A and Class B 'SO' shipborne mobile equipment outside base station coverage

Figure 2. 27 Message Types. Adapted from: [9].

2.2.1.3. Message Formats

Data is transferred using a transmission packet. Each default packet is 256 bits long, equivalent to one slot. The packet is formatted into different sections; these sections are seen in Figure 3 [9]. The 168 data bits are the area of concern in this research.

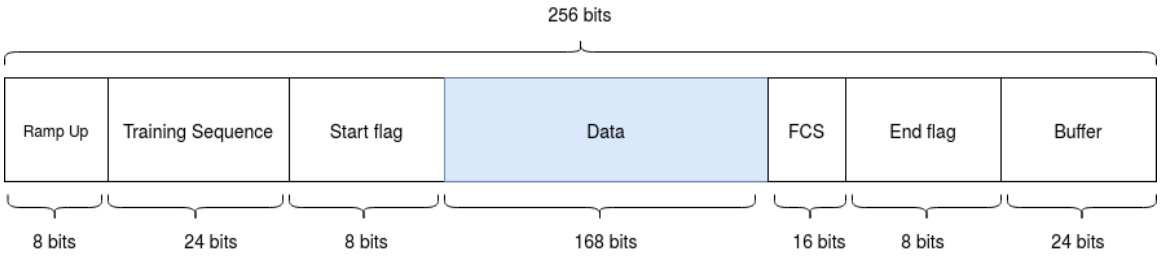
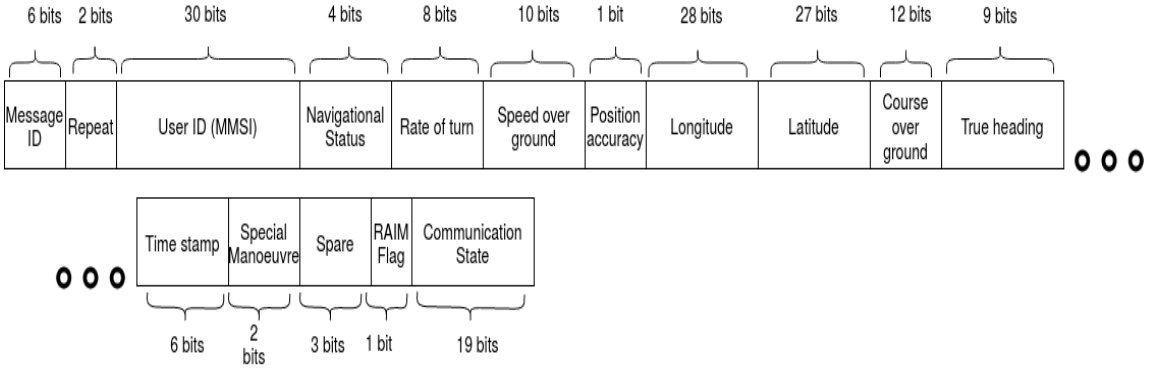
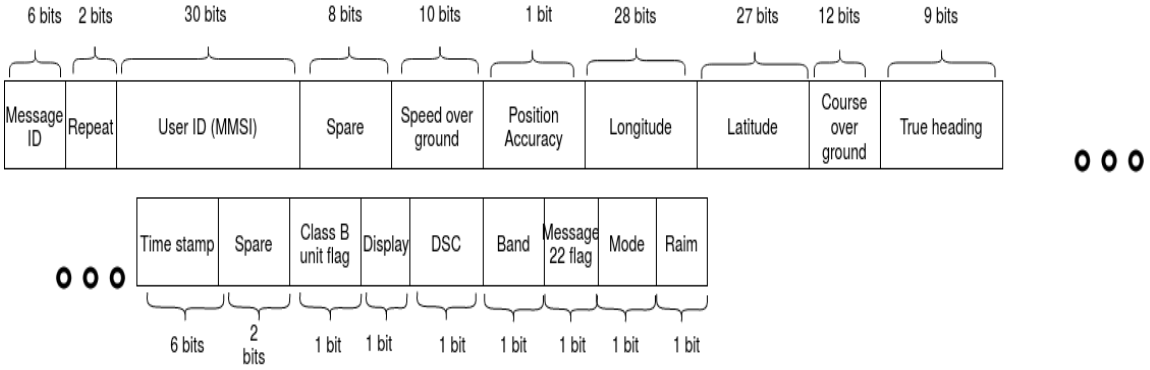


Figure 3. Packet format

The 168 data bits of the message are structured into smaller sections dependent on the message type. Every message type contains multiple fields, and each field is represented by a section of bits (the words ‘field’ and ‘parameter’ are used interchangeably). The binary structure of messages 1, 2, 3 and 18 are illustrated in Figure 4. The most notable difference between position reports type 1, 2, 3 and position reports type 18 is the navigation status and rate of turn parameters; type 18 messages do not contain these fields. Descriptions of these parameters are contained in Figure 5 [9].



(a) Message Type 1,2,3 Structure



(b) Message Type 18 Structure

Figure 4. Message Structure for Types 1,2,3, and 18

Parameter	Description
Message ID	Message type identifier
User ID	MMSI
Navigation Status	0=Under way using engine, 1=at anchor, 2=not under command, 3=restricted maneuverability, 4=constrained by draught, 5=moored, 6=aground, 7=engaged in fishing, 8=under way sailing, 9=reserved for future, 10 = reserved for future, 11=power driven vessel towing astern, 12=power driven vessel pushing ahead or towing alongside, 13=reserved for future, 14=AIS-SART, 15=undefined
ROT	0 to 126+ = turning right at up to 708 deg/minute. 0 to -126 = turning left at up to 708 deg/minute
SOG	Speed in 1/10 knot (0-102.2) 1023 not available, 1022=102.2 knots or higher
Position Accuracy	1=high (<10m), 0=low (>10m)
Longitude	In 1/10000 min. East=positive, West=negative (2's complement)
Latitude	In 1/10000 min, North=positive, South=negative (2's complement)
COG	In 1/10=(0-3599)
Heading	Degrees (0-359) (511=default)
Time Stamp	UTC seconds. 60=not available, 61=positioning system is in manual mode, 62=electronic position fixing system operates in estimated mode, 63=system inoperative
Special Man.	0=not available, 1=not engaged in special man., 2=engaged in special man.
Spare	not used
RAIM	Receiver autonomous integrity monitoring
Comm. State	*1=SOTDMA, 2=SOTDMA, 2=ITDMA*

Figure 5. Parameter Description

2.2.2. NMEA-0183

The intended purpose of NMEA interface standards is to"... facilitate interconnection and interchangeability of equipment, minimizing misunderstanding and confusion between manufacturers, and assisting purchasers in selecting compatible equipment" [10]. NMEA-0183 defines the data transmission protocol and sentence format for a 4800-baud serial data bus [10]. This standard is used by most electronic marine equipment to permit sharing information within the network, including the equipment used in the EEMS lab.

2.2.2.1. Definitions

We use the following terminology throughout the remainder of this work:

- Talker: any device sending data to other devices.
- Listener: any device receiving data from other devices [10].

2.2.2.2. Data Format Protocol

All transmitted data is interpreted as ASCII characters. There are three types of ASCII characters allowed in the protocol: (1) reserved characters are used for specific formatting purposes such as field delimiting, they are not used in the data fields, (2) valid characters consists of all ASCII characters allowed in the data fields, (3) undefined characters are all other characters not defined as valid or reserved, these are not to be transmitted [10]. Tables containing all valid and reserved characters are demonstrated in Figure 6.

Valid Characters

dec	symbol	dec	symbol	dec	symbol
32		64	@	96	`
34	"	65	A	97	a
35	#	66	B	98	b
37	%	67	C	99	c
38	&	68	D	100	d
39	'	69	E	101	e
40	(	70	F	102	f
41	)	71	G	103	g
43	+	72	H	104	h
45	-	73	I	105	i
46	.	74	J	106	j
47	/	75	K	107	k
48	0	76	L	108	l
49	1	77	M	109	m
50	2	78	N	110	n
51	3	79	O	111	o
52	4	80	P	112	p
53	5	81	Q	113	q
54	6	82	R	114	r
55	7	83	S	115	s
56	8	84	T	116	t
57	9	85	U	117	u
58	:	86	V	118	v
59	;	87	W	119	w
60	<	88	X	120	x
61	=	89	Y	121	y
62	>	90	Z	122	z
63	?	91	[	123	{
		93	]	124	|
		95	_	125	}

(a) Valid NMEA-0183 Characters

Reserved Characters

Character	Function
<CR>	End of sentence delimiter (carriage return)
<LF>	Line feed
$	Start of parametric sentence delimited
*	Checksum field delimited
,	Field delimiter
!	Start of encapsulation sentence delimited
\	Reserved for future use
^	Code delimiter for HEX representation of ISO-8859-1
~	Reserved for future use
<del>	Reserved for future use

(b) Reserved NMEA-0183 Characters

Figure 6. Valid and Reserved NMEA-0183 Characters. Adapted from: [10].

There are three overarching fields that compose a NMEA-0183 sentence. First, the

address field begins after the “$” or “!” character and defines the sentence. The “$” delimiter is used to start parametric sentences and the "!" character starts encapsulated sentences.

Second, the data fields follow the address field and a “,” delimiter. This section of the message is composed of multiple fields separated by the “,” character. The longest section of this field is the payload containing the AIS parameters (in the case of an AIS message).

Lastly, the checksum field is included in all sentences, and it follows the ‘*’ delimiter. The checksum value is the XOR (exclusive OR) of all characters in the sentence between the "$" or "!" and "*" characters [10]. An example of a NMEA sentence is provided in Figure 7.

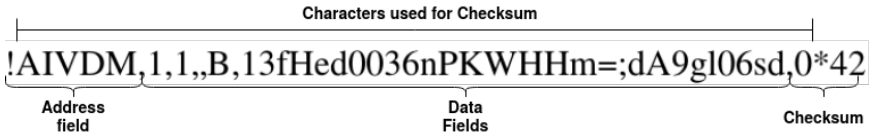


Figure 7. NMEA-0183 Sentence Structure

NMEA-0183 sentences use 6-bit ASCII characters to represent typical 8-bit ASCII text. The 6-bit ASCII conversion table is shown in Figure 8. NMEA-0183 sentences can only contain characters from the 6-bit ASCII alphabet; this allows NMEA-0183 to support any protocol that uses binary or text formats [10]. To find the binary representation of a character using the table, first trace the row of the character to a 2-bit value on the left column, these are the two left most bits. Next, trace the column up to a 4-bit value, these are the last four bits. For example, the character "F" is traced to 1 (01b) and 6 (0110b); putting these together, "F" is represented by 6-bits (010110b).

		Low 4-bits															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
High 2-bits	0	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	1	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	2	P	Q	R	S	T	U	V	W	`	a	b	c	d	e	f	g
	3	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w

Figure 8. ASCII 6-bit Conversion Table. Source: [11].

2.2.2.3. Parametric Sentences

The NMEA-0183 standard supports two sentence classes, parametric and encapsulated. The first class, parametric sentences, start with the "$" delimiter and represent the majority of sentences defined in the standard. The fields in a parametric sentence include the address field, the data field, and the checksum value [10]. This sentence structure is illustrated in Figure 9. Parametric sentences are not typically used to transmit position reports and thus are not highly relevant to this research.

$ADDRE,DATA*CK<CR><LF>

Figure 9. NMEA-0183 Parametric Sentence Structure. Source: [11].

2.2.2.4. Encapsulated Sentences

The second class are encapsulated sentences. These start with the "!" delimiter. Encapsulated sentences transport data from other protocols and have a greater information bandwidth by allowing payloads to be divided and sent over multiple sentences. AIS uses encapsulated sentences since certain message types contain large volumes of data. The structure of encapsulated sentences is similar to parametric sentences, except encapsulated sentences have a few extra fields.

This structure can be seen in Figure 10. Notice the extra fields T, N, S, and F. T is total number of sentences to be transmitted for a single message. N is the current sentence in the sequence. S is the sequential message ID. Fill bits of value zero must be added to

the ‘F’ field when the number of bits in a sentence is not divisible by six in order to allow parsing for 6-bit ASCII. The value in F is the number of fill bits [10].

!ADDRE,T,N,S,DATA*CK<CR><LF>

Field	Description	Size (Bytes)
"!"	Start of Sentence Delimiter	1
","	Field Delimiter	1
ADDRE	5-Character Address Field	5
T	Total Number of Sentences	0+
N	Current Sentence's Number	0+
S	Sequential Message ID	0+
DATA	Data Sentence Block	Variable
F	Fill Bits	1
"*"	Checksum Delimiter	1
CK	2-Character Checksum Value	2
<CR><LF>	End of Sentence	2

Figure 10. NMEA-0183 Encapsulated Sentence Structure. Source: [11].

2.2.2.5. AIVDM/AIVDO Sentences

When AIS messages are received from an external source, these messages are distributed within the voyage network via NMEA-0183 AIS VHF Data-Link Message (VDM) sentences. In the case when the data is coming from within the network and needs to be transmitted out, AIS VHF Data-link Own-vessel (VDO) report sentences are used. These messages are defined by using the characters “!AIVDM” and “!AIVDO” in the address field given in Figures 9 and 10. The structure of AIVDM sentences is illustrated in Figure 11; the structure of AIVDO sentences is the same [10] with the exception of the address field characters.

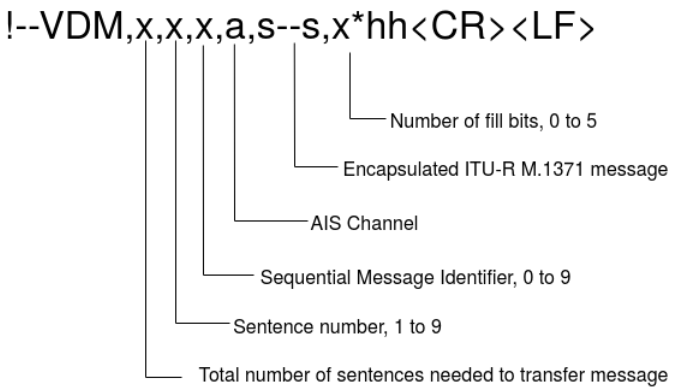


Figure 11. AIVDM Sentence Structure. Adapted from [10].

2.3. Network Interconnection

2.3.1. Light-Weight Ethernet Protocol

The International Electrotechnical Commission (IEC) developed the Lightweight Ethernet (LWE) Protocol, as "... a new computer network standard for use of Ethernet in maritime navigation networks" [12]. The official name of the standard is IEC 61162-450 'Multiple Talkers and Multiple Listeners - Ethernet Interconnection' [12]. This protocol is based on Ethernet and Internet Protocol (IP) and was created as a response to an ever-increasing need for larger bandwidth and high speed communication between industrial devices. LWE is popular in maritime networks as it is easy to implement, creates a more efficient network, and supports NMEA-0183 communication. Also, "[LWE] mandates the use of: single switched Ethernet, UDP datagrams, IP multicast, and a function block approach for devices" [12]. These mandates allow for more efficient routing and help achieve the highest possible bandwidth. The voyage network used for our experiments uses LWE as it allows for multiple devices to be connected without the need to have multiple serial connections for each device.

2.3.2. AIS Message Process

Sending and receiving AIS messages is achieved primarily (not entirely) by the functions of two devices. For example, to transmit AIS data, the Furuno MC-3000S Sensor Adapter collects first all sensor data as NMEA-0183 characters. Next, the sensor adapter feeds the Furuno F-150 AIS Transponder Unit the data formatted as AIVDO sentences via serial connection. The data is then formatted to AIS ITU-R M.1371-5 messages by the transponder. Finally, the AIS transponder modulates the binary data and transmits it over VHF RF signals.

The process of receiving AIS messages is reversed. The transponder unit demodulates and parses the received VHF signal. The binary data is then sent to the sensor adapter as NMEA-0183 AIVDM sentences via serial connection. The sensor adapter converts the binary data to UDP/IP packets and transmits the packets over LWE to the Furuno HUB-100 Ethernet switch to be broadcast and received by other listeners on the network. Figure 12 illustrates the EEMS network setup and communication process between the FA-150 Transponder Unit and the ECDIS. Although the only end device illustrated is the ECDIS, the HUB broadcasts the AIS messages to all end devices.

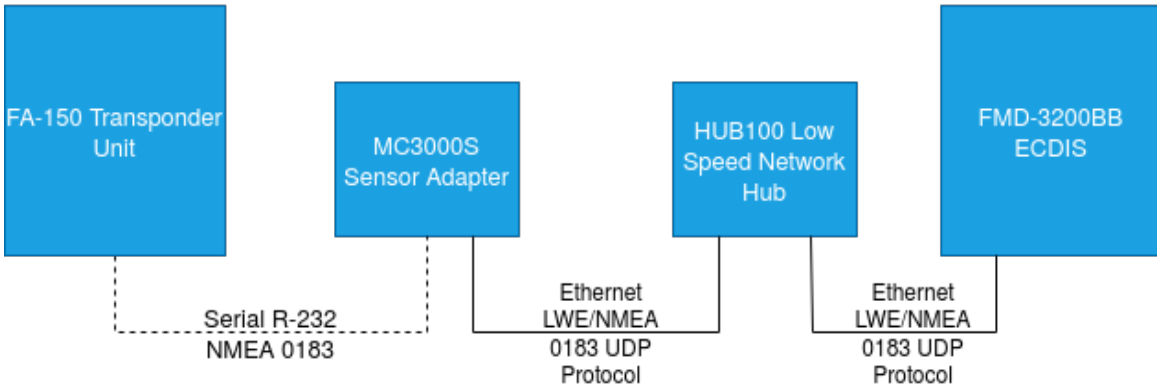


Figure 12. EEMS AIS Network Setup. Adapted from: [11].

2.4. Prior Related Research

2.4.1. AIS Signal Research Tool

NPS alumnus Mitchell Rubinstein intended to exploit the RF aperture to a ship NMEA-0183 network; this led to his research question, "Can we create an RF research tool?" [13] In his research, Rubinstein and a contractor created the AIS Signal Research Tool (A-SRT), a software defined radio (SDR) application able to replicate and customize AIS RF signals. The A-SRT was developed using a Universal Software Radio Peripheral (USRP) X310, with a WBX-120 RF daughterboard as the SDR and GNU Radio as its foundation [13].

The tool takes AIVDM sentences from a text file and formats each sentence into a VHF signal following the ITU-R M.1371-5 protocol. The tool was tested on a Furuno FA-30 AIS receiver connected to OpenCPN, a free chart plotter navigation software. During the tests, it was discovered that OpenCPN and the receiver accepted AIS messages as long as they followed the correct protocol format. This discovery proved that AIS messages could be customized to contain erroneous data and will still be accepted by the network; this is the genesis of a series of NPS theses about the vulnerabilities of AIS security [13].

2.4.2. AIS Based Cyber-Attacks

2.4.2.1. Contact Spoofing

Contact spoofing is a type of attack on AIS networks that relies on falsifying AIS position reports in order to induce some level of chaos, danger, or confusion affecting the operations of the ship and its crew. False AIS position reports generate 'ghost ships', ships displayed by the AIS or ECDIS that are not actually there. The first spoofing attack attempted by Rubinstein proved successful. The "Pac-Man" script, a series of NMEA-0183 AIVDM sentences creating a Pac-Man scenario in which two vessels chase each other, was transmitted to the network and successfully simulated the scenario in OpenCPN.

Joshua Heaney, another NPS alumnus, followed Rubinstein's work and conducted a spoofing attack of his own. Heaney's attack still used the A-SRT to transmit the "Pac-Man" script [11]. Alternatively, this attack was directed at the Type Approved devices in the EEMS

network which includes the FA-150 Transponder and ECDIS unit. The attack was noted successful when both the ECDIS and FA-150 displayed the ghost ships [11]. This proved that industrial devices such as those found on ships are vulnerable to contact spoofing.

2.4.2.2. Arbitrary Data Injection

The second type of attack performed in the EEMS lab was a data injection attack conducted by Steven Marion [14]. Marion's goal was to inject data via the RF aperture and have the data be recorded by an implant he had placed in the network, while avoiding detection by other devices. The implant was a laptop operating on Ubuntu connected to an open port of the HUB-100 running a Python program that captured the injected messages. Injecting arbitrary data became a simple task as he realized he could substitute the payload of real NMEA-0183 sentences with segments of object code as long as the characters in the sentence were all valid. Furthermore, avoiding detection was accomplished by using an invalid checksum making all devices on the network discard the message.

Marion conducted a specific data injection attack where he was able to transmit code segments, compile all the segments on the implant, and execute the script, triggering it with one more RF transmission [14]. This attack begins with the A-SRT transmitting NMEA-0183 encapsulated sentences containing segments of the *HelloWorld.py* script. The implant then runs a program called *Listen.py* which looks for key words in the encapsulated sentence and appends the code segments until the object code is complete. Finally, one more message containing the trigger word 'KILL' is transmitted and causes the implant to execute the HelloWorld script [14]. The sequence of this attack is illustrated in Figure 13.

A more vicious attack was formulated by Marion but never executed in his research [14]. The novel attack follows the same sequence in Figure 13 but the uploaded code is actually malicious. He proposes that it is possible to upload an executable that scans for ports, records file structures, and captures information in the network. This data can then be extracted by another function, 'Transmit Out', that causes the implant to send the information to the AIS to be transmitted. In his research he proved that AIS transmissions could be executed by a 'Transmit Out' function in the implant [14]. Although the attack was never conducted, the technical implementation is not much harder than his previous attack.

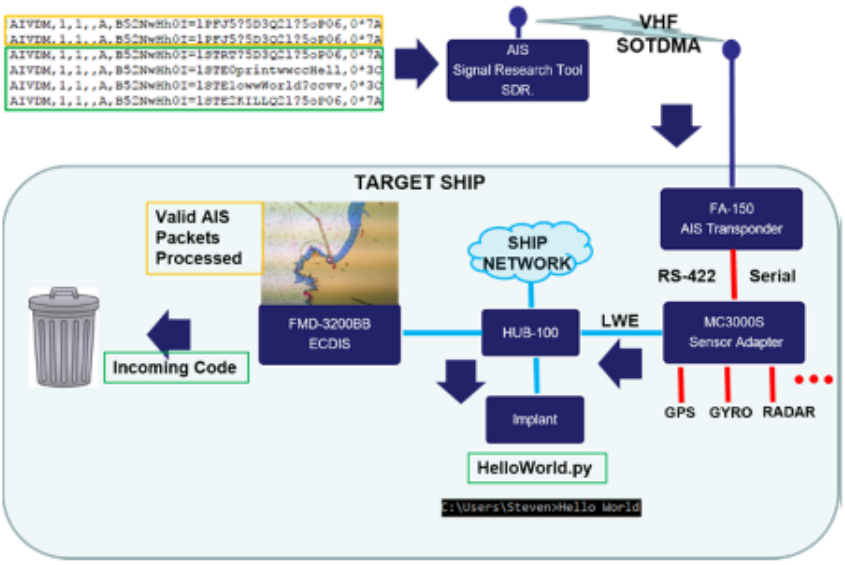


Figure 13. Marion's Code Upload and Execution Process. Source: [14].

2.4.3. Alternative AIS Security Approach

An alternative AIS cyber-security architecture was designed and tested by another NPS student subsequent to Marion's work. Duc H. Nguyen [15] proposed to solve the AIS contact spoofing vulnerability problem using cryptographic techniques. His research further demonstrates the importance and difficulty of securing AIS networks. His research is explored in more detail in this section.

2.4.3.1. "Hardening Automatic Identification Systems: Providing Integrity Through an Application of Lightweight Cryptographic Techniques"

Nguyen's thesis proposes an "...AIS-specific application of a lightweight cryptographic algorithm to potentially decrease the vulnerable attack surface and thereby improve safety of navigation at sea" [15]. Ultimately, he is trying to provide proof-of-concept that integrity and authentication capabilities can be added to AIS compatible with NMEA-0183 and AIS protocols. The architecture through which these capabilities are added is a three-mechanism design demonstrated in Figure 14. This design requires the receiving AIS station, ship B, to respond with a challenge message asking the original sender, ship A, to

verify the normal AIS message transmission previously sent. Ship A then sends a response message to ship B containing cryptographic authentication material. The research selects an asymmetric cryptographic signature algorithm meant to reduce bandwidth needed by the cryptographic architecture, therefore increasing ability to scale to high traffic areas. Interestingly, the architecture leverages AIS message types that carry arbitrary binary data in order to send the necessary cryptographic material such as keys. Technical problems in his research includes message capacity not being enough for ideal cryptographic algorithms, success rates of less than 50%, and other more complicated dilemmas. Ultimately, he was able to achieve proof-of-concept using this architecture to authenticate messages [15].

The weaknesses of Nguyen's design, compared to the design in this research, mostly deals with scaling capacity. The end-to-end time of the cryptographic process lasted a median time of 5.67 second [15], too long for areas congested with ships. Furthermore, the Central-Processing-Unit (CPU) load of the design was significant in the testing environment with only two AIS transponders [15].

Although Nguyen's design is hard to scale and fairly computationally expensive, there are various strengths in the architecture. First, the authentication security applies to all message types available for AIS whereas the design in this research does not. Furthermore, because cryptographic algorithms are well established, this method is more robust in providing strong security [15]. The two designs demonstrate that there are trade-offs when creating an AIS cyber-security architecture. A truly minimalist approach struggles to maintain a highly secure and robust system but scaling capacity is highly apparent. A more complex system provides clear and greater security but is resource intensive and does not scale as well.

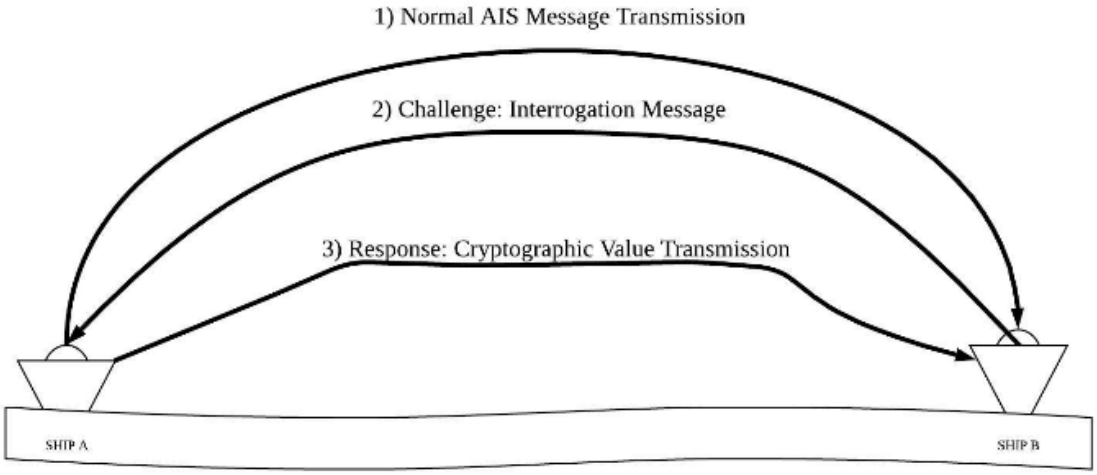


Figure 14. Nguyen's Three-Mechanism Design Architecture. Source: [15].

2.5. Use Case

The U.S. Navy and maritime industry would be the primary beneficiaries of an improvement in AIS security. An important part of our research is to create a minimalist design that can easily be implemented on naval vessels without breaking the bank or completely altering AIS capabilities. Our proposed design does not include security against all possible attacks for all AIS message types, but our proposed approach does attempt to analyze the bulk of AIS messages. Our research results should be considered as providing a higher level of minimum security for vessels when they are admitting AIS data into their shipboard networks.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Design and Methodology

3.1. Research Questions

Naval Postgraduate School researchers demonstrated that network systems with AIS are vulnerable to cyber-attacks. Rubinstein [13] demonstrated the AIS protocol is inherently flawed as there is no method through which it checks the confidentiality or integrity of the received messages. Joshua Heaney [11] and Steven Marion [14] then proved that Type Approved systems are vulnerable to spoofing and injection attacks.

This prior work led to our primary research question: Could a security system be implemented on AIS-networked systems that prevents falsified messages from reaching the other components on the network? In order to create a system that is practical and can be scaled to high-traffic areas, we questioned if a design that does not require modifying AIS protocol or functionality could be built.

With respect to this objective, the system was designed with three subordinate research questions in mind: "How do we detect contact spoofing?", "How do we detect and prevent arbitrary data injections?", and "How do we integrate the system while keeping it low-cost and low-power?" These questions led us to explore the literature on AIS security and develop novel and creative ways to build on previous research in order to design an AIS-specific security system. Using Python to write the security software supported the objective of keeping this system low-cost.

3.2. Threat Model

3.2.1. Threat Agents

The first threat this security system is designed to protect against are contact spoofing attacks. Contact spoofing is the injection of false position reports. This type of attack was proved possible in [13] and was later demonstrated to be successful on Type Approved systems [11]. The goal of these attacks is to threaten the safety of the ship by

setting off closest point of approach (CPA) alarms, encouraging an unnecessary change in speed or heading, or otherwise disrupting the crew of the ship. Our design for contact spoofing detection is specifically constructed to work on AIS message types 1, 2, 3 and 18 because these messages are frequently transmitted, essential for safety, and contain data that we believe can be leveraged for a detection algorithm.

The second threat targeted by our system is arbitrary data injection attacks. Steven Marion [14] conducted an attack on the EEMS voyage network in which he was able to transmit code into the system via AIS and later execute the code on an implant he had previously inserted on the network. Refer to Figure 13 in Chapter 3 for a detailed explanation of this attack. These attacks are distinct from contact spoofing injections by nature of their payload. The aim of these threats is not to create ghost traffic; instead, they carry a malicious sequence of bits or characters. A sub-goal of this type of attack may be to make the injected code look random.

The design of this security system does not extend to other potential cyber-attacks. The AIS protocol allows for transmission of unstructured arbitrary data in message types 6, 8, 12, 14, 17, 25, and 26, making it difficult to prevent data injection attacks through these messages. These types of messages are used to communicate binary data, send acknowledge replies, and send safety related messages. Thankfully, these message types are rarely transmitted making them automatically suspicious. Also, creating an automatic malicious payload detection scheme for these message types would require more code, potentially slowing system response. Because message types 6, 8, 12, 14, 17, 25, and 26 represent a very tiny minority of traffic, a potentially better security measure would be to simply sandbox these messages when received and alert the crew. This feature was not implemented in our system due to time constraints.

3.2.2. Mitigation

The three security layers that were built into the overall design are explained in this section. The first one is the falsified message detection layer. The objective of this layer is to detect when an entity is falsifying AIS positional reports. The algorithm aims to prevent dubious contacts from entering the system and being displayed on the ECDIS.

The second layer is an allow-list whose goal is to deny any message that seems

extraordinary. The allow-list layer checks the field values of message types 1, 2, 3 and 18 and compares them to a register of values that are considered normal. When the fields are given values that are not normal, the message is denied from entering the network.

The last section of the security system adds an extra layer of security. In this process, some of the values in the AIS message fields are modified before being encoded again and forwarded to the rest of the voyage network. The goal of this layer is to corrupt malicious data that successfully passes through the first two layers without affecting the usability of the data by the crew.

3.3. Design

In order to not change the protocols and functionality of AIS, it was decided that the best approach would be to construct an external, "pass-through" security system that works independently from the AIS transponder, Furuno MC-3000S Sensor Adapter, and the rest of the network components. The voyage network in the EEMS lab is structured so that the AIS transponder communicates with the sensor adapter through a serial connection. The sensor adapter then packages the NMEA-0183 encapsulation sentences into UDP packets to be transmitted to the HUB-100 through an Ethernet cable; from here, the message will be distributed to the other components on the network (refer to Figure 12). The security system is designed to be integrated between the sensor adapter and the HUB-100 (the rest of the network). Referencing to Figure 15, the green component is where the security system would be placed in the network. While this shows an implementation among Furuno systems, the security system can be implemented on any network that passes AIS protocol traffic; the system may need to be adapted for the connection protocols other than LWE.

The processes of our security system are differentiated into three layers: the spoofing detection layer, the injection prevention layer, and a data modification layer. These layers are each written and put together in Python where they can be executed in sequence.

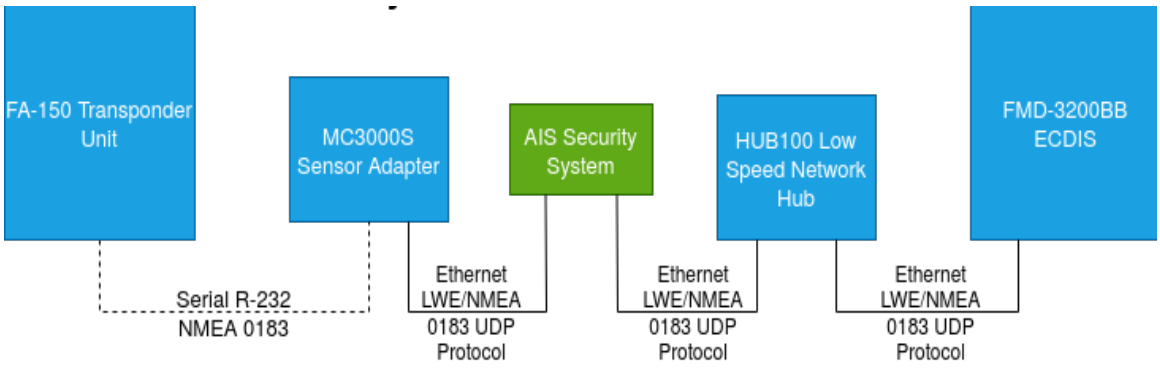


Figure 15. Navigation Network Integrated with Security System

The decision process and technical characteristics of the security system design are explored in the next section. First, the contact spoofing layer's design process is explored. Existing designs and the selected approach are described. Second, the architecture of the arbitrary data injection layer is explored. Finally, layer three is discussed and a system overview is presented at the end of the section.

3.3.1. Layer 1: Contact Spoofing Detection

3.3.1.1. Previous Research Considered in Design

The approach described in a paper from The Institute of Electrical and Electronics Engineers (IEEE) titled, "Countering Real-Time Stream Poisoning: An architecture for detecting vessel spoofing in streams of AIS data" [16], is interesting because of its simplicity, its ability to work in real-time, and its high scaling capacity. They used a straight-forward algorithm to detect when a position report is spoofed. This algorithm is described in Figure 16. The only parameters used in this algorithm are the position (M) and timestamp (M.timestamp). The algorithm calculates the average speed that it would take a ship to travel the shortest distance between two consecutive reports. This algorithm assumes that any ship with a calculated speed of more than 50 knots is spoofed. Furthermore, the architecture seen in Figure 17 is used to create a system that can handle large volumes of data efficiently. This framework is called the Akka framework and makes the distribution of workload possible [16]. In short, the master node balances the workload for the worker nodes. The

worker nodes actually implement the detection algorithm and they report to the consensus node who then measures system performance. This system had high accuracy when tested against an artificially created data set [16]. This approach was not entirely used for our design as we believe the detection scheme is too unsophisticated and could easily be avoided (by reporting speeds under 50 knots).

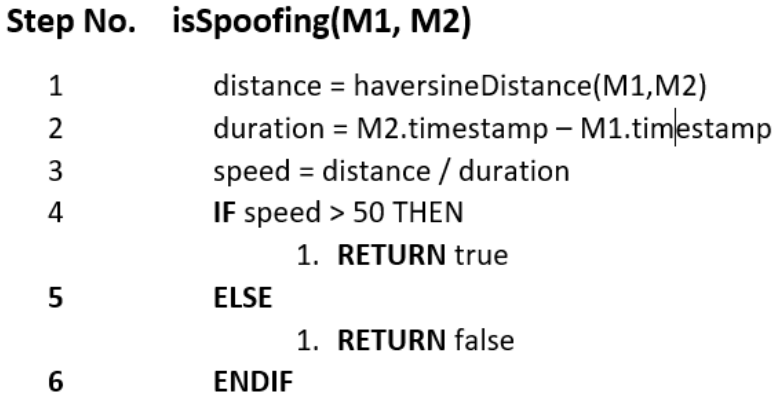


Figure 16. Kontopoulos' Detection Algorithm. Source: [16].

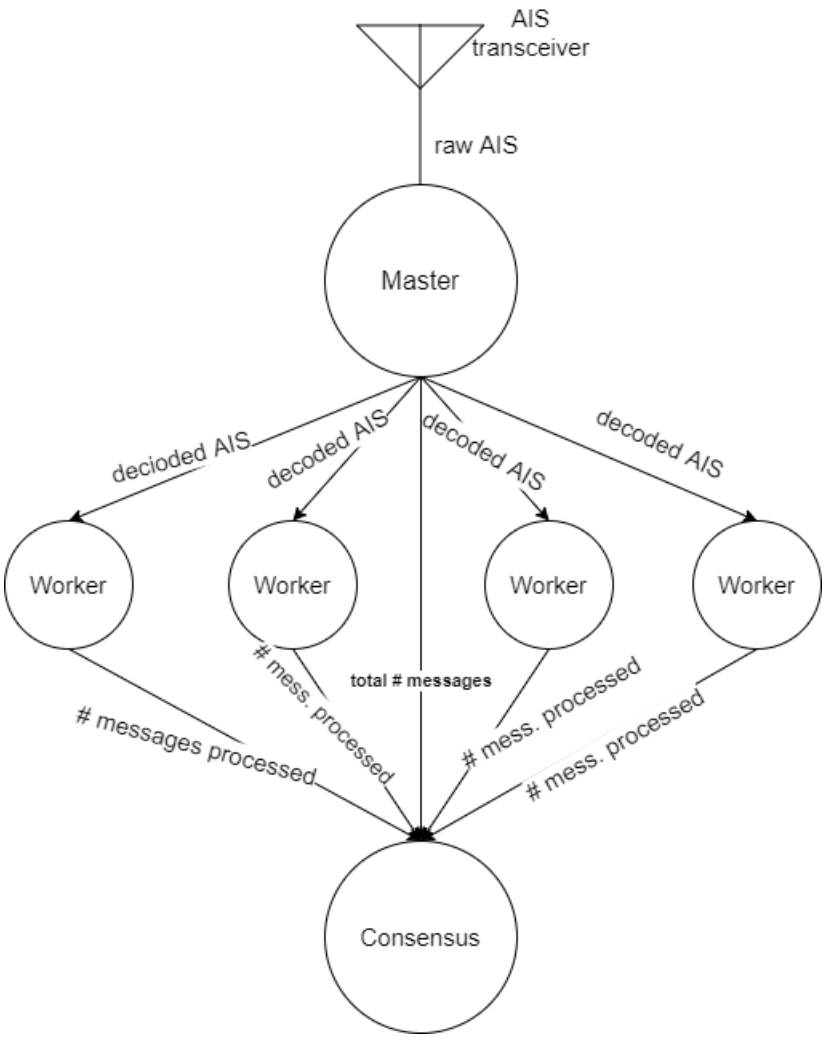


Figure 17. Akka framework. Source: [16].

A second approach we found informative comes from the IEEE paper, “Maritime Anomaly Detection using Gaussian Process Active Learning” [17]. The author of this paper took a bottom-up approach in which he uses the large amounts of AIS data available to build a machine learning (ML) model that calculates the normality of vessel behavior [17]. Normality is calculated according to the velocity of a vessel given her current latitude and longitude. Different vessel types have different behavior; for this reason, the data was pre-processed to be separated by vessel types. Furthermore, the heading of the vessels had

to be restricted to values between zero and 180 degrees to account for the bi-modal nature of shipping lanes. The model was then trained on English Coastline data using an Active Learning paradigm to reduce the computational cost associated with training on large data sets. To test their model, they generated anomalies by artificially moving data points from an original data set in any direction, therefore changing the latitude and longitude of certain samples. Their results showed that the model anomaly detection capability increased when spoofed vessels were located far away from the normal shipping lanes, but it was inaccurate when the vessel was displaced only a short distance from the shipping lane [17]. Although an interesting approach, we don't believe this method can be scaled and easily deployed on-board ships. It would require intensive work to extend the model to a global scale. Furthermore, the need for constant updating of the model would be too costly for our purpose.

Interesting research that is not directly linked to contact spoofing detection but is still insightful for this research can be found in the article, "Experimental Comparison of Ad Hoc Methods for Classification of Maritime Vessels Based on Real-life AIS Data" [18]. This research attempts to classify vessels by their vessel type using the information found in AIS messages. Their first step was to pre-process the data in order to convert it into something that would result in a more robust classifier. During the pre-processing stage, they condensed the list of possible vessel types. Three resulting sets of vessel types were created, each one more condensed than the previous (refer to Figure 18). Next, they trained classifying models on the pre-processed data. The following AIS attributes are used as inputs for the classifier in [18]: longitude, latitude, course over ground, speed over ground, ship length, ship width, ship draught. Using these fields, the classifier could predict the vessel type. Refer to Figure 19 for an illustration of the classification process and classifying parameters.

The research in [18] evaluated the performance of five classifiers: (1) Decision Tree Predictor, (2) Fuzzy Rule Predictor, (3) k Nearest Neighbor Predictor, (4) Neural Network Predictor, and (5) Naive Bayes Predictor [18]. The decision tree, fuzzy rule, and k nearest neighbor models performed exceptionally well. The accuracy increased as the number of resulting classes were condensed to seven. These three predictors constantly had an accuracy close to 90% [18]. We found these results intriguing and tried to incorporate this research to our own design; this is discussed later in this chapter.

16 Resulting Classes	12 Resulting Classes	7 Resulting Classes
A1 Fishing	B1 Fishing	C1 Fishing
A2 Cargo	B2 Cargo	
A3 Tanker	B3 Tanker	C2 Commercial
A4 Passenger	B4 Passenger	
A5 High Velocity	B6 High Velocity	C3 Persons
A6 Sailing	B6 Sailing	
A7 Pleasure	B7 Pleasure	C4 Fun
A8 Military	B8 Military	
A9 Law Enforcement	B9 Law Enforcement	
A10 Pilot	B10 Pilot	C5 Official
A11 Towing		
A12 Underwater Work		
A13 Port Tender	B11 Service	C6 Service
A14 Anti-Pollution		
A15 SAR		
A16 Medical/noncombatant	B12 Humanitarian	C7 Humanitarian

Figure 18. Sets of Resulting Classes. Source: [18].

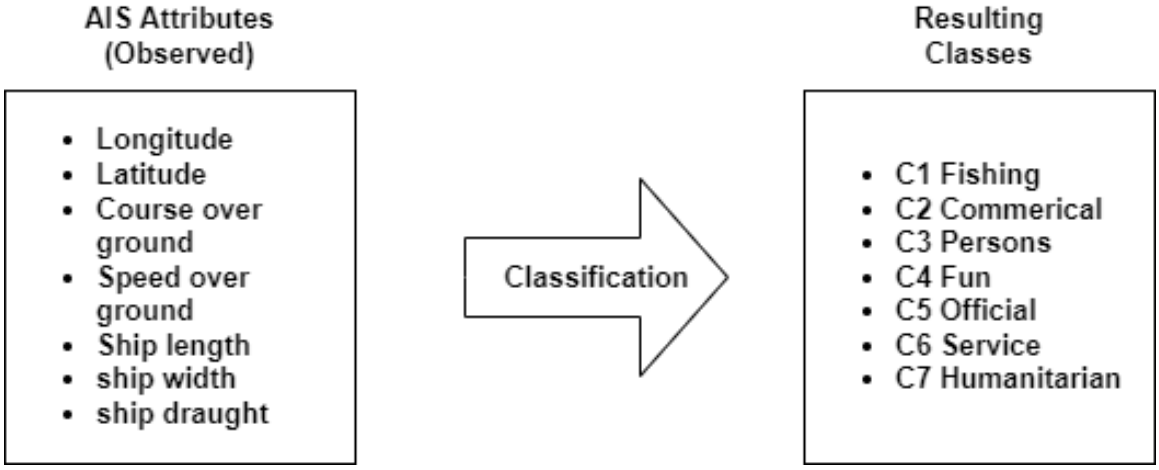


Figure 19. Classification Task With 7 Resulting Classes. Source: [18].

Lastly, the research paper "Detection of false AIS messages for the improvement of maritime situational awareness" [19] looks at a novel approach to detect false AIS traffic. The purpose of this paper is to introduce "... the quality dimensions of data that shall be used in a quality assessment of AIS messages, in order to point out the dubious ones" [19]. The authors believe that by processing the data in AIS messages in a mathematical way, they could rate the integrity and reliability of the data. This paper introduces a message-based approach that uses the inner-information of messages to detect the ones that are falsified. The seven quality dimensions introduced in [19] are:

1. Accuracy: "Considering an attribute *a* of an entity *e*, its standard value is *v*" [19]. Given a value *v'*, inaccuracy is measured by considering the difference between the correct value *v* and *v'*. One difficulty is that the correct value is not always trivial and can even be undefined.
2. Precision: Measurement of degree of detail in the given values (i.e., number of decimal places).
3. Reliability: Measure to which the values obey the rules defined by the outline.
4. Timeliness: Binary true or false on whether values are up to date. Does not account for attributes that are changing by nature (e.g., speed, location).
5. Completeness: Proportion of values that need to be filled in and are filled in.

6. Consistency: The values of different terms must agree within a message or between messages.
7. Integrity: Measurement of consistency leveraged with the use of temporal dimensions.

The paper does not tangibly define mathematical algorithms that could be used in assessing the quality dimensions of AIS messages. Instead, it provides a general application of the quality dimensions leaving it up to the reader to come up with their own methods for assessing quality [19]; these general guidelines are described in Figure 20. The method and guidelines described in [19] are utilized to create the contact spoofing detection architecture seen in this research.

Quality Dimension	Application of the dimension
Accuracy	Definition of standard alues, for instance in the case of the speed of the vessels
Precision	Definition of latitude and longitude, as seen above
Reliability	General coherence of messages with respect to ITU recommendations
Currentness	Use of time stamps of messages
Completeness	Definition of all the fields that should be filled in depending on the message, then computation of their ratio
Consistency	Coherence of information within a message or between messages
Integrity	Enhancement of consistency with temporal recording of actions

Figure 20. Application of Quality Dimensions. Source: [19].

3.3.1.2. Selected Design to Detect Falsified Position Reports: Quality Dimensions Approach

Our approach, for the purpose of spoofing detection, is to assess the quality dimensions of incoming AIS messages in order to point out the dubious ones, as described in [19].

Two quality dimensions are assessed in this research, integrity and accuracy. Measuring the degree of integrity and accuracy are not trivial tasks. No specific methods to assess these quality dimensions were explored in [19]. For this reason, novel algorithms for measuring the quality of AIS messages were created in this research and are described in this section. Next, by applying the quality assessment algorithms to real life AIS data, we calculated the quality of real messages and used the results to set a threshold that separates the real, high-quality messages, from the false, low quality messages; this ability to distinguish between high and low quality reports is ultimately how the detector functions.

Integrity As mentioned previously, integrity is the measure of consistency in the data with respect to a temporal dimension. The geographic location of a vessel is the selected attribute, a, to measure integrity. Location consists of two parameters, latitude and longitude, which change in time with respect to the speed and heading of the vessel (as well as environmental parameters). Non-environmental parameters are provided in positional reports (message types 1, 2, 3 and 18) in the following fields: latitude, longitude, speed-over-ground (SOG), course-over-ground (COG), and timestamp. Using these parameters, taken from two consecutive reports, it is possible to mathematically predict the expected location of a vessel at the time of the second report. In simple terms, the latitude and longitude are updated using the distance that we can expect the vessel to travel given its speed and heading. The equations for predicting the expected position for the second report are noted below:

$$lat_{expected} = lat_{t=1} + \Delta lat \tag{1}$$

$$lon_{expected} = lon_{t=1} + \Delta lon \tag{2}$$

$$\Delta lat = speed[kn] * \Delta t[hr] * cos(course).degrees() \tag{3}$$

$$\Delta lon = speed[kn] * \Delta t[hr] * sin(course).degrees() \tag{4}$$

$$\Delta t = t_2 - t_1 \tag{5}$$

Integrity is then measured by taking the error (the difference) between the predicted position and the reported position at the time of the second report (the latter is provided in the AIS message). This error is the distance between the calculated coordinates and the reported coordinates; it is measured in nautical mile units.

$$error = distance([lat_{t=2}, lon_{t=2}], [lat_{expected}, lon_{expected}]).nm() \tag{6}$$

Figure 21 illustrates the algorithm in action.

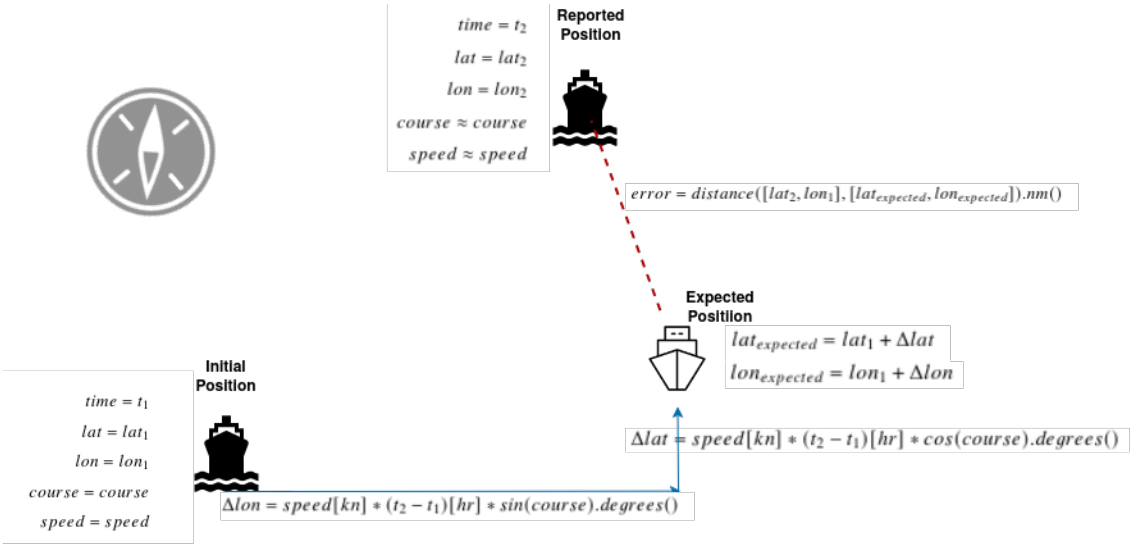


Figure 21. Integrity Algorithm: Calculating Expected Position

Accuracy The second quality dimension assessed in this design is accuracy. Reiterating, given an attribute *a* of entity *e*, accuracy is optimal when the given value *v'* is equal to the correct value *v*. Our initial approach to measure accuracy consisted of two attributes, SOG and vessel type, each assessed separately.

The algorithm to measure accuracy for attribute SOG is described first. In terms of AIS, *e* is the message, *a* is parameter SOG, and *v'* is the value in SOG, aka the velocity of the vessel. The correct value, *v*, is not explicit, therefore it is necessary to find a way to decide what *v* should be. The design uses an idea first seen in [16] in which speed is calculated by taking the shortest distance between two positions from two consecutive reports and dividing that distance by the difference in the corresponding timestamps. The calculated speed is then considered to be the correct value *v*. Inaccuracy is then calculated by taking the absolute difference between *v* and *v'*. Notice the decision to choose which speed is the correct value is a matter of opinion and could be reversed whilst providing the same results.

The algorithm is mathematically described below:

$$speed_{calculated} = \frac{distance([lat_{t=1}, lon_{t=1}], [lat_{t=2}, lon_{t=2}]).nm()}{\Delta t[hr]}[kn] \tag{7}$$

$$\Delta t = t_2 - t_1 \tag{8}$$

$$error = |speed_{t=2} - speed_{calculated}| \tag{9}$$

This algorithm is further illustrated in action in Figure 22.

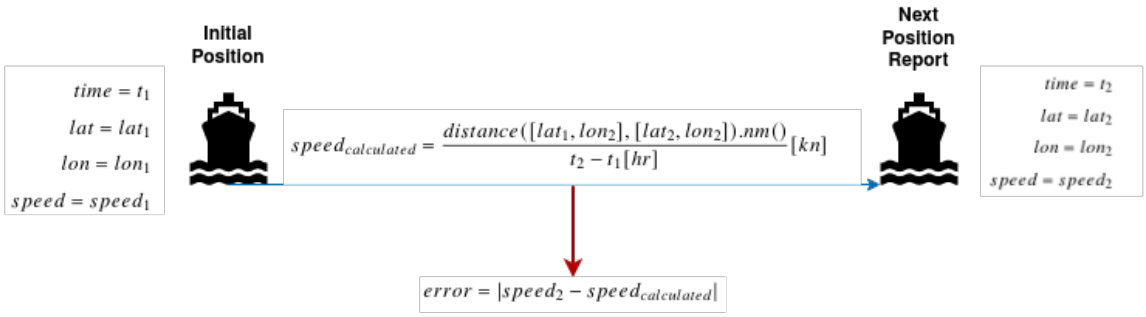


Figure 22. Accuracy Algorithm: Calculating Speed for Accuracy Assessment

The second attribute considered for assessing accuracy, but ultimately left out of the final design, is vessel type. The idea for this part of the design branches from the vessel type classification work that is described in [18]. The main idea is that the correct vessel type value, *v*, can be attained by classifying the vessel using the fields: latitude, longitude, SOG, COG, ship length, ship width, and ship draught. By means of an accurate classification model, accuracy can be assessed in a binary way by comparing the given vessel type *v'* (reported in message) to the predicted type *v*. If the model prediction is equal to *v'*, accuracy is given a passing quality score; if they did not match, the message is considered dubious.

This classification process is adapted from [18]. The process of creating a classifying model started by collecting historical AIS traffic. Second, three registers of resulting classes were created by condensing the initial 17 vessel types into 13 types, and then further into 8 types. Refer to Figure 23 for a detailed illustration of the reduction of resulting classes. The data was then labeled with the vessel types from the reduced classes. Using the MATLAB built in ML libraries, four supervised ML classification models were trained on the data:

1. Fine k Nearest Neighbors
2. Medium k Nearest Neighbors
3. Fine Decision Tree
4. Bagged Decision Tree

These models take the message parameter values as input and output the vessel type. The process of classifying is illustrated in Figure 24.

This ML classification work is then merged into the quality dimensions approach by taking the output of the classification model as the correct value, v. The accuracy is assessed in a binary way by comparing predicted vessel type and reported vessel type. Ultimately, we did not implement this model as message types 1, 2, 3 and 18 do not have the following fields: ship length, ship width, and ship draught. This makes it difficult to assess the quality of single messages via this method. Furthermore, we discovered that models become inaccurate when trained on one specific region and tested in a different region. This problem was discovered when we trained our models either on a western region of the U.S., West bounded at -160°and East bounded at -114°, or an eastern region of the U.S., West bounded at -114°and East bounded at -60°, and then proceeded to test the model using model from the opposite region. The accuracy of k Nearest Neighbor and Decision Tree models when tested on these regions can be seen in Figure 25; the average accuracy is below 70%. We were unable to determine the cause for this drop in accuracy in the time available for this research.

17 Types	12 Types	9 Types
T1 Fishing	T1 Fishing	T1 Fishing
T2 Cargo	T2 Cargo	T2 Commercial
T3 Tanker	T3 Tanker	
T4 Passenger	T4 Passenger	T3 Persons
T5 High-Velocity	T5 High-Velocity	
T6 Sailing	T6 Sailing	T4 Fun
T7 Pleasure	T7 Pleasure	
T8 Military	T8 Military	T5 Official
T9 Law Enforcement	T9 Law Enforcement	
T10 Pilot	T10 Pilot	
T11 Towing	T11 Service	T6 Service
T12 Underwater Work		
T13 Port Tender		
T14 Anti-Pollution	T12 Humanitarian	T7 Humanitarian
T15 SAR		
T16 Medical		
T17 WIG	T13 WIG	T8 WIG
T18 other	T14 other	T9 other

Figure 23. Resulting Classes: Vessel Type. Adapted from: [18]

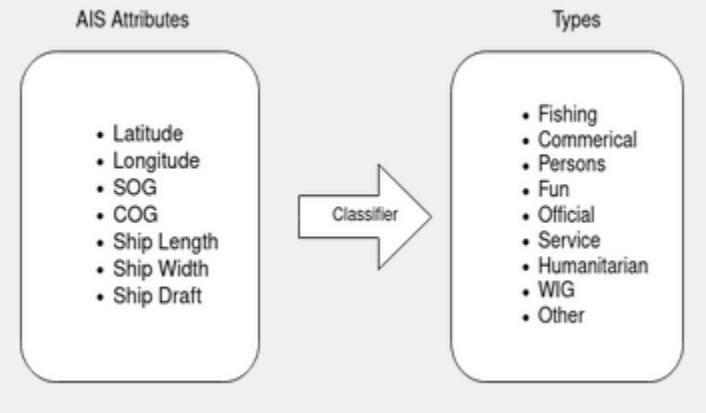


Figure 24. Design: Classification Process. Adapted from: [18]

14 Type Classifer Trained West, Tested East			9 Type Classifer Trained West, Tested East		
Classifier	**Accuracy (%)**		**Classifier**	**Accuracy (%)**	
Fine KNN	64		Fine KNN	68.5	
Med KNN	65		Med KNN	70	
Fine Tree	68		Fine Tree	72	
Med Tree	66.2		Med Tree	71	

Figure 25. Classification Results When Model Predicts on a Different Region.

Using Empirical Data for Setting the Detection Thresholds The purpose of using real-life AIS traffic is to quantify the integrity and accuracy of real AIS streams and use it for setting a threshold, γ, that separates the real and spoofed traffic. The hypothesis is that spoof traffic will result in an error that surpasses γ, while real traffic results in an error less than the threshold. More specifically, the design is built to maintain a false alarm rate of 20% or less. This is done by finding the 80% thresholds in integrity and accuracy error for real life AIS traffic. In other words, 80% of samples from real messages should result in an error that is less than the threshold.

$$
\begin{aligned}
&if\ (error \geq \gamma_{80\%}):\\
&\quad spoof = true\\
&elseif\ (error < \gamma_{80\%}):\\
&\quad spoof = false
\end{aligned}
$$

The integrity assessment algorithm in section 3.3.1.2. was implemented in MATLAB to analyze the quality of streams of messages coming from four different sources:

(1) MarineCadastre.gov [20], (2) Danish Maritime Authority [21], (3) SeaVision [22], and (4) an archive of AIS traffic logs collected from global satellites for dates 01/01/2012, 07/30/2012, and 11/28/2012. Source 4 is a hard drive maintained at the Naval Postgraduate School EEMS lab. Once the algorithm was executed, the probability distribution function (PDF) and cumulative distribution function (CDF) of the calculated errors for each data set were generated. The distributions illustrate that error is concentrated around zero nautical miles for all six data sets. The error distribution for the SeaVision data is illustrated in Figure 26. The error distributions for all six data sets can be found in Appendix A. Next, the 80% thresholds were noted for each data set and the median of these thresholds was selected for the design in order to keep the false alarm rate as close to 20% as possible. Figure 27 contains a table of the recorded 80% threshold value for each data set, as well as the median threshold.

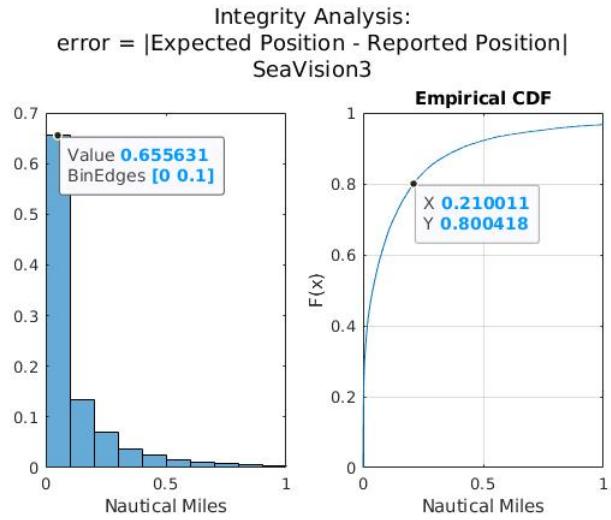


Figure 26. Integrity Analysis: SeaVision Data PDF and CDF for Integrity Error

	A	B	C	D
1			Quality Dimension: Integrity Empirical Metrics	
2	**Data Set**	**80% Threshold**	**Sample Size**	
3	Marine Cadastre (Cleaned)	0.17	244276	
4	Danish Maritime Authority	0.01	1893444	
5	Seavision	0.21	56483	
6	AISHub 20121128 Log	0.29	500000	
7	AISHub 20120101 Log	0.08	500000	
8	AISHub 20120730 Log	0.25	500000	
9	Total		3694203	
10				
11				
12				
13	**Median Threshold**	0.19		
14	**Mean Threshold**	0.1683333333		
15				

Figure 27. 80% Thresholds for Integrity Error for Six Data Sets

An accuracy assessment of real AIS data was conducted similarly to the integrity assessment. The accuracy error algorithm described in section 3.3.1.2. was implemented on the same six data sets to attain the speed error (in knots) that was present in each. Using the calculated speed errors, we generated the CDF for each data set. The CDFs can be found in Appendix B. Next, we found the 80% threshold for each data-set; these can be seen in Figure 28. These thresholds varied widely between the sets of data, reducing our confidence for using the accuracy assessment algorithm in the security system design. We selected a liberal range of 12.2 knots as the max allowed error because the first four data sets had an 80% threshold between zero and three knots but the threshold is four times bigger in data set five and even larger in data set six. Such variation in the results of the accuracy algorithm implied it would produce undesirable performance in the overall security system.

	A	B	C
1	**Data Set**	**80% Threshold**	**Sample Size**
2	Marine Cadastre (Cleaned)	0.14	244276
3	Danish Maritime Authority	1.02	1893444
4	Seavision	0.23	56483
5	AISHub 20121128 Log	2.85	500000
6	AISHub 20120101 Log	12.28	500000
7	AISHub 20120730 Log	>50	500000

Figure 28. Table Containing Accuracy Assessment Error Thresholds

In summary, for layer one of our security system, we developed novel algorithms to assess two quality dimensions of AIS messages: accuracy and integrity. We then used empirical data to measure the quality of real-world AIS traffic. From these empirical results,

a threshold was established to separate high quality data. High quality data would be assumed to be valid (i.e., real), while low-quality data would be assumed to be falsified data.

3.3.2. Layer 2: Arbitrary Data Injection Prevention

In this section we explore the methodology used for layer two of the design. The technique used and the way we configured it to work specifically for AIS is explained in the later part of the section.

Arbitrary data injections are distinct from contact spoofing injections: The former can be a non-AIS formatted sequence of bits or characters, malicious code, or other types of malicious payloads that are not used for the purpose of creating ghost ships. One such example comes from Steven Marion's [14] attack sentences seen in Figure 29. These sentences carry a 'malicious' payload, the hello world script. Sentences carrying malicious payloads will still be accepted by an AIS and simply be parsed to fill in the fields defined for the message type. The objective of the selected arbitrary data injection prevention technique is to take away the ability of the adversary to freely place bits or characters anywhere in the NMEA-0183 sentence. By minimizing the degree to how arbitrary a NMEA-0183 sentence can be, the adversary's payload capacity is diminished.

```
!AIVDM,1,1,,A,B52NwHh0I=lSTRT?5D3Q2l?5oP06,0*7A
!AIVDM,1,1,,A,B52NwHh0I=lSTE0printwwccHell,0*3C
!AIVDM,1,1,,A,B52NwHh0I=lSTE1owwWorld?ccvv,0*3C
!AIVDM,1,1,,A,B52NwHh0I=lSTE2KILLQ2l?5oP06,0*7A
```

Figure 29. Marion's Malicious Sentences to be Transmitted. Source: [14].

3.3.2.1. Selected Method for Preventing Data Injections: Allow-list

Allow-listing is the method selected to meet the data injection security objective. Allow-listing is a common cyber-security practice for input data sanitization. An allow-list is a register of entities that are permitted into the system while everything else is blocked. This method is a better security practice than its counterpart, deny-listing. A deny-list is a register of entities that are not allowed in the system. The default of a deny-list is to allow all entities except the ones on the list [23]. The allow-list approach is better for AIS because there is no database that contains historical attacks towards the system, making it

nearly impossible to create an effective deny-list since this would require knowledge about all possible existing attacks. Allow-lists only care about what is allowed, for this reason it does a better job than deny-listing in defending against novel attacks.

3.3.2.2. Allow-Listing the SOG and ROT Parameters

The allow-lists for the design were generated using empirical data from real-life AIS traffic; they specifically check the SOG and rate-of-turn (ROT) parameters of position reports (message types 1, 2, and 3 only). The hypothesis that led to this specific design is that SOG and ROT have a relationship to the navigation status of a vessel. We predicted that vessels in each navigation status were going to have a range of standard SOG and ROT values. For example, it is expected that vessels that are moored, anchored, or aground will have zero speed and zero turn rate. Using millions of samples of AIS data, the normal range of values in SOG and ROT with respect to each of the 15 navigation statuses were statistically attained. This layer of the security design works in four steps:

1. Get navigation status from message.
2. Compare reported SOG value to allow-list specific for navigation status.
3. Compare reported ROT value to allow-list specific for navigation status.
4. If both values are on the list, permit message into system; if not on the list, deny.

This allow-list forces an adversary to meet the requirements for allowed combinations of values in the SOG, ROT, and navigation status fields. This design ultimately aims to take away the ability of using these three fields for arbitrary data.

3.3.2.3. Dynamic Allow-List Using The Vessel's Physical Location

The allow-listing design is expanded further by implementing a dynamic allow-list that checks the coordinates of incoming reports in comparison to the implementing vessel's own coordinates. This expansion includes implementation for message type 18. The typical range of AIS is 15-20 nautical miles but can be extended to 40-60 nautical miles [24]. The maximum range (60 nm) is approximately equal to one degree in geographical coordinates. The concept of the design is to create an allow-list that leverages the cyber-physical characteristic of voyage networks by dynamically changing the allow-list to coordinates that are within 60 nautical miles from the position of the implementing vessel. This greatly reduces

the allowed values in the latitude and longitude fields, taking away approximately 55 bits that an adversary could have used for a malicious payload. While the situational awareness of distant vessels is eliminated, we believe the security gains outweigh this loss.

3.3.2.4. Allow-listing Overview

Ultimately, the combination of the discussed allow-lists take away approximately 77 out of 168 (46%) bits to be used for injection attacks. Furthermore, these bits are scattered through the NMEA-0183 sentence, making a successful attack even more difficult. A flow diagram of the allow-listing security system is illustrated in Figure 30.

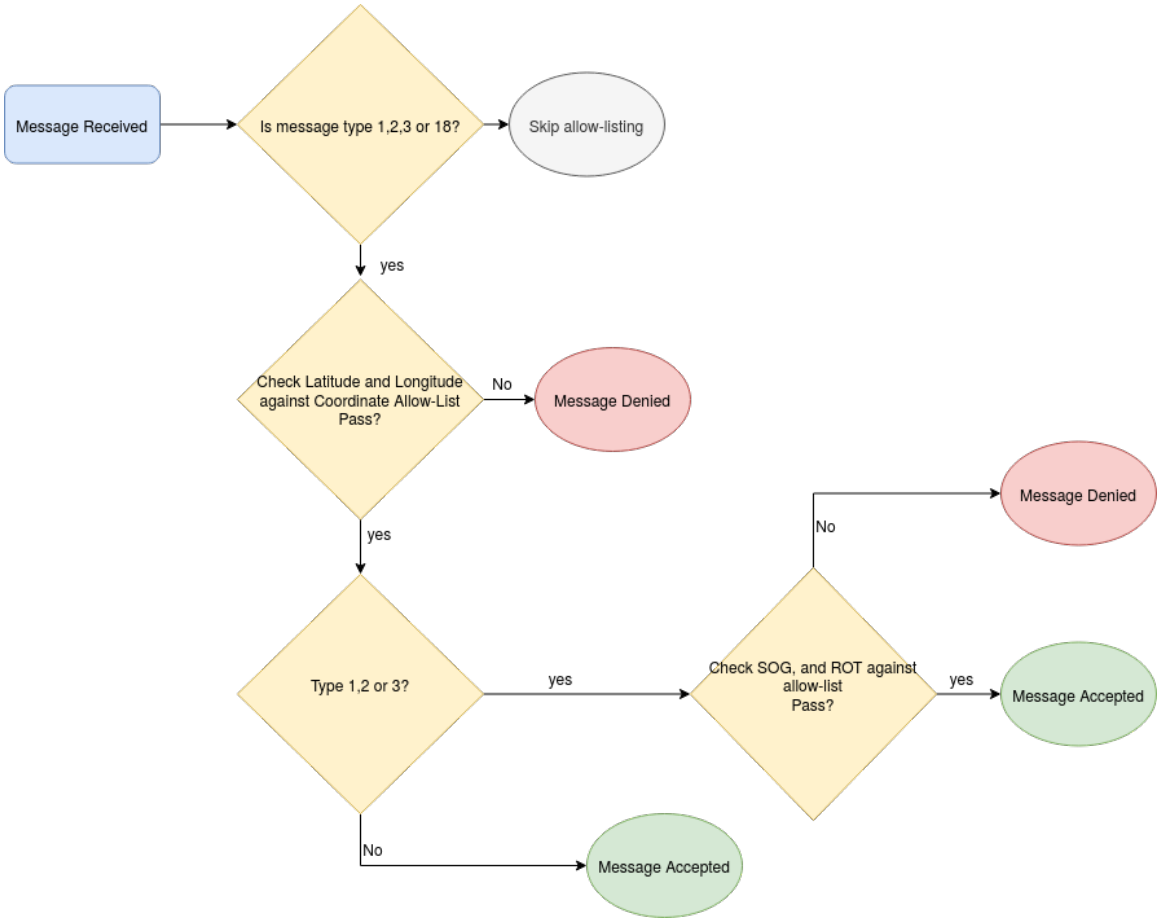


Figure 30. Design: Allow-list Process

3.3.2.5. Using Empirical Data to Create Allow-Lists

Four data sets totaling over 88 million samples of message types 1, 2, 3 were analyzed to find the typical range of values in speed and in rate-of-turn with respect to the navigation status. Data sets one and two come from global satellite logs, data set three is from MarineCadastre.gov [20], and data set four was attained from the Danish Maritime Authority database [21]. The objective of this analysis was to empirically quantify the range of values that are normal for both these fields.

First, the data was pre-processed to separate the samples by navigation status. Second, the PDF and CDF of the ROT and SOG values for each subset (subsets separated by navigation status) were generated using MATLAB. The PDF is useful to visually illustrate the normal behavior of vessels under each status while the CDF tells us the percentage of samples that fall under or above a threshold. The allow-lists were then created using the resulting statistical behavior seen in the distributions.

Too many distribution graphs were gathered to be fit promptly into this thesis. For this reason, the data is condensed into Figures 31 and 33. The normal SOG range for each data set is illustrated in the graph in Figure 31.In both Figures 31 and 33, data 1 and data 2 are satellite logs from dates 01/01/2012 and 07/30/2012, data 3 is the MarineCadastre.gov [20] data, and data 4 is the Danish Maritime Authority [21] data. Figure 31 illustrates the range for which 85% of samples in each data set will have a speed value within the limits of that range. Three desirable characteristics were paid attention to when creating the SOG allow-list:

- Allow over 85% of legitimate samples to pass.
- Make the allow-list as short as possible, while maintaining effectiveness.
- Use logic that says different statuses result in different behaviors.

Weighting these three points with the statistical results of the data resulted in the SOG allow-list in Figure 32. The first column defines the navigation status while the second column defines the allowed speed for each status.

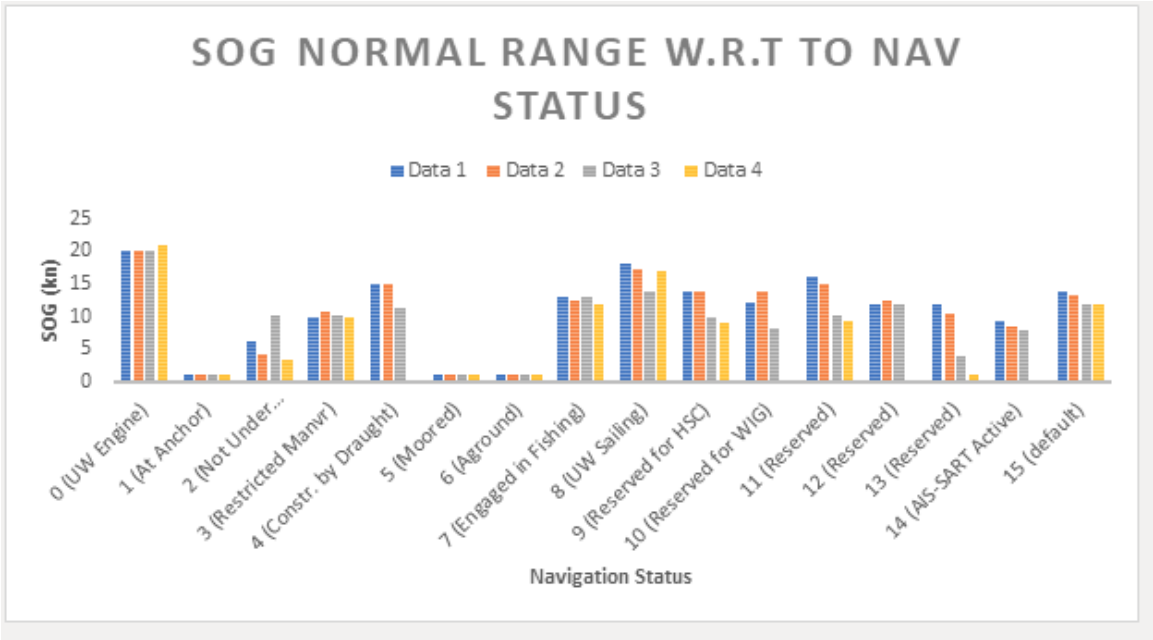


Figure 31. SOG Normal Range with respect to Nav Status for 4 Data Sets

Nav Status	SOG Allow-List	Notes
0	[0 20]	
1	[0 1]	
2	[0 15]	7-15 normal in data 1&2 (not shown in graph)
3	[0 11]	
4	[0 15]	Data set 4 had some significant deviation
5	[0 1]	
6	[0 1]	
7	[0 13]	
8	[0 18]	
9	[0 14] [26 38]	26-38 normal in data 1,2, 3 (not shown in graph)
10	[0 15]	
11	[0 16]	
12	[0 16]	
13	[0 12]	
14	[0 9.5]	
15	[0 14]	

Figure 32. SOG Allow-list for 16 Navigation Statuses

Similarly, the statistical distribution of the ROT values was used to create the ROT

allow-list. The normal range of values are illustrated similarly as before in Figure 33. Figure 33 illustrates the range for which 95% of samples in each data set will have a ROT within the limits of that range. The behavior of the turn rate distribution differs from the speed distribution; for this reason, the desirable characteristics for the ROT allow-list are slightly different:

- Allow over 95% of legitimate samples to pass.
- Make allow-list as short as possible, while maintaining effectiveness.

Notice that for the SOG allow-list, the acceptable data range was selected to contain 85% of legitimate samples. For the ROT allow-list, the acceptable data range was selected to contain 95% of samples. This is due to the compactness of ROT values around zero. This characteristic made it easy to maintain a short allow-list while still keeping most of the samples within range. The ROT allow-list can be seen in Figure 34. ROT values of -128,-127, and 127 are always allowed as these are common default values.

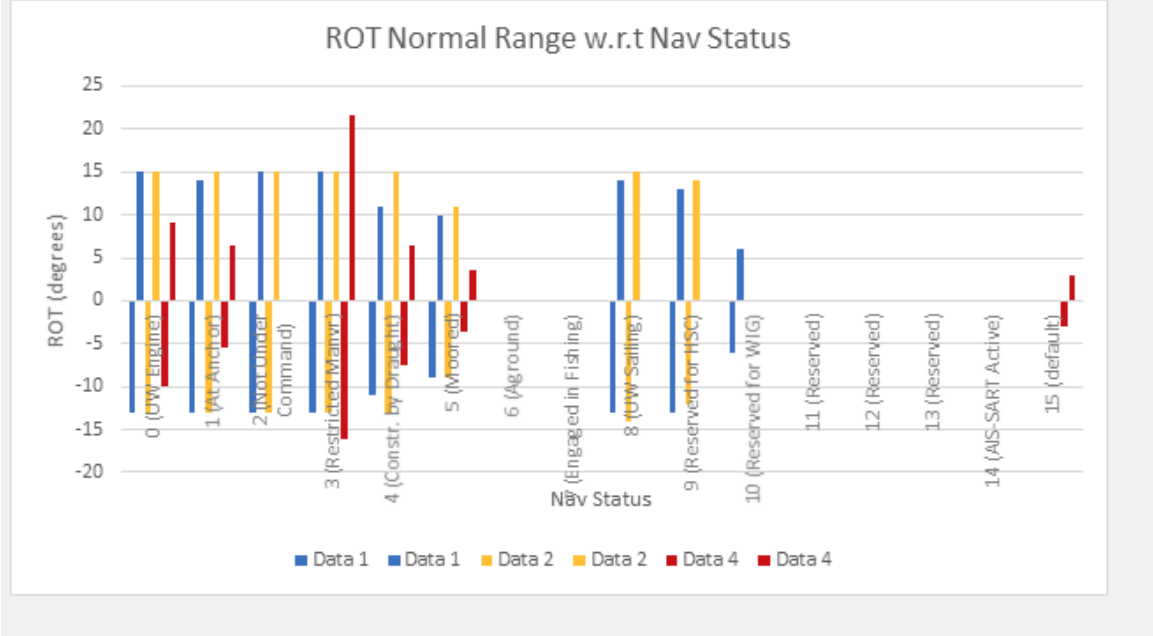


Figure 33. ROT Normal Range with respect to Nav Status for 4 Data Sets

Nav Status	ROT Allow-List	Notes
0	[-13 15]	include [-128, -127, 127]
1	[-13 15]	include [-128, -127, 127]
2	[-13 15]	include [-128, -127, 127]
3	[-16 21]	include [-128, -127, 127]
4	[-13 15]	include [-128, -127, 127]
5	[-9 11]	include [-128, -127, 127]
6	[0]	include [-128, -127, 127]
7	[0]	include [-128, -127, 127]
8	[-14 15]	include [-128, -127, 127]
9	[-13 14]	include [-128, -127, 127]
10	[-6 6]	include [-128, -127, 127]
11	[0]	include [-128, -127, 127]
12	[0]	include [-128, -127, 127]
13	[0]	include [-128, -127, 127]
14	[0]	include [-128, -127, 127]
15	[-3 3]	include [-128, -127, 127]

Figure 34. ROT Allow-list for 16 Navigation Statuses

3.3.3. Layer 3: Data Modification

The last layer of the security system has one purpose: to corrupt potentially-malicious data. The allow-list is designed to prevent arbitrary data injection attacks, but it should not be assumed that the allow-list will be accurate (or successful) 100% of the time. The data modification layer is designed to alter the values of messages that pass security layer one and layer two checks in order to disrupt the information flow or payloads potentially coming from malevolent actors. Certain computer objects like assembly code or machine language code need to be exact in order to execute the function they are meant for. If slight modifications to objects of this type are made, it could possibly disrupt an adversary's code or command and control process. Furthermore, if the adversary is simply using AIS messages to covertly transmit information, there is a possibility the information will be corrupted when the data is modified.

As such, our security layer three modifies six out of the 16 fields in message types 1,2, and 3: rate-of-turn, speed-over-ground, longitude, latitude, course-over-ground, and heading. The same is done for messages type 18, except for the (non-existent) ROT field.

Corrupting the information in legitimate messages to the point where it becomes unusable is counter-productive and therefore should be mitigated. For this reason, our modifications to each one of the six fields is so minuscule the crew would not even notice the change and the change would not affect the safety of the ship.

3.3.4. System Overview

The final design of the security system is illustrated by the flow diagram on Figure 35. First, the allow-list layer will check that the speed, rate-of-turn, and coordinates of the vessel are 'allowed', i.e., that the received values fall within determined "acceptable" ranges. If the message is allowed, it gets sent to the quality dimensions assessment layer. The MMSI of the vessel is checked to find out if previous position reports exist for the current vessel. If previous reports exist, the integrity assessment and accuracy assessment processes are executed. Next, if the message has been allowed through the initial layers, it is processed by the data modification layer before being forwarded to the rest of the system.

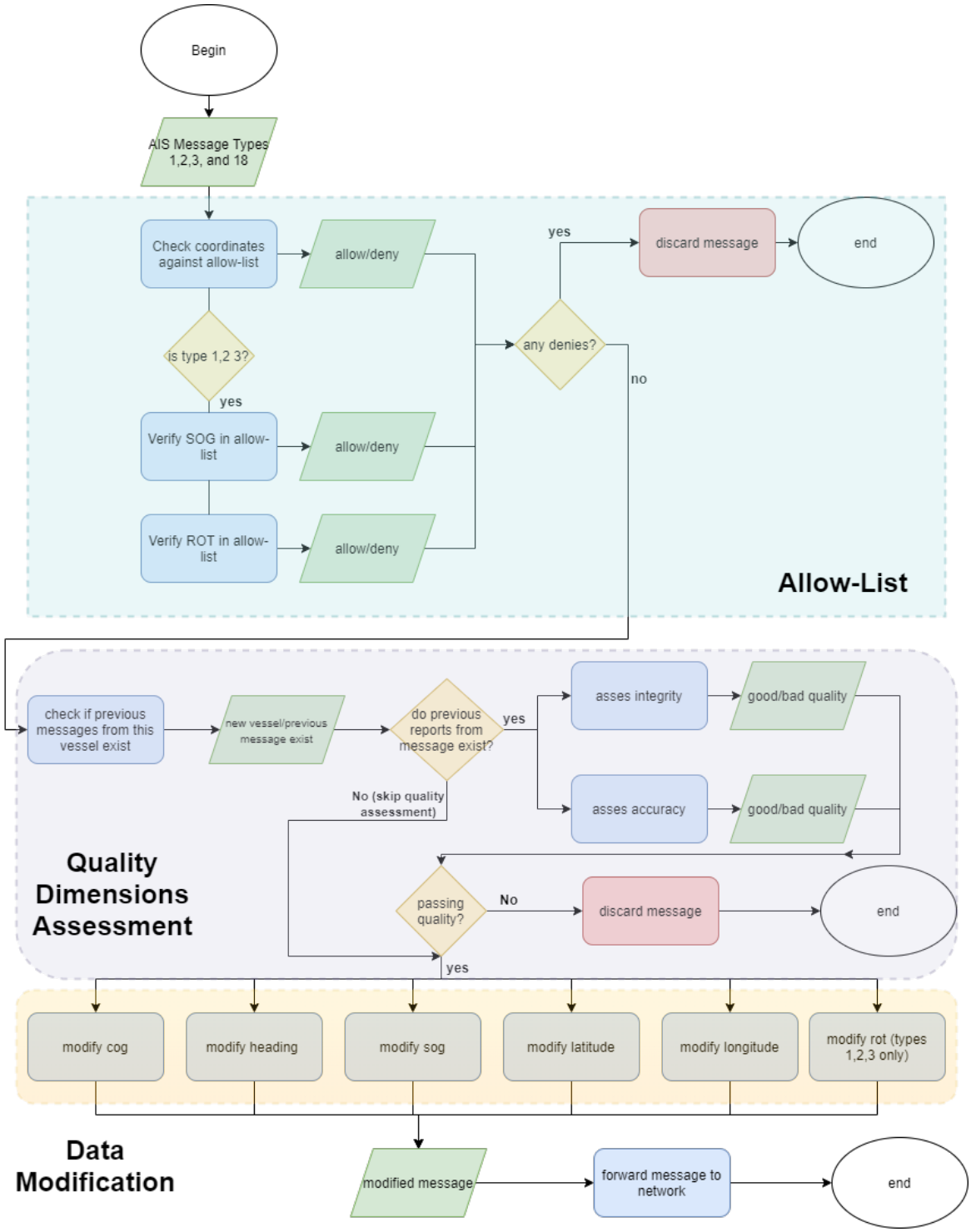


Figure 35. Security System Design Overview

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Experimentation Methodology

In this chapter we describe the methodologies for conducting experimental trials to measure the performance of the security system. The first section describes the virtual environment where experimental trials are conducted. In the second half of this chapter, we describe the tests that we applied to our three-layer security system.

4.1. Simulating The Voyage Network Environment

The next step in this research requires testing the security system software in a realistic network environment. This was done by simulating the EEMS lab environment on a Linux machine. An open-source application named *VDRplayer.py* is used to send AIS messages as UDP packets. The security system software is implemented between a receiving socket listening for incoming messages from the VDR-player, and a sending socket that packages up the messages that are to be forwarded for the rest of the network.

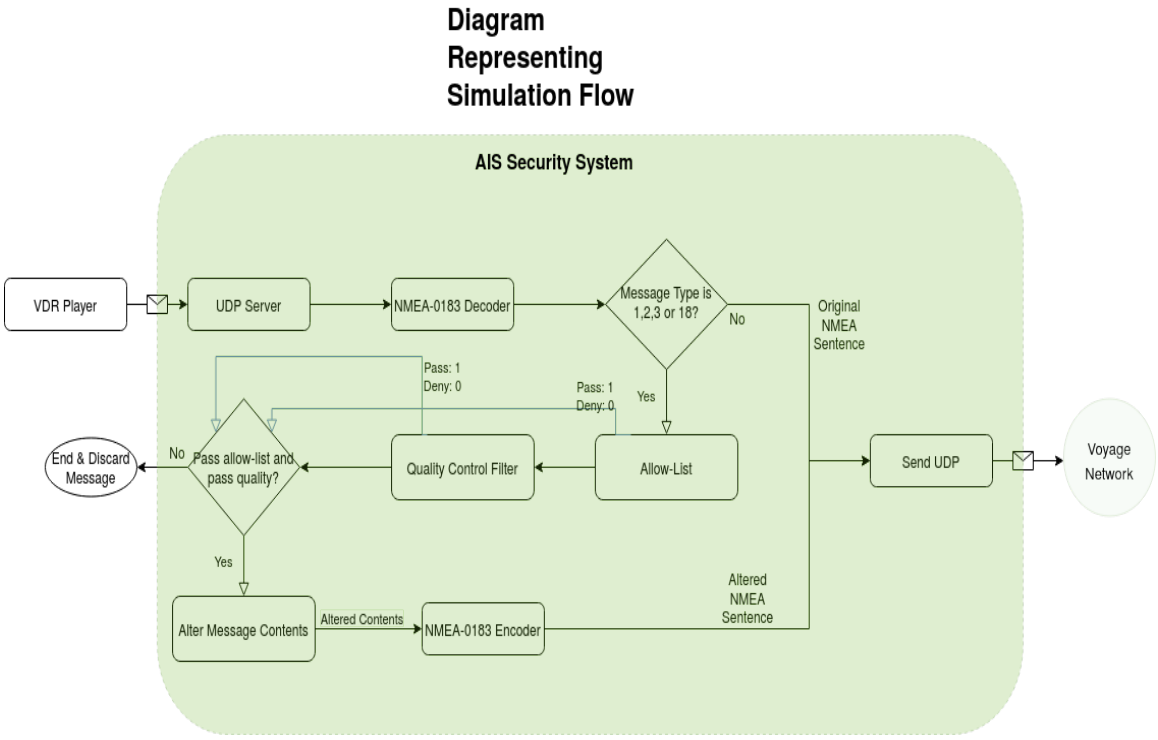


Figure 36. Security System Simulation Diagram

4.1.1. Software

The security system and the simulation are both completely written in Python V.3.7.10. The program *ais_security.py* is the main file and it controls the sequence of events. This program opens up a socket to listen on, controls the execution of the three security layers, and forwards the message via UDP. The program *ais_security_functions.py* holds the algorithms necessary for the security system. This script is separated into three Python classes: *allowList, alter_message, and quality_control*. Each one of these classes contain multiple functions that implement the security algorithms. Another custom program, *encoder.py*, encodes the contents of the message into NMEA-0183 sentences after they have been analyzed and modified. Lastly, *ais_udp_client.py* implements a UDP socket at client side and sends the encoded message to a server. The code for these Python programs can

be found in Appendix C and on the NPS GitHub page [25] :

pyais The received NMEA-0183 messages are decoded using the *pyais* library [26]. This module supports all 27 message types but it cannot decode specific payloads; these apply to message types 6, 8, 26, and 27. The decoded message contents are stored in a dictionary structure for easy access.

socket Another important module used in this program is *socket* [27]. This module allows the user to easily create sockets. Sockets are used to send messages across a network. The main functions for this module are socket(), bind(), listen(), send(), and recv(). These four functions allow the user to create a socket, receive incoming messages, and send messages. This was used for receiving and sending the AIS NMEA sentences via UDP.

VDRplayer.py *VDRplayer.py* [28] is a program that will stream a file containing NMEA data. Each line is read and sent via UDP or Transmission Control Protocol (TCP) [28]. This script is important for the simulation as it is the substitute for the AIS transponder and sensor adapter.

4.1.2. Workstation

Implementing the security system on an inexpensive computer, something like a Raspberry Pi, is ideal. The difference in computing power between the workstation, where the software is tested, and a Raspberry Pi will impact the performance of the system. The specs of the workstation used in this research are listed:

- OS: Linux Ubuntu 16.04.7 LTS
- CPU: Intel Xeon E5-2630 v2 64 bit @ 2.6 GHz, 24 cores
- GPU: Nvidia GK106GL [Quadro k4000]
- RAM: 64GB

4.2. Testing

This section describes the tests we applied to our three-layer security system.

4.2.1. Fuzzing the System

"Fuzz testing, or fuzzing, is a dynamic application security testing technique for negative testing. Fuzzing aims to detect known, unknown, and zero-day vulnerabilities" [29]. This testing technique was used to find out if the security system would detect and prevent arbitrary data injections. Using a NMEA-0183 sentence generator, we were able to create thousands of samples containing erroneous data. The synthesized messages were then sent via *VDRplayer.py* to simulate an injection attack. The goal for our security system was to deny and discard all these arbitrary messages.

Although testing the system with real-life malicious data would be a better standard, there is no data base that keeps logs of malicious AIS attacks. Fuzzing is a good standard since the nature of all possible AIS attacks is unknown.

random_NMEA_generator.py A program named *random_NMEA_generator.py* was created to generate random NMEA messages of types 1, 2, and 3. This program works by generating a random value for each of the fields in a message. The value is chosen from a uniform distribution in the range of the minimum and maximum values for the specific field. The values are then encoded into NMEA sentences using *encoder.py*. The range for each of these fields is listed:

- type: [1-3]
- repeat: [0-3]
- mmsi: [100000000-999999999]
- status: [0-15]
- ROT: [-128 127]
- SOG: [0 1023] (divide by 10 to get actual speed)
- accuracy: [0 1]
- lon: [-134,217,728 - 134,217,727] (divide by 600,000 for degrees) [-223.696213333 - 223.6962116666]
- lat: [-67,108,864 - 67,108,863] (divide by 600,000 for degrees) [-111.8481066666 - 111.848105]
- COG: [0 - 4095] (divide by 10)
- heading: [0 - 511]
- timestamp: [0-63]

- maneuver: [0-3]
- raim: [0-1]
- radio: [0-524287]

4.2.2. Testing Against Legitimate Data

A common problem seen in detection systems is high false alarm rates. A false alarm is when the system will claim a good sample is malicious. The false alarm rate is described in percentages to describe how often a good message will be detected and labeled malicious. In order to test the false alarm rate of the system in this research, traffic that is assumed non-malicious is injected. The testing data comes from the archive of satellite logs maintained on a hard drive in the EEMS laboratory. Different logs than the ones used for the design were used for testing, these new testing logs are from dates 08/25/2012, 01/27/2013, and 04/29/2013. We make the assumption that all messages in these logs are real, though we can't fully know if there exists contact spoofing or other erroneous data already within the data sets. Given our assumptions, testing against this data means that any and all detections made by the system will be a false alarm.

4.2.3. Testing "Pac-Man" Script

Rubinstein [13] generated a spoofing attack script during his research that resembles a real attack. No information on how he generated this traffic is known from his work. The attack he formulated was also tested against our system and the vulnerability of our design to this attack will be analyzed.

4.2.4. Computing Speed and Performance

The computing performance of the software needs to be analyzed. The CPU utilization and RAM utilization need to be recorded. Also, our performance metric of 'messages-per-second' needs to be determined. Low RAM and CPU utilization is ideal since the goal is to be able to implement this on a low-power computer. Measuring how many messages-per-second the system will be able to process is crucial as it will be a deciding factor on whether it can scale to high traffic areas.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Results

Once the design was complete, the final step was to measure its performance. The performance of individual layers, as well as of the system as a whole, is analyzed. First, the performance of the layer implementing the quality dimensions algorithm is tested by injecting the system with Rubinstein's [13] "Pac-Man" script and legitimate AIS traffic. Second, the allow-listing layer is tested by injecting legitimate traffic and by fuzzing the system. Fuzzing the system does not invoke the quality dimensions layer since multiple reports from the same vessel are necessary to activate the algorithm.

One thing to note is that we are testing in a lab environment where the location is static and we are not recording AIS messages in the local area, therefore the allow-list analyzing the coordinates of incoming vessel traffic is set to allow all messages to pass; the dynamic coordinate allow-list is not tested in this thesis.

Finally, the processing speed of the system is measured in two ways. First, messages are injected at low speeds in order to measure the time it takes to process a single message. Second, bulks of messages are injected at a high velocity to analyze the effects on processing time. Meanwhile, the CPU and RAM utilization are recorded when the system is running.

5.1. Performance of Quality Dimension Approach

The first tests conducted were on the spoofing detection layer. The metrics of interest are false alarm and true detection rate. The resulting rates for this layer were measured individually from the allow-list. Furthermore, the performance of the integrity assessment and accuracy assessment algorithms were measured individually as well. This provides an insight about how much these two algorithms agree, how they differ, and the synergy when they are put together.

5.1.1. Vulnerability to "Pac-Man" Script

Rubinstein's "Pac-Man" script consists of only 42 samples; this is not enough to make any claims about the capabilities of the system. Even then, it was interesting to find out how this security design held up against a fellow researcher's attack. The results are described in the screenshot in Figure 37. All samples are considered spoofed in this scenario. The system's overall detection rate is measured at 88% while the integrity and accuracy algorithms had a detection rate of 75% and 93%, respectively. These results tell us that the accuracy detection scheme is stricter as it only allowed three samples to pass through, while the integrity algorithm allowed ten. The system allowed only five samples to pass in total; two of these automatically passed because they were the first report of each of the two vessels in this data. This means that the three samples not detected by the accuracy algorithm were also not detected by the integrity algorithm, meaning the algorithms agree but one is stricter than the other.

After analyzing the traffic more thoroughly, we discovered that the timestamp for all vessels remained static. This has a critical effect on the algorithms. For the integrity algorithm, the position reported in the first out of two consecutive reports would also be the expected position. This means the error is simply the distance between the two position reports. The accuracy algorithm would fail as it tried to divide the distance by zero (refer to Chapter 3 for the algorithms). This division by zero problem in the accuracy algorithm is prevented by checking that the latitude and longitude are exactly the same whenever the timestamps are exactly the same. When the latitude and longitude are changing but the timestamp is static, the message is denied.

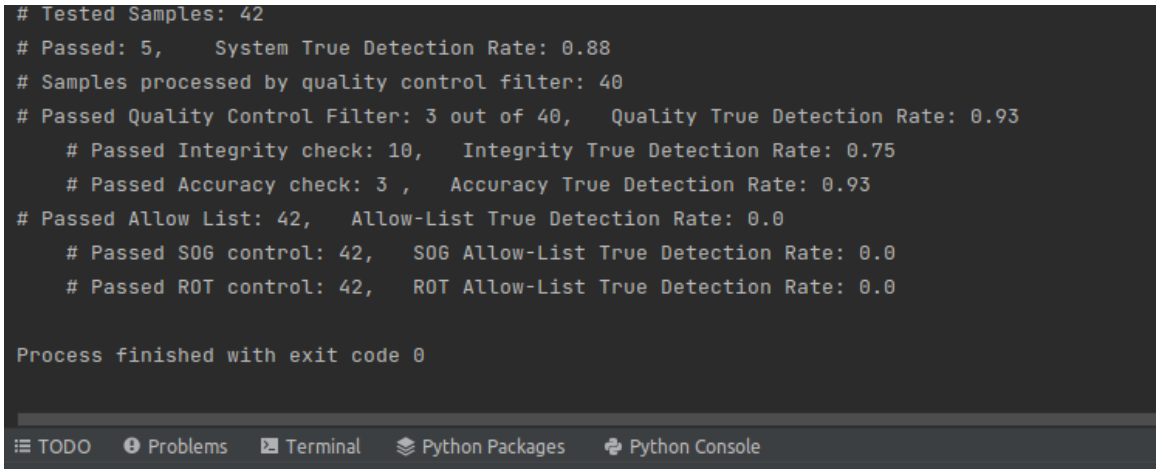


Figure 37. Results: Vulnerability to Pac-Man Script

5.1.2. Reaction to Legitimate Data

Three randomly chosen satellite data logs from the archives were used for testing. The three logs are recordings from dates 08/25/2012, 01/27/2013, and 04/29/2013. None of these data sets used in creating and proof-of-concepting the designed security layers. Three testing trials, one for each data log, were then conducted. Each trial would end once 50,000 samples were injected. The data sets include very few samples that are not message types 1, 2, 3 or 18; these were simply passed through the system but not considered in the statistics. The 'Tested Samples' metric provides the number of messages that were actually tested out of the 50,000 injected. The metric of interest for these trials is the false alarm rate. The assumption is that all samples are legit, therefore true detection rate cannot be established.

The results from these three trials can be seen in Figure 38. In trial one, the quality dimensions filter (quality control filter), had a false alarm rate of 32%. The integrity and accuracy algorithms had false alarm rates of 26% and 29%. For integrity, we expected a false alarm rate of 20% given the design used the 80% threshold mentioned in Chapter 3. Furthermore, the combined false alarm rate was slightly higher than the individual rates, meaning that the algorithms mostly agreed on what was detected but each one also had a few false alarms the other did not detect. The results from trial two and three were significantly different from trial one. Trial two had an overall false alarm rate that was lower at 12%, and

trial three resulted in an even lower false alarm rate of 3%. One thing all three trials agreed on was the synergy of the two algorithms since the overall false alarm rate was about the same or slightly higher than the individual rates every time. The variation of the results can possibly mean that the metadata of each of these logs vastly differs. Although we do want low false alarm rates, this variation in the results is not ideal as it impedes us from making a conclusive statement on the performance of this layer. Overall, the false alarm rate of this layer is recorded between 3% and 32% but the results gave us an inconclusive answer about the capability. The detailed results of these trials are captured in the screen grabs in Figure 39.

Testing Quality Dimensions Analysis Against Legitimate Data

Trial No.	Accuracy FAR (%)	Integrity FAR (%)	Combined FAR (%)
1	29	26	32
2	9	7	12
3	3	2	3

Figure 38. False Alarm Rate Results of Quality Dimensions Analysis Tested against Legitimate Data

```
# Tested Samples: 50000
# Passed: 37591,    System FAR: 0.25
# Samples processed by quality control filter: 29180
# Passed Quality Control Filter: 19831 out of 29180,   FAR|processed:0.32
    # Passed Integrity check: 21737,   FAR|processed: 0.26
    # Passed Accuracy check: 20718,   FAR|processed: 0.29
# Passed Allow List: 46374,   FAR: 0.07
    # Passed SOG control: 47983,   FAR: 0.04
    # Passed ROT control: 48335,   FAR: 0.03

Process finished with exit code 0
```

Problems Debug Terminal Python Packages Python Console ▶ Run

(a) Trial 1: log from 08/25/2012

```
# Tested Samples: 50000
# Passed: 44200,    System FAR: 0.12
# Samples processed by quality control filter: 37063
# Passed Quality Control Filter: 33302 out of 37063,   FAR|processed:0.1
    # Passed Integrity check: 34622,   FAR|processed: 0.07
    # Passed Accuracy check: 33788,   FAR|processed: 0.09
# Passed Allow List: 47737,   FAR: 0.05
    # Passed SOG control: 48549,   FAR: 0.03
    # Passed ROT control: 49128,   FAR: 0.02

Process finished with exit code 0
```

Problems Debug Terminal Python Packages Python Console ▶ Run

(b) Trial 2: log from 01/27/2013

```
# Tested Samples: 50000
# Passed: 46824,    System FAR: 0.06
# Samples processed by quality control filter: 20073
# Passed Quality Control Filter: 19377 out of 20073,   FAR|processed:0.03
    # Passed Integrity check: 19688,   FAR|processed: 0.02
    # Passed Accuracy check: 19441,   FAR|processed: 0.03
# Passed Allow List: 47419,   FAR: 0.05
    # Passed SOG control: 48588,   FAR: 0.03
    # Passed ROT control: 48789,   FAR: 0.02

Process finished with exit code 0
```

Problems Debug Terminal Python Packages Python Console ▶ Run

(c) Trial 3: log from 04/29/2013

Figure 39. Results: Testing against Legitimate Data

5.2. Allow-Listing Results

5.2.1. Reaction to Legitimate Data

By injecting the system with legitimate traffic, the performance of the allow-listing layer was analyzed as well. Referring back to Figure 39, the output 'Passed Allow List' is the results for this layer. These results can be seen more clearly in Figure 40. In this test, the metric of interest is the false alarm rate. In Chapter 3, it was explained that the allowed values for SOG were chosen to allow 85% of samples to pass while the ROT allowed values were chosen to allow 95% of samples to pass. Given these percentages, it was expected that the combined false alarm rate would be at maximum 20%, worst case scenario, but more likely less than 15%. The three trials were consistent in this case. The false alarm rates for the combined allow-lists are 7%, 5%, and 5% for trials one, two, and three, respectively. The combination of the allow-lists had a negative synergy for the false alarm rate. This can be seen from the fact that the overall allow-list false alarm rate is the sum of the individual rates. This means the samples they detected as dubious did not overlap.

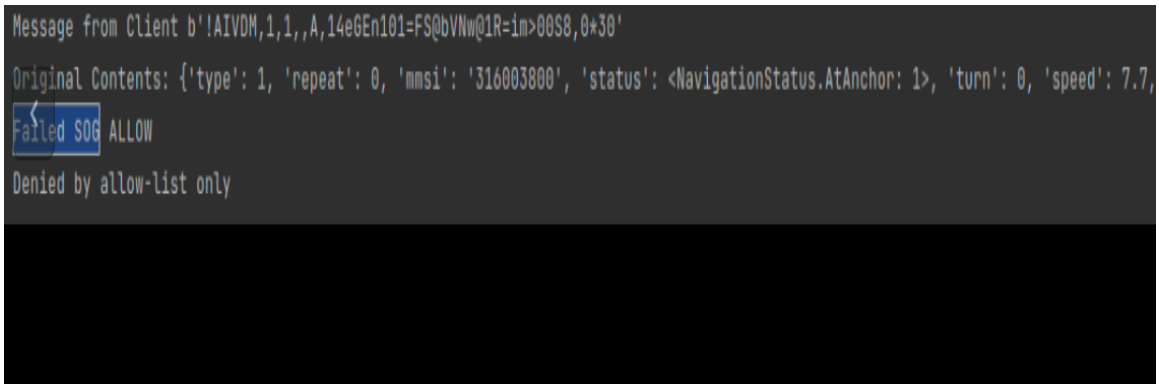
Testing Allow-list Against Legitimate Data

Trial No.	SOG Allow-list FAR (%)	ROT Allow-list FAR (%)	Combined FAR (%)
1	4	3	7
2	3	2	5
3	3	2	5

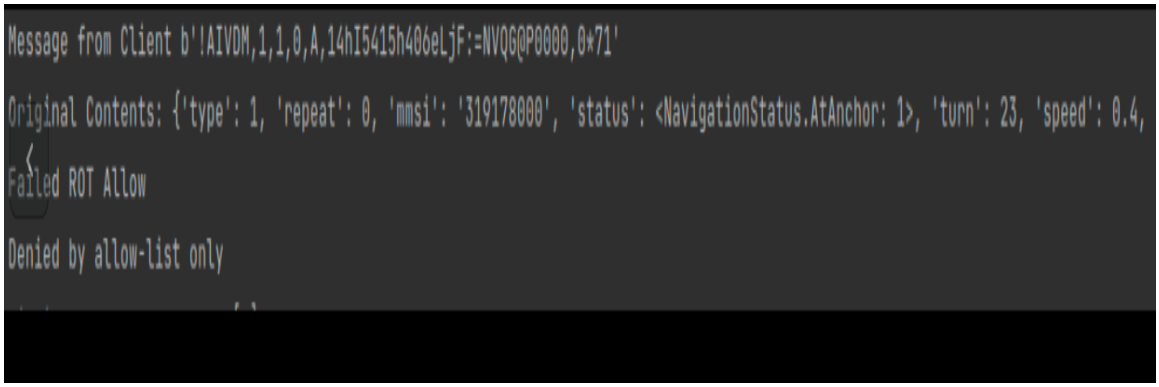
Figure 40. False Alarm Rate Results of Allow-list Tested against Legitimate Data

Cases of samples that failed the allow-list were investigated. Although the assumption is that all messages were legit, most of the messages detected by allow-listing did seem suspicious. Examples of some of these detected but suspicious samples can be seen in Figure 41. Sample 1 consists of a vessel that is supposedly anchored but has a speed of 7.7 knots. Sample 2 illustrates a vessel that is also anchored but has a turn rate of 23. Sample 3

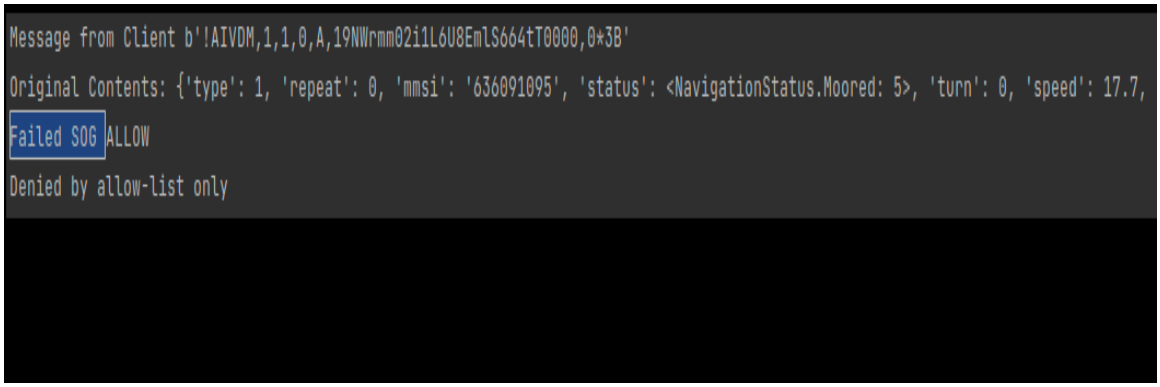
shows a vessel that is moored but is moving with a speed of 17.7 knots. Logically, samples like these should not be possible. Another significant factor that contributed to the false alarm rate is vessels in the ‘under way using engine’ category with a speed that is barely above the max allowed speed. Referring to the allow-list in Figure 32 in Chapter 3, the max allowed SOG for vessels in this status is 20 knots. Figure 42 illustrates two sample with speeds at 21.3 and 20.7 knots that were denied by the SOG allow-list. The source of this issue is that the allow-list only allows vessels traveling between 0 and 20 knots if they are ‘underway using engine’, we could increase the max allowed speed to solve for these close values.



(a) Sample 1: Sample With Navigation Status 'at anchor' Fails SOG Allow-list



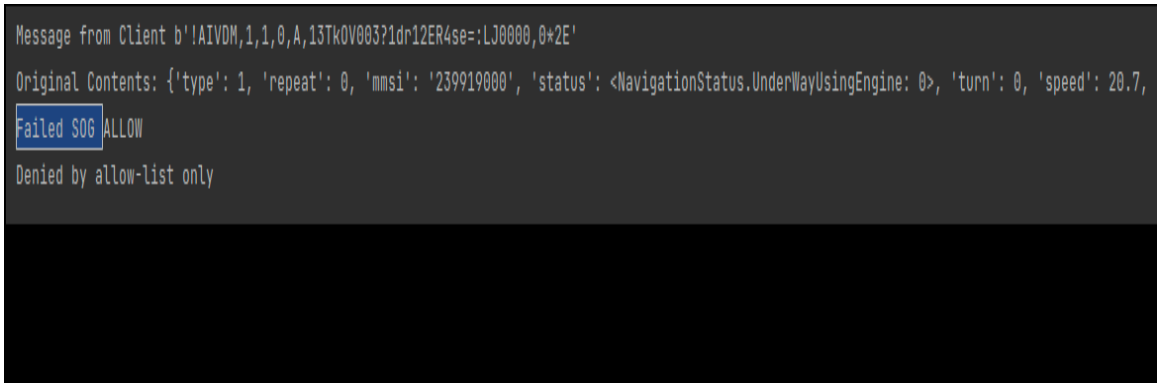
(b) Sample 2: Sample With Navigation Status 'at anchor' Fails ROT Allow-list



(c) Sample 3: Sample With Navigation Status 'moored' Fails SOG Allow-list

Figure 41. Allow-list Denied Cases of Suspicious Messages

(a) Sample 4: Vessel 'under way using engine' With Speed 21.3 Knots



(b) Sample 5: Vessel 'under way using engine' With Speed 20.7 Knots

Figure 42. Vessels 'under way using engine' With Speed Just Outside Allow-list

5.2.2. Performance Against Fuzzing

Three trials of fuzzing attacks were conducted to test the reaction and performance of the allow-list. Each trial consists of an injection of 50,000 randomly generated NMEA-0183 sentences. The metric of interest for these trials is true detection rate. Since all samples are synthetically generated, it would be ideal for the allow-list to deny them all. The results of these trials are seen in Figure 43. More detailed results can be seen in the screen grabs in Figure 44. All three trials show a detection rate of 99%; out of the 50,000 samples injected, between 428 and 449 passed. Also, in all three trials, the SOG allow-list true detection rate is 88% and the ROT allow-list rate is 93%. The difference between the combined rate and

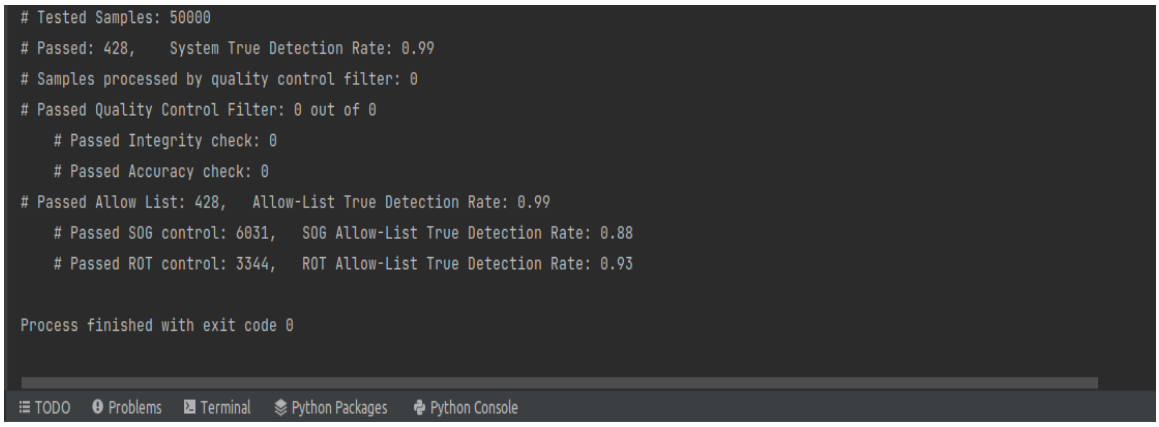
individual rates suggests a positive synergy between the lists for detecting arbitrary data.

Furthermore, the true detection rate can be improved by adding the coordinates allow-list. After analyzing the fuzzing samples that passed undetected, it was discovered that many of those samples would not have passed the coordinates allow-list. One such case is demonstrated in Figure 45. The sample is reporting a latitude of -96 degrees. This latitude is possible in the AIS standard but invalid in real life. Invalid coordinates should never be reported by a legitimate vessel. The coordinates allow-list would prevent such examples from passing.

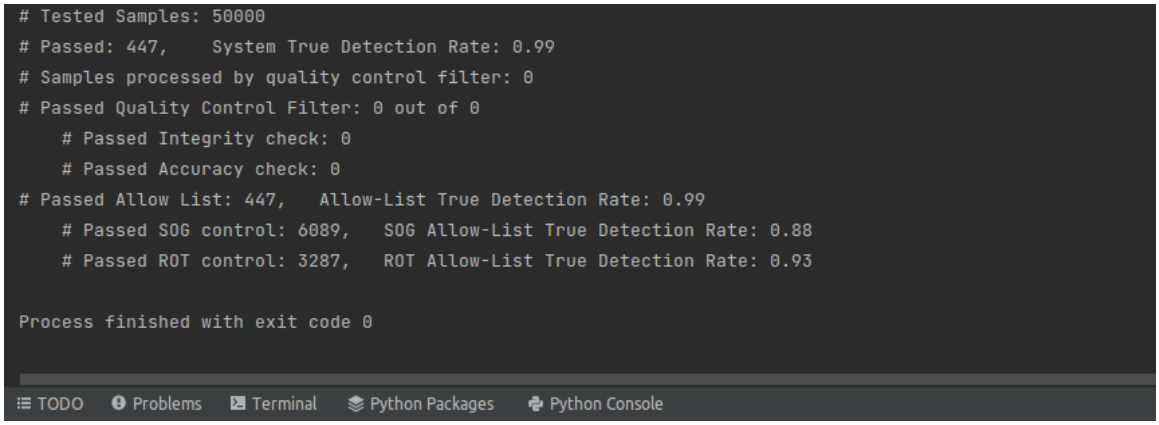
The cases that did get through, and had allowable coordinates, happened to be messages that looked like 'normal' traffic. The allow-list is not meant to detect messages that look like typical traffic making the last 1% of fuzzing messages difficult to detect.

Fuzzing Allow-list	
Trial No.	Detection Rate (%)
1	99
2	99
3	99

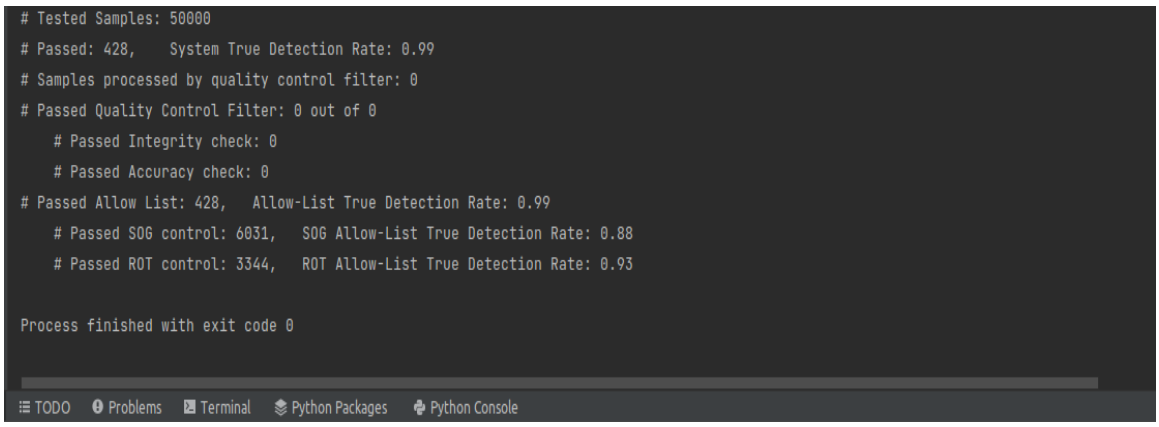
Figure 43. Allow-list Fuzzing Results



(a) Fuzzing Trial 1 Results



(b) Fuzzing Trial 2 Results



(c) Fuzzing Trial 3 Results

Figure 44. Fuzzing Results

Figure 45. Fuzzing Sample Case With Impossible Coordinates

5.3. System Computing Performance

5.3.1. Message Processing Speed

The processing time of the security system was measured utilizing Python's *time* library. The function *time_ns* invokes a high precision timer capable of measuring time elapsed in nanoseconds. This timer is set to begin counting as soon as a message is received, and it stops counting after the send message function is executed. In summary, all functions, starting with the NMEA-0183 decoder, up to the send message via UDP line, are included in this time frame. Seven trials were conducted for which the send speed of *VDRplayer.py* was increased sequentially. The time it takes *VDRplayer.py* to send a message is in the 10^{-5} seconds range so it is considered insignificant for this experiment. The results for these trials are illustrated in Figure 46. One thing to note before analyzing these results is the difference between the number of samples received by the system, and the number of messages forwarded from the system, this is because some messages were denied and therefore, not forwarded.

Trial one measures the time it takes to process a single message. This was done by sending messages to the system very slowly. Trials two and three measure processing time for bulks of messages being received fairly fast. It can be seen that the median processing time for these three trials is approximately two milliseconds. In trials four and five, the time between messages received was reduced to a value faster than the processing time seen in the first three trials. It was predicted that some samples would be dropped, but interestingly,

this did not happen, and the processing time was actually halved. We believe that this halving in processing time is highly likely due to the CPU becoming overclocked to meet the high load of messages. Lastly, in trials six and seven, the time between messages was further reduced to 0.5 milliseconds. Comparing the number of samples sent in these last two trials to the samples sent in trials three, four, and five, it can be seen that messages were dropped when messages were received every 0.5 milliseconds. Interestingly, the processing time for these trials is even faster than the previous, but still not under 0.5 milliseconds, hence the dropped messages.

This data is separated into three zones: green, yellow, and red. The green zone describes the safe speed at which messages can be received. It is safe to assume that messages received three milliseconds apart can be easily handled by the system. The yellow area describes the caution area for which messages being received faster than the two-millisecond processing time could cause some problems. It is believed that the CPU was overclocked when messages were being received this fast; this could lead to some issues. Lastly, the red area is the danger zone in which messages are being received too fast causing the system to drop some of them. Looking at the processing time of around 0.8 milliseconds for these last two trials, it can be assumed that sending more than 1200 messages per second to the system will cause it to drop some of them. Referring to Chapter 2, in order for the system to scale well it had to be able to process 75 messages every second, the message processing time deemed from these experiments demonstrates the system is well capable of processing faster than necessary.

Trial No.	Period Between Messages	# Samples Received by System	# Samples Forwarded by System	median processing time (ms)	# messages processed / second
1	2000ms	500	313	2.09	478.5
2	100ms	10000	6573	2.13	469.5
3	10ms	50000	33768	1.84	543.5
4	1ms	50000	33768	0.941	1062.7
5	1ms	50000	33768	0.958	1043.8
6	0.5ms	50000	32978	0.769	1300.4
7	0.5ms	50000	32435	0.79	1265.8
			data set used: 20120825.log		

Figure 46. Performance: Speed Results

5.3.2. RAM and CPU Utilization

The last metrics measured for computing performance are CPU and random-access-memory (RAM) utilization. Using Ubuntu's built in command-line function, *htop*, the details of a running program can be accessed. The results from *htop* are contained in Figure 47. These results were measured as messages were being injected every three milliseconds. The metrics of interest are 'RES', 'CPU%', and 'MEM%'. The first, 'RES', stands for resident memory. This is the size of memory currently being used by the program. This is a better indicator of memory usage than 'VIRT', as the latter stands for virtual memory and it is the memory requested by the program but not how much it is actually using. The screenshot shows that approximately 37504 kilobytes (kB) of RAM is used. This number stays consistent no matter how long the program runs. Second, 'CPU%' is the fraction of CPU time used up by the program from a single core. 100% means that one core is being used completely where as 200% would mean two whole cores are being used up. The screenshot shows that approximately 40% is used, a relatively low percentage for the number of messages being processed every second. Lastly, 'MEM%', is the fraction of memory used out of the computer's total available RAM [30]. This number is 0.1%, just a tiny portion of the 64 Gigabytes available. This percentage would increase for a machine with less RAM so it is not as relevant. Overall, the CPU and RAM utilization of this program is not significant thus we assess that this security system could be implemented on a low-powered computer. Unfortunately, we did not have time for this.

HTOP Results for ais_security.py		
RES RAM (kB)	CPU (%)	MEM (%)
37504	49.6	0.1

Figure 47. Performance: CPU and RAM Utilization

CHAPTER 6:
Conclusions

This research explored a cybersecurity capability to defend against contact spoofing and data injection attacks executed via AIS position report messages for shipboard voyage networks. The architecture of the security system is a minimalist design that does not rely on expensive hardware and can be implemented while complying with ITU-R M.1371-5 and NMEA-0183 standards. Furthermore, the security system operates independently from other network devices and is self-contained within the vessel, meaning external AIS stations are not affected by the system.

The main focus throughout the research was to create a system that would greatly reduce the success of contact spoofing and data injection attacks while simultaneously making the design simple, with an ability to handle bursts of data quickly. The approach selected to defend against contact spoofing was to assess the quality dimensions of the data to differentiate between real and false position reports. This approach built on the theoretical ideas found in [19], making the ideas more tangible by using mathematical algorithms to numerically assess the quality dimensions. The two quality dimensions assessed in this research are integrity and accuracy.

The algorithms did not perform as well as intended against large amounts of AIS data from satellite logs, and the false alarm rate varied highly between tests. Although these algorithms did perform well against Rubinstein's [13] "Pac-Man" script, the results should not be taken into account since the script is simplistic and it doesn't contain enough samples to claim numerically significant results.

Our design uses an allow-listing approach to defend the network from arbitrary data injections. Millions of samples of historical AIS data were statistically analyzed to find out what 'normal' looks like for AIS position reports. Normality was established using the speed and turn rate of the vessel in comparison with the reported navigational status.

The allow-list approach yielded satisfactory results. First, when tested against historical AIS satellite data logs, the false alarm rate was calculated at less than 7%, whereas

the goal was to maintain the false alarm rate under 20%. Then, the allow list system was fuzzed to analyze how it would react to truly random data. In all three trials, the security system detected 99% of the synthetically generated fuzzing messages. Furthermore, the allow-listing design also consists of a coordinate check to prevent random values in the latitude and longitude fields of the report. The coordinate design was not tested in this research but it is a simple concept and theoretically, it should decrease the probability of a successful arbitrary data injection attack. Overall, the allow-listing approach works well in the simulated environment and should be tested further in a more realistic environment.

Another important goal in this research was to design the security system such that it requires minimal resources to operate, is simplistic, and has the capacity to scale well. In order to scale well, we needed a security system that could handle 75 messages per second as this is how many messages can be transmitted using the SOTDMA scheme. After analyzing the end-to-end execution time of the security program, it was found that the system could safely handle between 543 and 1062 messages per second when implemented on the machine described in Chapter 4. Also, the program was not computationally expensive in terms of CPU load or RAM utilization.

The program was never implemented on a Raspberry-Pi due to time constrains. The Raspberry Pi 4 has up to 8 GB of RAM, a gigabit Ethernet port, and a CPU powerful enough to run a desktop [31]. We believe that this program would perform well enough to meet the performance standards necessary on a Pi 4.

The design of our algorithms was kept simple as they were completely built using Python and there are no black boxes; the algorithms are straight forward and transparent. Overall, the system could scale well to high traffic environments without the need for expensive resources.

6.1. Recommendations

We do not recommend the contact spoofing algorithms used this research to prevent contact spoofing attacks in their current state. Our research does not show that the proposed design is robust or capable of securing the network from these types of attacks and we are not confident it will increase safety at sea. Testing the design with more realistic and known

(i.e., labeled) data, in a low-risk environment, for a prolonged period of time, could yield a better confidence level

The general approach of assessing the quality dimensions of data should be explored further. Creating the algorithms to implement our approach required creativity and a lot of testing but could ultimately be successful. One design that could prove useful is described in the 'Future Work' section found later in this chapter.

We do recommend further testing of the allow-listing design found in this research. The allow-list is a transparent concept that yields good results. One problem that does arise from this design is that it prevents the crew from seeing vessels that are acting outside of the normal range but demonstrating some dangerous characteristics (e.g., vessels traveling at very high velocities approaching the AIS station). The trade-off between preventing data injection attacks and full knowledge of what is going on at sea needs to be assessed individually by the user; user needs could require changing the limits of the allow-list. This design needs to be rigorously tested in a non-simulated environment over a prolonged period of time before being completely adopted.

We do recommend that future researchers on the topic of AIS security highly consider a self-reliant, minimalist system. It is unlikely that a security system that requires external AIS stations to perform extra work, or that requires a modification in the AIS protocols, will scale well. Furthermore, the security system should not rely on external devices on the network to be able to operate. This research illustrates a system that works independently and only relies on receiving messages via UDP. Furthermore, the results show that high processing speeds can be achieved by implementing the system mostly in software. Higher processing speeds could probably be achieved if the program is implemented in lower-level languages such as C++; Python was satisfactory for this research.

6.2. Weakness of Research

The most significant weakness of this research was the assumptions made about the data sets used for the design and in testing. The sources for the data sets include SeaVision.dot.gov [22], MarineCadastre.gov [20], the Danish Maritime Authority [21], and historical AIS satellite logs passed down from previous researchers at NPS. For all data sets,

it was assumed that the data is unmodified, presents a realistic environment of the high seas, and contain no erroneous data. These assumptions could be wrong if the agencies modified the data or if a significant amount of synthetically generated samples are already included in the data. As discussed, we identified unusual and possibly spoofed messages in the data sets used for our work, and these abnormal messages likely affected some of our results.

One possible problem that could be unaccounted for in this research is discrepancies in the timestamps of the satellite log data sets. Timestamps are on a minute basis and only contain values from 0 through 63 (seconds). It was assumed that two consecutive reports from the same vessel are not more than a minute apart and are actually in the correct chronological order in the data sets. This assumption is reasonable, but if it is wrong, it is significantly detrimental to the integrity and accuracy algorithms. The non-satellite data sets did have a UTC timestamp but it was assumed that the agencies did not remove position report samples from their data sets. If they did, this could mean that consecutive reports in our data sets are not actually consecutive reports in real life. Again, this is a reasonable assumption but if not true, the results of the accuracy and integrity algorithms should be discarded as faulty. In both cases, this assumption could be a source of the higher false-alarm rates in some of our algorithms.

Another weakness of this research is the assumption that the security system will easily be able to inject UDP packets back into the network. We believe injecting packets directly into the network is more complex based on results of [11]. In [11], they struggled to inject packets directly. The implications are that more code could be required, possibly increasing the end-to-end processing time. This is an item of future work.

Lastly, there is a concern with the equivalency of the EEMS environment to a real vessel's voyage network. Our proposed system relies on the fact that in the EEMS laboratory the AIS and sensor adapter transmit only to the switching HUB using LWE. Other vessels might not follow the same network structure and the system would have to be adapted to the local ship's network.

6.3. Future Work

AIS security is a largely unexplored topic leaving many opportunities for future researchers and developers to design novel security methods. Starting with this research, the quality dimensions analysis approach for detection of contact spoofing can be improved. More robust algorithms to assess the quality dimensions of AIS data are necessary. One improvement that could be made to the integrity algorithm explained in Chapter 3 is to increase the time between the reports being analyzed. Instead of using back-to-back consecutive reports, it might be better to use every *n* report. This would allow for a more noticeable error in the integrity of the data.

The coordinate check element of the allow-list is not implemented in this research. This segment of the allow-list should be implemented and tested using a dynamic AIS station receiving real AIS data from AIS stations within its region. The code for this allow-list is in the program *ais_security.py*, but it is not yet fully functional since we need to constantly update the latitude and longitude from our own vessel reports. Furthermore, this research only explored the allow-listing approach for message types 1, 2, 3, and 18; future work could expand the allow-list for other message types.

In this research, test were not conducted using Type Approved devices. Future work could explore how the design will react when implemented on an actual voyage network consisting of industry level devices. Even further, the design could be tested on live maritime traffic. Future work of this kind would be beneficial for improving the allow-list.

Another item for future work is to analyze how Type Approved AIS transponders react when ITU-R M.1371-5 undefined characters are present in the injected packets. We did not have time to analyze this work with the time allotted for this research.

Outside of this research, AIS security needs to be expanded to defend against possible attacks coming from all 27 message types. Our proposed algorithms and methodology become a lot more complex when considering binary messages such as 6, 8, 25, and 26. Voyage networks will never be fully secure while attack vectors in AIS are left unsecured.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: A. Results for Empirical Assessment of Integrity

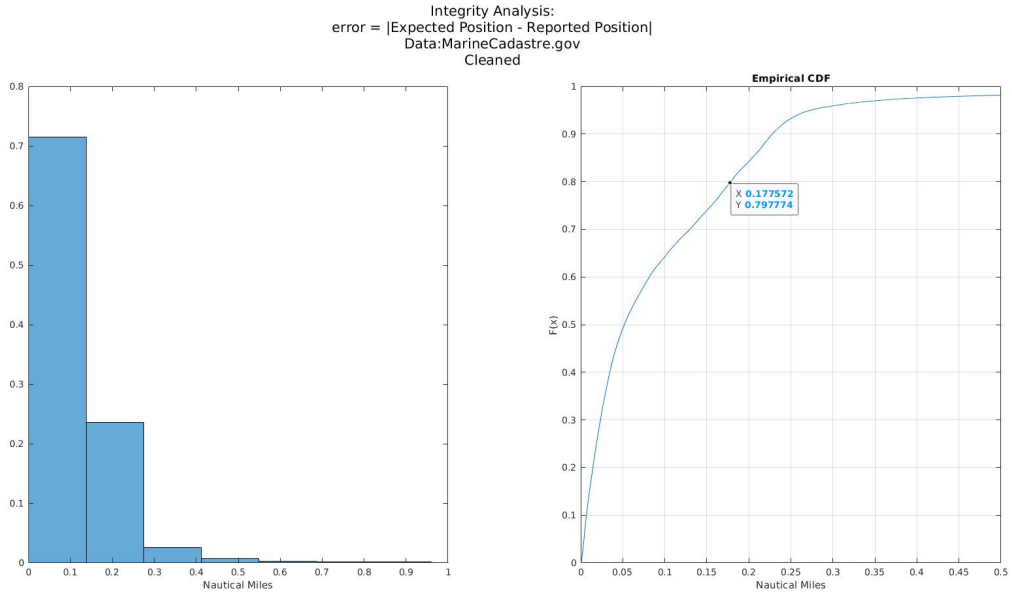


Figure A.1. MarineCadastre Data

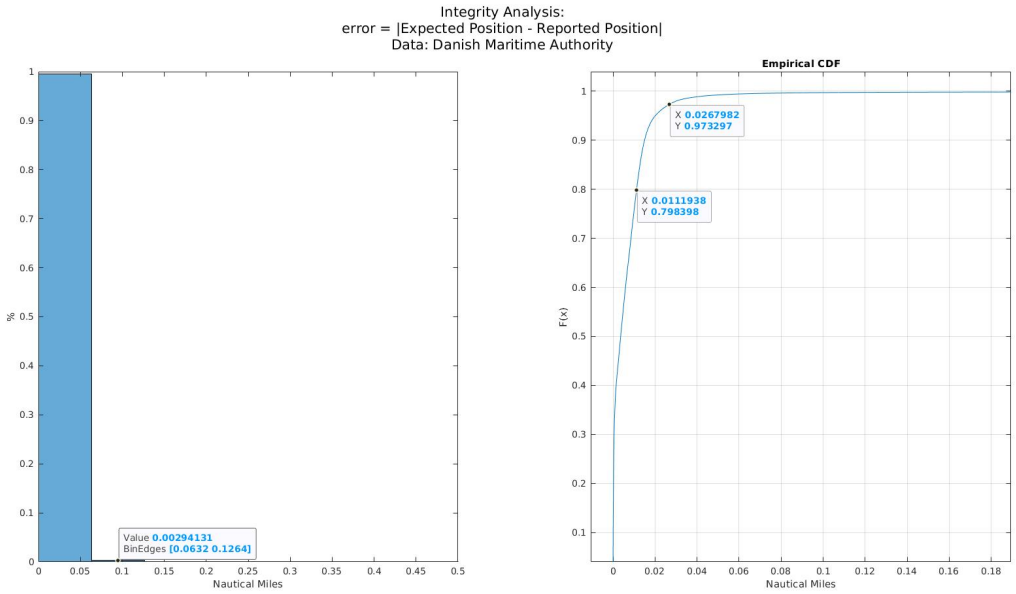


Figure A.2. Danish Maritime Authority Data

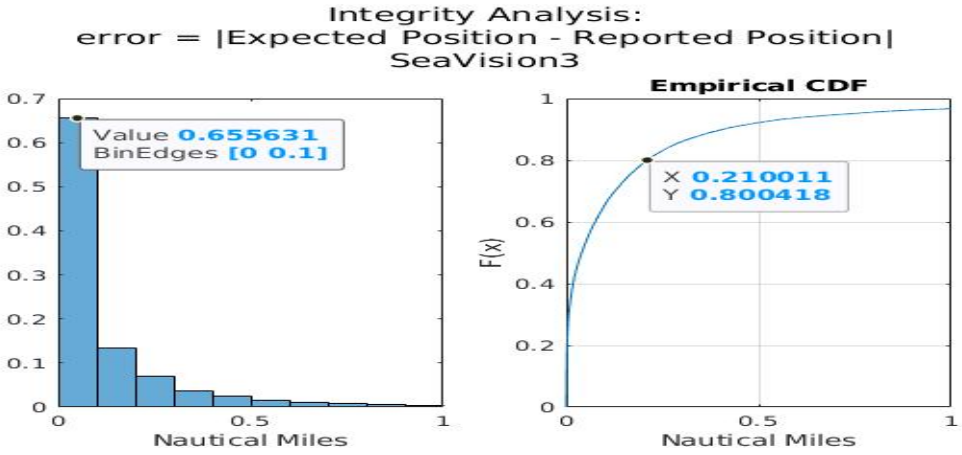


Figure A.3. SeaVision Data

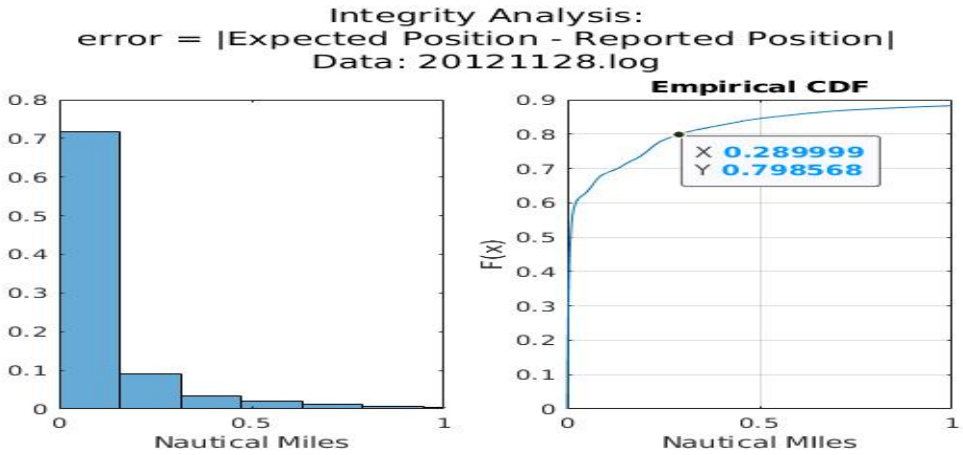


Figure A.4. 20121128.log

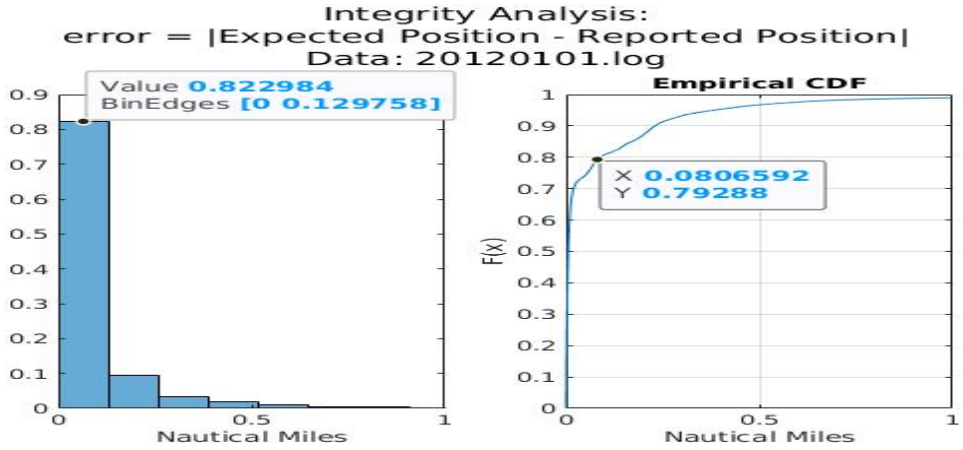


Figure A.5. 20120101.log

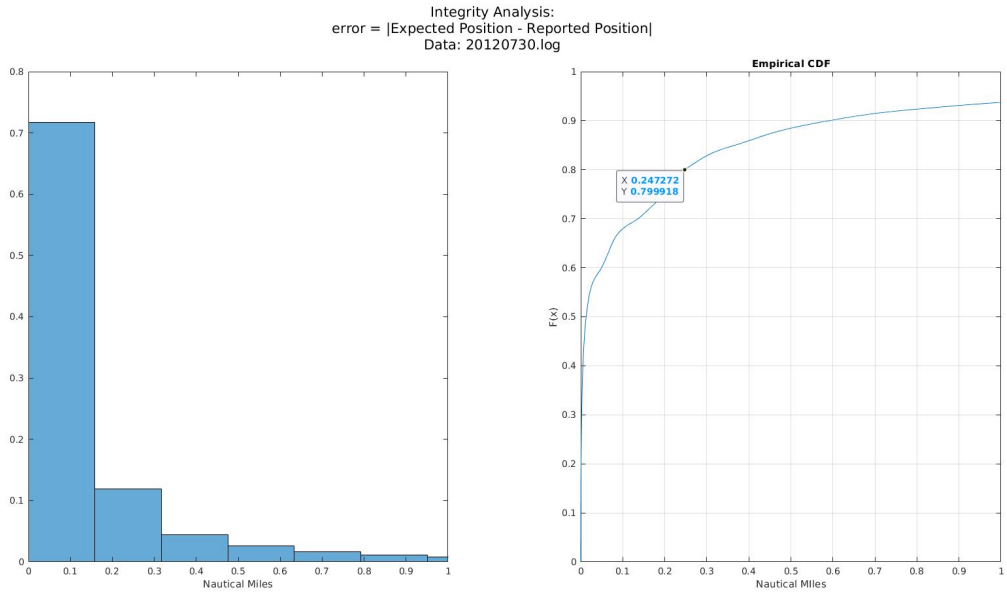


Figure A.6. 20120730.log

APPENDIX: B. Results for Empirical Assessment of Accuracy

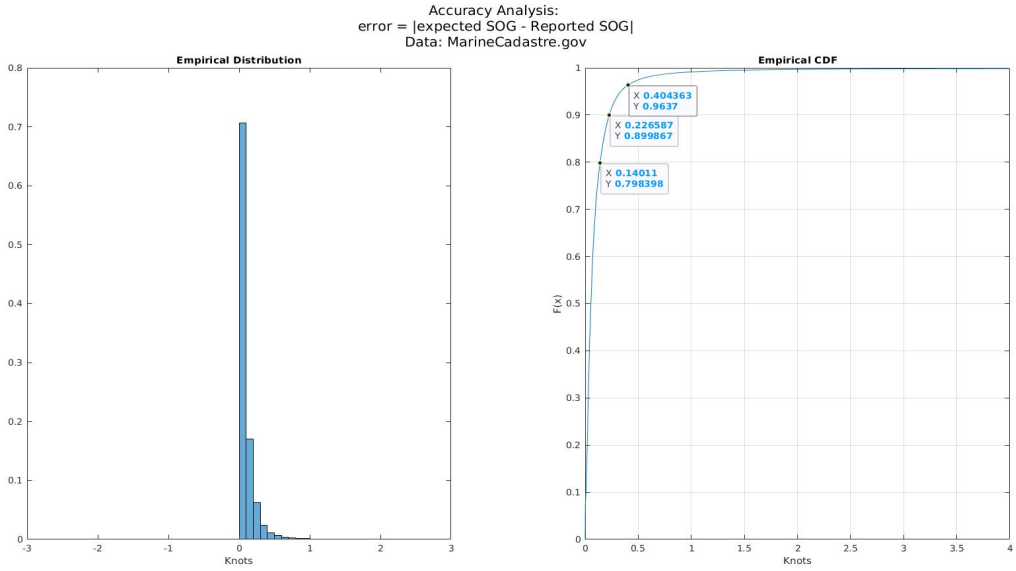


Figure A.1. MarineCadastre Data

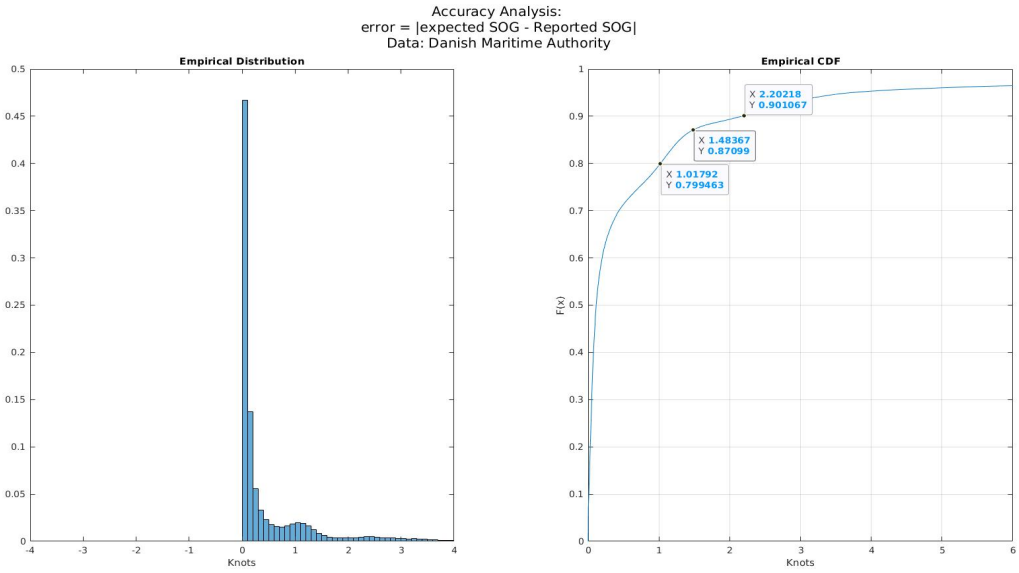


Figure A.2. Danish Maritime Authority Data

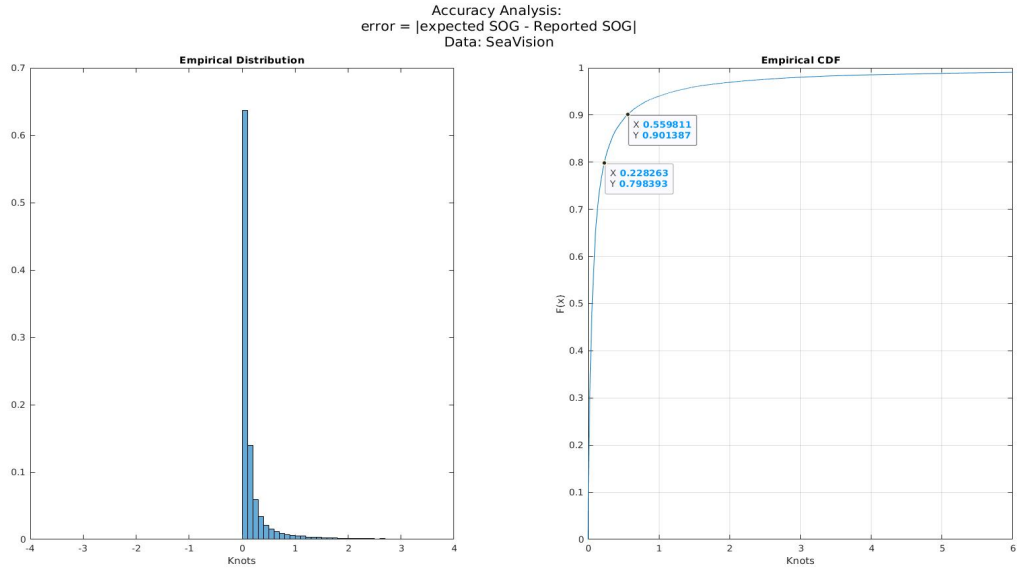


Figure A.3. SeaVision Data

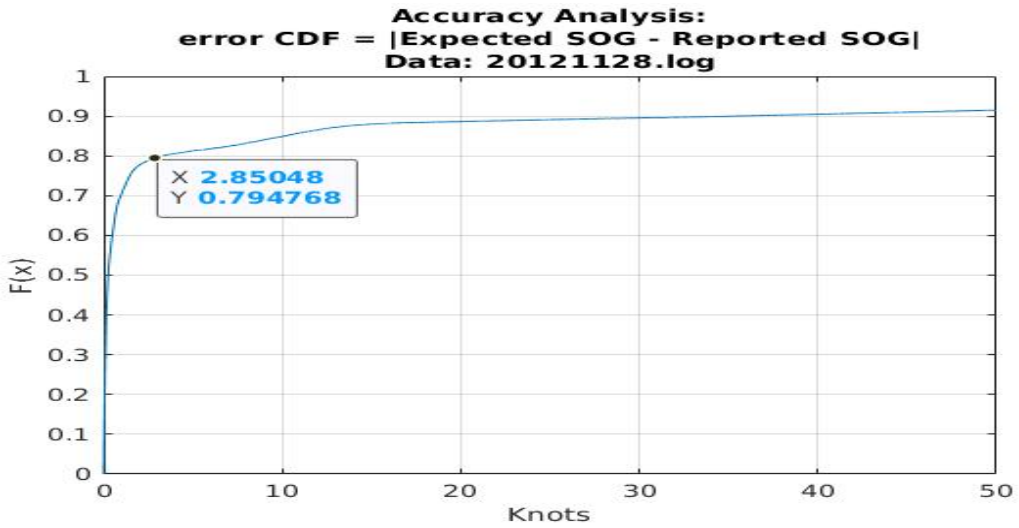


Figure A.4. 20121128.log

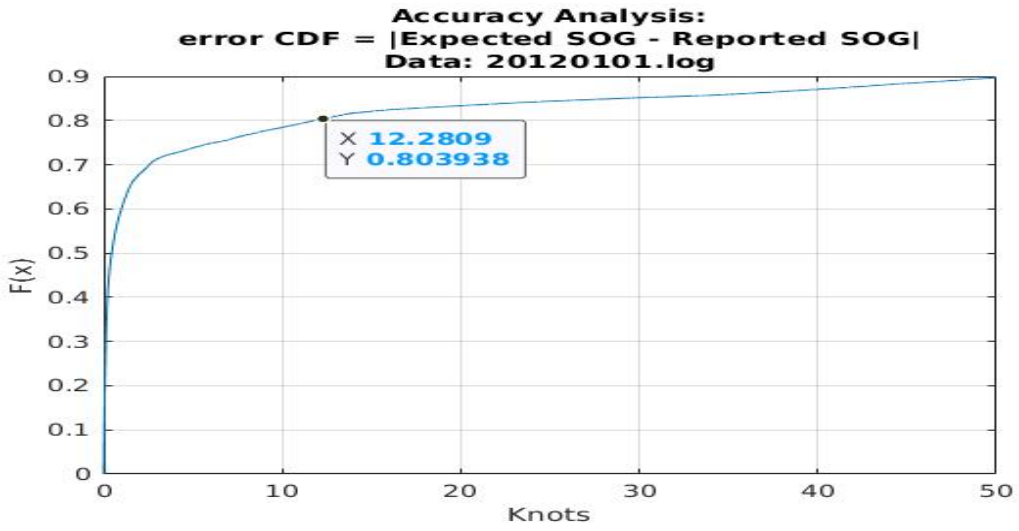


Figure A.5. 20120101.log

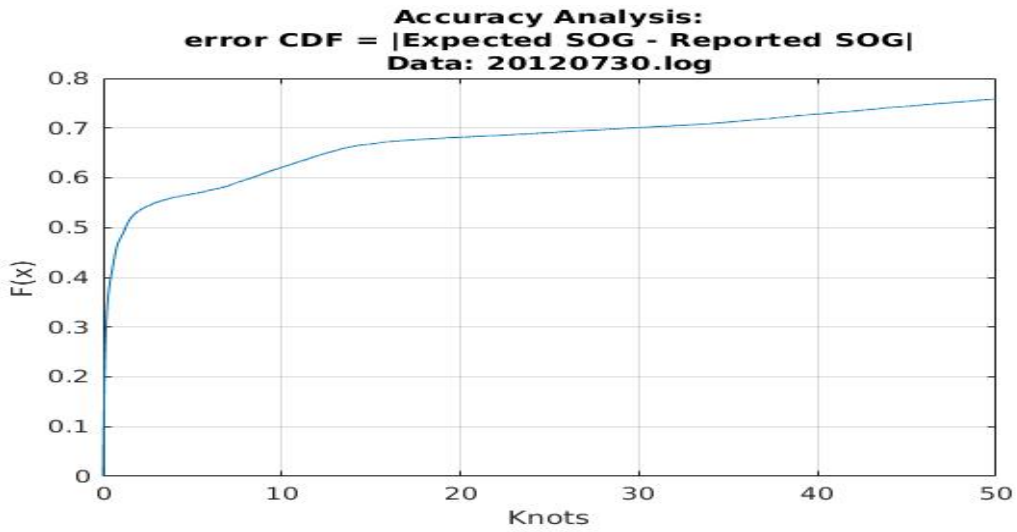


Figure A.6. 20120730.log

APPENDIX: C. Security System Python Code

```
# Jorge Vasquez
# 23SEPT2021
# ais_security.py
# Receives NMEA Sentences via UDP. If the AIS message type is 1,2,3,
    ↪  or 18, they are filtered through an allowlist,
# the quality of the message is assessed, and the contents in the
    ↪ message are altered in an attempt to prevent strings
# of code from being injected into the voyage network through the
    ↪ AIS receiver. The altered message is then coded into
# NMEA-0183 protocol and sent via UDP.

import socket
from pyais import decode_msg
import re
from ais_security_functions import allowlist
from ais_security_functions import alter_message
from ais_security_functions import quality_control
from encoder import NMEAencoder
from ais_udp_client import send_udp
from collections import deque
localIP = "127.0.0.1"
localPort = 55555
bufferSize = 1024

stack = deque(maxlen=1000)

# create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.
    ↪ SOCK_DGRAM)

# Bind to address amd ip
```

```
UDPServerSocket.bind((localIP, localPort))
print("UDP Server up and listening")

# MAIN LOOP: listen for UDP packets containing NMEA-0183 encoded AIS
    ↪ messages and filter types 1,2,3, and 18.
# If different type, forward them via UDP
while (True):

    try:
        # Initiate values
        allowed_sentence = 0
        allowed_sog = 0
        allowed_rot = 0
        allowed_coor = 0
        pass_quality = 0
        pass_integrity = 0
        pass_accuracy = 0
        first_occur = 0

        # RECEIVE: receive NMEA through UDP socket
        bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
        # start_time = time.time_ns() # start processing timer as
            ↪ soon as we receive
        message = bytesAddressPair[0]
        address = bytesAddressPair[1]
        end = [m.start() for m in re.finditer(',', format(message))
            ↪ ][5]
        message = message[0:end + 3]
        clientMsg = "\nMessage from Client {} ".format(message)
        clientIP = "Client IP Address:{}".format(address)

        # DECODE: decode NMEA message and make a copy to be altered
            ↪ later
```

```
content = decode_msg(message)
altered_content = decode_msg(message)

print(clientMsg)
print(f'Original Contents: {content}')

if content['type'] in [1, 2, 3, 18]:
    # ALLOW-LISTING: Filter the contents from the NMEA
    ↪ sentence through the allow list
    msg = allowlist(content)
    [allowed_sentence, allowed_sog, allowed_rot, allowed_coor]
    ↪ = msg.allow() # 1 for allow, 0 for deny

    # QUALITY CONTROL
    stack.appendleft(content)
    quality = quality_control(stack)
    [pass_quality, pass_integrity, pass_accuracy, first_occur]
    ↪ = quality.get_quality()

    # ALTER + ENCODE + SEND IF PASSED: Encode altered contents
    ↪ to new NMEA sentence and forward via UDP
    if allowed_sentence + pass_quality == 2:
        # ALTER NMEA: Alter contents of message
        msg2 = alter_message(altered_content)
        altered_content = msg2.alter()
        msg3 = NMEAencoder(altered_content, format(message))
        altered_nmea = msg3.encode()
        send_udp(altered_nmea)

    # PRINT FOR TESTING PURPOSES ONLY:
    if allowed_sentence + pass_quality == 2:
        print('PASS ALL')
        print(f'Altered Contents: {altered_content}')
```

```
            print(f'Altered NMEA Sentence: {altered_nmea}\n')
        elif allowed_sentence == 0 and pass_quality == 0:
            print('Denied by quality control filter and allow list'
                ↪ )
        elif allowed_sentence == 1 and pass_quality == 0:
            print('Denied by quality control filter only')
        elif allowed_sentence == 0 and pass_quality == 1:
            print('Denied by allow-list only')

    # Forward all other types of messages without filtering
    else:
        message = str(message)
        print(message)
        send_udp(message)



except Exception as ex:
    print(ex)
    print('\n')
```

```
# Jorge Vasquez
# 23SEPT2021
# ais_security_function.py
# Contains different classes that implement security functions to
   ↪ protect Automatic-Identification-System from malicious
# messages. Only works for message types 1,2,3, and 18.
# class allowlist verifies sog, rot, and coordinates for types 1,2,
   ↪ and 3 and only coordinates for type 18
# class quality_control assesses the integrity of messages
# class alter_message alters the fields in the AIS message in an
   ↪ attempt to alter strings of code, if present
```

```
import random
import numpy as np
import geopy.distance

# Coordinates allow-listing needs to be implemented. The vessel's
   ↪ latitude and longitude need to be updated
# often. This is a place holder for these values.
const_lat = 0
const_lon = 0


# Class of functions to verify AIVDM messages types 1,2 and 3
class allowlist(object):

   def __init__(self, fields):
      self.type = fields['type']
      self.SOG = fields['speed']
      self.lon = fields['lon']
      self.lat = fields['lat']
      if self.type in [1, 2, 3]:
         self.Nav_Status = int(fields['status'])
         self.ROT = fields['turn']

   # verify Speed-Over-Ground is in allow list
   def verify_sog(self):
      if self.Nav_Status == 0 and self.SOG <= 20:
         allow = 1
      elif self.Nav_Status == 1 and self.SOG <= 1:
         allow = 1
      elif self.Nav_Status == 2 and (self.SOG <= 15 or self.SOG in
         ↪ range(6.9, 15.1)):
         allow = 1
```

```
    elif self.Nav_Status == 3 and self.SOG <= 11:
        allow = 1
    elif self.Nav_Status == 4 and self.SOG <= 15:
        allow = 1
    elif self.Nav_Status == 5 and self.SOG <= 1:
        allow = 1
    elif self.Nav_Status == 6 and self.SOG <= 1:
        allow = 1
    elif self.Nav_Status == 7 and self.SOG <= 13:
        allow = 1
    elif self.Nav_Status == 8 and self.SOG <= 18:
        allow = 1
    elif self.Nav_Status == 9 and (self.SOG <= 14 or self.SOG in
        ↪ range(26, 38)):
        allow = 1
    elif self.Nav_Status == 10 and self.SOG <= 15:
        allow = 1
    elif self.Nav_Status == 11 and self.SOG <= 16:
        allow = 1
    elif self.Nav_Status == 12 and self.SOG <= 16:
        allow = 1
    elif self.Nav_Status == 13 and self.SOG <= 12:
        allow = 1
    elif self.Nav_Status == 14 and self.SOG <= 9.5:
        allow = 1
    elif self.Nav_Status == 15 and self.SOG <= 14:
        allow = 1
    else:
        allow = 0
        print('Failed␣SOG␣ALLOW')
    return allow

# verify Rate-of-Turn is in allow list
```

```
def verify_rot(self):
    if self.Nav_Status == 0 and self.ROT in range(-13, 15):
        allow = 1
    elif self.Nav_Status == 1 and self.ROT in range(-13, 15):
        allow = 1
    elif self.Nav_Status == 2 and self.ROT in range(-13, 15):
        allow = 1
    elif self.Nav_Status == 3 and self.ROT in range(-16, 21):
        allow = 1
    elif self.Nav_Status == 4 and self.ROT in range(-13, 15):
        allow = 1
    elif self.Nav_Status == 5 and self.ROT in range(-9, 11):
        allow = 1
    elif self.Nav_Status == 6 and self.ROT in range(-1, 1):
        allow = 1
    elif self.Nav_Status == 7 and self.ROT in range(-1, 1):
        allow = 1
    elif self.Nav_Status == 8 and self.ROT in range(-14, 15):
        allow = 1
    elif self.Nav_Status == 9 and self.ROT in range(-13, 14):
        allow = 1
    elif self.Nav_Status == 10 and self.ROT in range(-6, 6):
        allow = 1
    elif self.Nav_Status == 11 and self.ROT in range(-1, 1):
        allow = 1
    elif self.Nav_Status == 12 and self.ROT in range(-1, 1):
        allow = 1
    elif self.Nav_Status == 13 and self.ROT in range(-1, 1):
        allow = 1
    elif self.Nav_Status == 14 and self.ROT in range(-1, 1):
        allow = 1
    elif self.Nav_Status == 15 and self.ROT in range(-3, 3):
        allow = 1
```

```
    elif self.ROT == -128 or self.ROT == -127 or self.ROT == 127:
        allow = 1
    else:
        allow = 0
        print('Failed ROT Allow')
    return allow

# verify coordinates are in vessels neighborhood
def verify_coordinates(self):
    max_allowed = 360
    lat_diff = abs(const_lat - self.lat)
    lon_diff = abs(const_lon - self.lon)
    if lat_diff < max_allowed and lon_diff < max_allowed:
        allow = 1
    else:
        allow = 0
        print('Failed coord allow')
    return allow

# calls all functions in allow list class for ultimate
    ↪ verification of messages
def allow(self):
    allowed = 0
    sog_allow = 0
    rot_allow = 0
    coor_allow = self.verify_coordinates()
    if self.type in [1, 2, 3]:
        sog_allow = self.verify_sog()
        rot_allow = self.verify_rot()
        if sog_allow + rot_allow + coor_allow == 3:
            allowed = 1
    elif self.type == 18:
        if coor_allow == 1:
```

```
            allowed = 1
            sog_allow = 1
            rot_allow = 1
        return allowed, sog_allow, rot_allow, coor_allow

# Change the contents of the message without disrupting the
    ↪ functionality of AIS.
# Changes are small enough to not cause problems but it changes some
    ↪  of the characters
# in the NMEA sentence
class alter_message(object):

    def __init__(self, message_contents):
        self.type = message_contents['type']
        self.sog = message_contents['speed']
        self.course = message_contents['course']
        self.lon = message_contents['lon']
        self.lat = message_contents['lat']
        self.head = message_contents['heading']
        if self.type in [1, 2, 3]:
            self.rot = message_contents['turn']
        self.message_contents = message_contents

    # change the course value by a random value between 0.1 and 1
    def alter_course(self):
        if self.course < 359:
            self.course = round(self.course + round(random.uniform
                ↪ (0.1,1),1), 1)

    def alter_heading(self):
        if self.head < 359 and self.head > 1:
            self.head = self.head + round(random.uniform(-1, 1))
```

```
# change the sog value by a random value between 0.1 and 1
def alter_sog(self):
    self.sog = round(self.sog + round(random.uniform(0.1, 1), 1),
        ↪  1)

# 50 meters ~ 0.03nm
# .03nm ~ 0.0005 degrees
# alter lat and lon by 50m
def alter_lat(self):
    self.lat = self.lat + 0.0005

def alter_lon(self):
    self.lon = self.lon + 0.0005

# randomly change by -1, 0 or 1
def alter_rot(self):
    if self.rot != -128 and self.rot != -127 and self.rot != 127:
        self.rot = self.rot + random.randint(-1,1)

def alter(self):
    self.alter_course()
    self.alter_sog()
    self.alter_lat()
    self.alter_lon()
    self.alter_heading()
    self.message_contents['speed'] = self.sog
    self.message_contents['course'] = self.course
    self.message_contents['lat'] = self.lat
    self.message_contents['lon'] = self.lon
    self.message_contents['heading'] = self.head
    if self.type in [1, 2, 3]:
        self.alter_rot()
        self.message_contents['turn'] = self.rot
```

```
        return self.message_contents

# Calculates the integrity of a message by comparing the expected
    ↪ coordinates from message to message to the
# coordinates given within the message. If the integrity of the
    ↪ message is high quality, the message is marked
# as passing
class quality_control(object):

    def __init__(self, stck):
        self.stck = stck
        self.type = stck[0]['type']
        self.SOG = stck[0]['speed']
        self.lon = stck[0]['lon']
        self.lat = stck[0]['lat']
        self.time = stck[0]['second']

    # Search for previous messages from the same vessel. Returns
        ↪ first_occurence = 1 if no previous messages found.
    # If previous message is found then return previous parameters
        ↪ and first_occurence = 0.
    def search_stack(self):
        first_occurence = 1
        for i in range(1, len(self.stck)):
            if self.stck[0]['mmsi'] == self.stck[i]['mmsi']:
                index = i
                first_occurence = 0
                break
        if first_occurence == 0:
            prev_sog = self.stck[index]['speed']
            prev_lat = self.stck[index]['lat']
            prev_lon = self.stck[index]['lon']
            prev_time = self.stck[index]['second']
```

```
            prev_cog = self.stck[index]['course']
            return first_occurence, prev_sog, prev_lat, prev_lon,
                ↪ prev_time, prev_cog
        else:
            return first_occurence, 0, 0, 0, 0, 0

    # Calculate: expected position-reported position. If the distance
        ↪  between these positions is greater than the
    # allowed_error then integrity = 0 (failing quality dimension
        ↪ integrity)
    def integrity(self, first_occurence, prev_sog, prev_lat, prev_lon
        ↪ , prev_time, prev_cog):
        integrity = 1
        allowed_error = 0.19 # nautical miles
        if first_occurence == 0:
            dt = abs(self.time - prev_time) # hrs
            #print(f'dt : {dt}\n')
            if dt > 30:
                dt = abs(dt - 60)
            expected_lat = prev_lat + prev_sog*dt/3600 * np.cos(np.
                ↪ radians(prev_cog)) / 60.0405
            expected_lon = prev_lon + prev_sog*dt/3600 * np.sin(np.
                ↪ radians(prev_cog)) / 60.0405
            expected_coordinates = (expected_lat, expected_lon)
            reported_coordinates = (self.lat, self.lon)
            distance = geopy.distance.distance(expected_coordinates,
                ↪ reported_coordinates).nautical
            print(f'Integrity␣Error:␣{distance}␣NM')
            if distance >= allowed_error:
                integrity = 0
        return integrity

    # sog and vessel type(if type static position report)
```

```
def accuracy(self, first_occurence, prev_sog, prev_lat, prev_lon,
    ↪  prev_time):
    accuracy_passed = 1
    allowed_sog_error = 12.28
    if first_occurence == 0:
        dt = abs(self.time - prev_time)
        if self.time < prev_time:
            dt = abs(dt-60)
        try:
            prev_coordinates = (prev_lat, prev_lon)
            current_coordinates = (self.lat, self.lon)
            dist = geopy.distance.distance(prev_coordinates,
                ↪ current_coordinates).nautical
            expected_sog = dist / dt * 3600
            sog_error = abs(expected_sog - (self.SOG + prev_sog)/2)
            print(f'Accuracy␣Error:␣{sog_error}␣kn')
            if sog_error >= allowed_sog_error:
                accuracy_passed = 0
        except Exception as ex:
            print(ex)
            if self.lat == prev_lat and self.lon == prev_lon and dt
                ↪  == 0:
                accuracy_passed = 1
            else:
                accuracy_passed = 0
    return accuracy_passed

def get_quality(self):
    first_occurence, prev_sog, prev_lat, prev_lon, prev_time,
        ↪ prev_cog = self.search_stack()
    integrity_bool = self.integrity(first_occurence, prev_sog,
        ↪ prev_lat, prev_lon, prev_time, prev_cog)
    accuracy_bool = self.accuracy(first_occurence, prev_sog,
```

```
            ↪ prev_lat, prev_lon, prev_time)
        good_quality = 1
        if integrity_bool == 0 or accuracy_bool == 0:
            good_quality = 0
        return good_quality, integrity_bool, accuracy_bool,
            ↪ first_occurence
```

```
# Jorge Vasquez
# 23SEPT2021
# ais_udp_client.py
# UDP server called as a function to send NMEA messages over udp
import socket


def send_udp(nmea_message):

    msgFromClient = nmea_message
    bytesToSend = str.encode(msgFromClient)
    serverAddressPort = ("127.0.0.1", 55545)
    bufferSize = 1024

    # Create a UDP socket at client side
    UDPClientSocket = socket.socket(family=socket.AF_INET, type=
        ↪ socket.SOCK_DGRAM)

    # Send to server using created UDP socket
    UDPClientSocket.sendto(bytesToSend, serverAddressPort)
```

```
# Jorge Vasquez
# 23SEPT2021
# encoder.py
# Encode contents of a message type 1,2,3, and 18 AIS message to
    ↪ NMEA sentence
```

```
# First argument is a dictionary of the fields and values in the
    ↪ message
# SAMPLE INPUT 1: msg_content = {'type': 1, 'repeat': 0, 'mmsi':
    ↪ '432712000',\
# 'status': <NavigationStatus.UnderWayUsingEngine: 0>, 'turn': 0, '
    ↪ speed': 15.9, 'accuracy': 0, 'lon': -139.545295,\
# 'lat': 45.51727833333333, 'course': 72.10000000000001,\'heading':
    ↪ 74, 'second': 0,\
# 'maneuver': <ManeuverIndicator.NotAvailable: 0>, 'raim': 0, 'radio
    ↪ ': 16384}
# SAMPLE INPUT 18: msg_content = {'type': 18, 'repeat': 0, 'mmsi':
    ↪ '338149219', 'speed': 10.0, 'accuracy': 1,\
# 'lon': -121.89218, 'lat': 36.6, 'course': 360.0, 'heading': 360, '
    ↪ second': 30, 'regional': 0, 'cs': 1, 'display': 0,\
# 'dsc': 1, 'band': 1, 'msg22': 1, 'assigned': 0, 'raim': 1, 'radio
    ↪ ': 917510}

# Second argument is the received NMEA message string. NMEA message
    ↪ has to be a string starting with b
# Sample Input: b'!AIVDM,1,1,,A,B52NwHh0I=lPFJ5?5D3Q2l?5oP06,0*7A'

import re

class NMEAencoder(object):
    def __init__(self, msg_content, nmea_message):
        self.msg_content = msg_content
        self.nmea_message = nmea_message
        self.type = msg_content['type']

# converts the contents of the message to a string of bits
    def content2bin_type123(self):
        s1 = bin(self.msg_content['type']).replace('0b', '').zfill(6)
        s2 = bin(self.msg_content['repeat']).replace('0b','').zfill
```

```
        ↪ (2)
    s3 = bin(int(self.msg_content['mmsi'])).replace('0b','').
        ↪ zfill(30)
    s4 = bin(int(self.msg_content['status'])).replace('0b','').
        ↪ zfill(4)
    s5 = bin(self.msg_content['turn'] & 0b11111111).replace('0b',
        ↪ '').zfill(8)
    s6 = bin(int(self.msg_content['speed']*10)).replace('0b','').
        ↪ zfill(10)
    s7 = bin(self.msg_content['accuracy']).replace('0b','').zfill
        ↪ (1)
    s8 = bin(int(self.msg_content['lon']*600000) & 0
        ↪ b111111111111111111111111111).replace('0b','').zfill
        ↪ (28)
    s9 = bin(int(self.msg_content['lat']*600000) & 0
        ↪ b111111111111111111111111111).replace('0b','').zfill
        ↪ (27)
    s10 = bin(int(self.msg_content['course']*10)).replace('0b',''
        ↪ ).zfill(12)
    s11 = bin(self.msg_content['heading']).replace('0b','').zfill
        ↪ (9)
    s12 = bin(self.msg_content['second']).replace('0b','').zfill
        ↪ (6)
    s13 = bin(self.msg_content['maneuver']).replace('0b','').
        ↪ zfill(2)
    s14 = '000'
    s15 = bin(self.msg_content['raim']).replace('0b','').zfill(1)
    s16 = bin(self.msg_content['radio']).replace('0b','').zfill
        ↪ (19)
    s = s1+s2+s3+s4+s5+s6+s7+s8+s9+s10+s11+s12+s13+s14+s15+s16
    return s

def content2bin_type18(self):
```

```
print(self.msg_content['type'])
s1 = bin(self.msg_content['type']).replace('0b', '').zfill(6)
s2 = bin(self.msg_content['repeat']).replace('0b','').zfill
    ↪ (2)
s3 = bin(int(self.msg_content['mmsi'])).replace('0b','').
    ↪ zfill(30)
s4 = '00000000'
s5 = bin(int(self.msg_content['speed']*10)).replace('0b','').
    ↪ zfill(10)
s6 = bin(self.msg_content['accuracy']).replace('0b','').zfill
    ↪ (1)
s7 = bin(int(self.msg_content['lon']*600000) & 0
    ↪ b11111111111111111111111111).replace('0b','').zfill
    ↪ (28)
s8 = bin(int(self.msg_content['lat']*600000) & 0
    ↪ b11111111111111111111111111).replace('0b','').zfill
    ↪ (27)
s9 = bin(int(self.msg_content['course']*10)).replace('0b','')
    ↪ .zfill(12)
s10 = bin(self.msg_content['heading']).replace('0b','').zfill
    ↪ (9)
s11 = bin(self.msg_content['second']).replace('0b','').zfill
    ↪ (6)
s12 = bin(self.msg_content['regional']).replace('0b','').
    ↪ zfill(2)
s13 = bin(self.msg_content['cs']).replace('0b','').zfill(1)
s14 = bin(self.msg_content['display']).replace('0b','').zfill
    ↪ (1)
s15 = bin(self.msg_content['dsc']).replace('0b','').zfill(1)
s16 = bin(self.msg_content['band']).replace('0b','').zfill(1)
s17 = bin(self.msg_content['msg22']).replace('0b','').zfill
    ↪ (1)
s18 = bin(self.msg_content['assigned']).replace('0b','').
```

```
        ↪ zfill(1)
    s19 = bin(self.msg_content['raim']).replace('0b','').zfill(1)
    s20 = bin(self.msg_content['radio']).replace('0b','').zfill
        ↪ (20)
    s = s1+s2+s3+s4+s5+s6+s7+s8+s9+s10+s11+s12+s13+s14+s15+s16+
        ↪ s17+s18+s19+s20
    return s

def bit2ascii(self,s):
    char=['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',\
    ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E',\
    'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q',\
    'R', 'S', 'T', 'U', 'V', 'W', "`", 'a', 'b', 'c', 'd', 'e',\
    'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',\
    'r', 's', 't', 'u', 'v', 'w']

    b = ['000000', '000001', '000010', '000011', '000100', '
        ↪ 000101',\
         '000110', '000111', '001000', '001001', '001010', '001011
            ↪ ',\
         '001100', '001101', '001110', '001111', '010000', '010001
            ↪ ',\
         '010010', '010011', '010100', '010101', '010110', '010111
            ↪ ',\
         '011000', '011001', '011010', '011011', '011100', '011101
            ↪ ',\
         '011110', '011111', '100000', '100001', '100010', '100011
            ↪ ',\
         '100100', '100101', '100110', '100111', '101000', '101001
            ↪ ',\
         '101010', '101011', '101100', '101101', '101110', '101111
            ↪ ',\
         '110000', '110001', '110010', '110011', '110100', '110101
```

```
            ↪ ',\
        '110110', '110111', '111000', '111001', '111010', '111011
            ↪ ',\
        '111100', '111101', '111110', '111111']


    # 168 bits
    NMEA = ''
    for n in range(0,162,6):
        bits = s[n:n+6]
        character = char[b.index(bits)]
        #character = map(ascii_to_6bit.get,bits)
        NMEA = NMEA + character
    begin = re.search('!AIVDM', self.nmea_message).start()
    NMEA = self.nmea_message[begin:begin+14]+NMEA+self.
        ↪ nmea_message[begin+41:begin+44]
    return NMEA

# https://code.activestate.com/recipes/576789-nmea-sentence-
    ↪ checksum/
def checksum(self, nmea_line):
    try:
        nothing, nmeadata = re.split('!', nmea_line)
    except:
        nmeadata = nmea_line
    calc_cksum = 0
    for s in nmeadata:
        calc_cksum ^= ord(s)
    return str(hex(calc_cksum).replace('0x', '').zfill(2)).upper
        ↪ ()


def encode(self):
```

```
        if self.type in [1, 2, 3]:
            s = self.content2bin_type123()
        elif self.type == 18:
            s = self.content2bin_type18()
        NMEA_sentence = self.bit2ascii(s)
        chksum = self.checksum(NMEA_sentence)
        NMEA_sentence = NMEA_sentence + '*' + chksum
        return NMEA_sentence
```

APPENDIX: D. random_NMEA_generator.py

```
# Jorge Vasquez
# September 2021
# random_NMEA_generator.py
# Generates random AIS NMEA-0183 sentences of types 1,2, and 3
import random
import numpy as np
import numpy.random
from encoder import NMEAencoder

numpy.random.seed(4)


def random_value_generator():
    mtype = np.random.random_integers(1,3)
    repeat = np.random.random_integers(0,3)
    mmsi = np.random.random_integers(100000000, 999999999)
    status = np.random.random_integers(0, 15)
    turn = np.random.random_integers(-128, 127)
    sog = np.random.random_integers(0,1023) / 10
    accuracy = np.random.random_integers(0,1)
    min = -1*2**(28-1)
    max = 2**(28-1)-1
    lon = np.random.random_integers(min,max) / 600000
    lat = np.random.random_integers(-1*2**(27-1),2**(27-1)-1) /
        ↪ 600000
    cog = np.random.random_integers(0,4095) / 10
    heading = np.random.random_integers(0, 511)
    time = np.random.random_integers(0, 63)
    maneuver = np.random.random_integers(0,3)
    spare = np.random.random_integers(0,7)
    raim = np.random.random_integers(0,1)
```

```
    radio = np.random.random_integers(0, 524287)
    content = {'type':mtype, 'repeat':repeat, 'mmsi':mmsi, 'status':
        ↪ status, 'turn': turn, 'speed':sog,\
              'accuracy':accuracy, 'lon':lon, 'lat':lat, 'course':cog
                 ↪ , 'heading': heading, 'second': time,\
              'maneuver': maneuver, 'raim':raim, 'radio':radio}
    print(content)
    placeholder = '!AIVDM,1,1,,A,B52NwHh0I=lPFJ5?5D3Q2l?5oP06,0*7A'
    msg = NMEAencoder(content, placeholder)
    NMEA_sentence = msg.encode()
    print(NMEA_sentence)
    return NMEA_sentence

nSamples = 60000
with open('random_generated_ais_v3.txt', 'w') as f:
    for i in range(1, nSamples):
        sentence = random_value_generator()
        f.write(sentence)
        f.write('\n')
```

List of References

[1] M. Chambers and L. Mindy, "Maritime trade and transportation by the numbers," *Bureau of Transportation Statistics*, Mar. 2013[Online]. Available: https://www.bts.gov/archive/publications/by_the_numbers/maritime_trade_and_transportation/index

[2] "International Maritime Organization," *Encyclopedia Britannica*. Available: https://www.britannica.com/topic/International-Maritime-Organization

[3] SOLAS Chapter V, Regulation 19.2, Carriage Requirements for Shipborne Navigational Systems and Equipment. [Online]. Available: https://www.navcen.uscg.gov/?pageName=AISRequirementsRev

[4] Automatic Identification System. 33 CFR § 164.46.

[5] "Automatic Identification System Overview," United States Coast Guard Navigation Center. Available: https://www.navcen.uscg.gov/?pageName=AISmain

[6] A. C. Baierwalter, "Reverse engineering and analysis of a commercial radar protocol to manipulate the human-machine interface," M.S. Thesis, Dept. of Electrical and Computer Engineering, NPS, Monterey, CA, USA, 2021.

[7] "AIS transponders," accessed Mar. 10, 2021. [Online]. Available: https://www.imo.org/en/OurWork/Safety/Pages/AIS.aspx

[8] A. King, "Seven things you should know about AIS," Jan. 2018, section: AIS – essential knowledge. Available: https://www.marinetraffic.com/blog/seven-things-know-ais/

[9] "Recommendation itu-r m.1371-5," International Telecommunication Union.

[10] *NMEA 0183 Interface Standard*, National Marine Electronics Association v3.01, 2012.

[11] J. M. Heaney, "Transmitting arbitrary data with ais," M.S. Thesis, Dept. of Electrical and Computer Engineering, NPS, Monterey, CA, USA, 2017.

[12] T. Morten Jagd Christensen, Thrane, and Ø. J. Rødseth, "Lightweight ethernet - a new standard for shipboard networks," pp. 31–32, Dec. 10, 2010. [Online]. Available: https://www.thedigitalship.com/component/jdownloads/send/8-2010/63-digital-ship-64-december-2010?option=com_jdownloads

[13] M. F. Rubenstein, "Cybersecurity of rf apertures to maritime voyage networks," M.S. Thesis, Dept. of Electrical and Computer Engineering, NPS, Monterey, CA, USA, 2016.

[14] S. A. Marion, "Maritime rf security analysis," M.S. Thesis, Dept. of Electrical and Computer Engineering, NPS, Monterey, CA, USA, 2017.

[15] D. H. Nguyen, "Hardening automatic identification systems: Providing integrity through an application of lightweight cryptographic techniques," M.S. Thesis, Dept. of Information Sciences, NPS, Monterey, CA, USA, 2020.

[16] I. Kontopoulos, G. Spiliopoulos, D. Zissis, K. Chatzikokolakis, and A. Artikis, "Countering real-time stream poisoning: An architecture for detecting vessel spoofing in streams of ais data," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 981–986.

[17] K. Kowalska and L. Peel, "Maritime anomaly detection using gaussian process active learning," in *2012 15th International Conference on Information Fusion*, 2012, pp. 1164–1171.

[18] C. Ray, R. Gallen, C. Iphar, A. Napoli, and A. Bouju, "Deais project: Detection of ais spoofing and resulting risks," in *OCEANS 2015 - Genova*, 2015, pp. 1–6.

[19] C. Iphar, A. Napoli, and C. Ray, "Detection of false ais messages for the improvement of maritime situational awareness," in *OCEANS 2015 - MTS/IEEE Washington*, 2015, pp. 1–7.

[20] Vessel Traffic Data. MarineCadastre.gov. [Online]. Available: https://marinecadastre.gov/ais/. Accessed Mar. 10, 2021.

[21] D. M. Authority, "Danish maritime authority - ais data." Available: https://www.dma.dk/SikkerhedTilSoes/Sejladsinformation/AIS/Sider/default.aspx

[22] U. S. D. of Transportation, "Seavision." Available: https://seavision.volpe.dot.gov/login

[23] "Blacklisting vs. whitelisting," *Consolidated Technologies, Inc.*, June 2021. [Online]. Available: https://consoltech.com/blog/blacklisting-vs-whitelisting/

[24] "What is the typical range of the AIS?" *MarineTraffic Help*. Available: https://help.marinetraffic.com/hc/en-us/articles/203990918--What-is-the-typical-range-of-the-AIS-

[25] jvasqu38, “ais_security,” https://github.com/NPS-CCW/ais_security, 2021.

[26] M0r13n, “pyais,” https://github.com/M0r13n/pyais, 2021.

[27] *socket — Low-level networking interface*, Python 3.10.0 documentation. Available: https://docs.python.org/3/library/socket.html

[28] transmitterdan, “Vdrplayer,” https://github.com/transmitterdan/VDRplayer, 2021.

[29] “Your Ultimate Guide to Fuzz Testing,” *ForAllSecure*. Available: https://forallsecure.com/resources/ultimate-guide-to-fuzz-testing

[30] P. Ņikiforovs, “htop explained,” peteris.rocks, Nov. 17. Available: https://peteris.rocks/blog/htop/

[31] Raspberry Pi 4. raspberrypi.com. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. Accessed Dec. 12, 2021.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California