



**CYBER SECURITY ANALYSIS AND
STRATEGY DEVELOPMENT FOR
SOFTWARE-DEFINED RADARS**

THESIS

Blake Edward Yerkes, Second Lieutenant, United States Air Force
AFIT-ENG-MS-17-M-083

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENG-MS-17-M-083

CYBER SECURITY ANALYSIS AND STRATEGY DEVELOPMENT FOR
SOFTWARE-DEFINED RADARS

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Masters of Science in Electrical Engineering

Blake Edward Yerkes, B.S.E.E.
Second Lieutenant, United States Air Force

23 March 2017

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENG-MS-17-M-083

CYBER SECURITY ANALYSIS AND STRATEGY DEVELOPMENT FOR
SOFTWARE-DEFINED RADARS

THESIS

Blake Edward Yerkes, B.S.E.E.
Second Lieutenant, United States Air Force

Committee Membership:

Lieutenant Colonel Mason Rice, PhD
Chair

Lieutenant Colonel John Pecarina, PhD
Member

Mr. Stephen Dunlap,
Member

Abstract

This research investigates a rudimentary software-defined radar. The research presents cyber attack and defense strategies on a test-bed architecture. The cyber attack and defense vectors are contextualized within the current state of software-defined radar systems by reviewing and summarizing various applications of software-defined radars currently in use. The system under test is open-source but representative of many software-defined radars found in current use. The operation of the system is described from the application level to the hardware operation. The system under test is found to be vulnerable to man-in-the-middle attacks. This attack vector is capable of modifying packet data to change configuration settings onboard the system front end and modifying the waveform data between the two nodes. Manipulation of the waveform data produces a predictable effect in the distance estimates produced by the radar. The results of the configuration command and waveform data manipulation are quantified and presented. The research presents strategies for securing the system under test. The research also presents tools for securing the system.

Table of Contents

	Page
Abstract	iv
List of Figures	viii
List of Tables	x
I. Introduction	1
1.1 Overview & Background	1
1.2 Problem Statement	2
1.3 Research Goals and Hypothesis	2
1.4 Approach	2
1.5 Assumptions & Limitations	3
1.6 Research Contributions	3
II. Background and Related Research	5
2.1 Use of Software-Defined Radars	5
2.1.1 Software-Defined Radar Research	5
2.1.2 Deployed and Developing Software-Defined Radar Systems	7
2.1.3 Summary of Architectures	9
2.2 System Description	10
2.2.1 Radar Operation	11
2.2.2 GNU Radio	12
2.2.3 Universal Software Radio Peripheral N210	16
2.2.4 Universal Hardware Driver	18
2.2.5 Gr-radar	21
2.2.6 Ettercap	30
III. Investigative Process and Developed Tools	33
3.1 Implemented Tools and Extensions	33
3.1.1 Host Side Software Tools	33
3.1.2 Man-In-The-Middle Tools	41
3.2 Implemented Attacks	47
3.2.1 Host Side Software Manipulation	48
3.2.2 Configuration Command Setting Manipulation	51
3.2.3 IQ Data Manipulation	52
3.3 Implemented Defenses	52
3.3.1 Delay Check	53
3.3.2 Software Modification Mitigation	54
3.4 Securing The Software-Defined Radar	54

	Page
3.4.1 Tools Required	55
3.4.2 Encryption	55
3.4.3 Symmetric Key Encryption	56
3.4.4 Public Key Infrastructure	56
3.4.5 Message Authentication Code	57
3.4.6 Digital Signature	57
IV. Research Methodology	59
4.1 Hardware Specifications	59
4.1.1 Computer Specifications	59
4.1.2 Analog Equipment	60
4.1.3 USRP N210 Specifications	60
4.1.4 Network Equipment	61
4.2 Software Specifications	62
4.2.1 Host Computer Specifications	62
4.2.2 Man-In-The-Middle Specifications	62
4.3 GNU Radio Companion Flow-Graph Parameters	63
4.3.1 Chosen Flow-Graph Parameters	64
4.3.2 Derived Flow-Graph Parameters	67
4.4 Attack Experiments	68
4.4.1 Host Side Software Manipulation	68
4.4.2 Waveform Sniffing	70
4.4.3 Experimental Setup For Configuration and IQ Sequence Tests	70
4.4.4 Sample Rate Configuration Command Setting Manipulation	73
4.4.5 IQ Data Manipulation	74
4.5 Defense Experiments	76
4.5.1 Software Modification Mitigation	76
4.5.2 Delay Check Intrusion Detection	76
V. Results and Analysis	78
5.1 Attack Experiments	78
5.1.1 Host Side Software Manipulation	78
5.1.2 Waveform Sniffing Results	80
5.1.3 Sample Rate Configuration Command Setting Manipulation Results	81
5.1.4 IQ Data Manipulation Results	84
5.2 Defense Experiments Results	87
5.2.1 Software Modification Mitigation Results	87
5.2.2 Delay Check Intrusion Detection Results	88
5.3 Summary	89

	Page
VI. Conclusion and Recommendations	91
6.1 Research Conclusion	91
6.1.1 Host Side Software	91
6.1.2 Hardware Interface	91
6.2 Current State	92
6.3 Research Contributions	93
6.4 Recommendations for Future Work	94
6.5 Concluding Thoughts	94
Bibliography	95

List of Figures

Figure	Page
1. Architecture of Generic USRP and Host Machine System	11
2. Simple GNU Radio Flow-Graph	13
3. XML Generation From Flow-Graph	14
4. Python File Generated From Flow-Graph	15
5. General USRP Architecture	16
6. USRP N210 Architecture	17
7. VITA 49 Radio Transport Fields	19
8. Data and Control Packet Format	20
9. General Estimation Cycle	22
10. Data Generation Algorithm	24
11. Echotimer Parameters	25
12. Transmission Process of IQ Data	26
13. Reception Process of IQ Data	27
14. ARP Cache Poisoning Redirecting Traffic	31
15. Estimator FSK Module	35
16. Calibration of Phase Inside Estimator-FSK	35
17. Estimator-FSK Inside GNU Radio Companion	36
18. Extended XML For Estimator-FSK Module	36
19. C++ Constructor For Estimator FSK	36
20. LUA Dissector Interpreting Packets Within Wireshark	38
21. Code For Dissecting Transmission Gain Packets	39
22. Modifying UHD to Examine Values	40

Figure		Page
23.	Ettercap Filter For Gain	43
24.	Packet Capture Used to Generate Filters	45
25.	Data Parsing Algorithm	46
26.	Ettercap Data Filter	47
27.	Wireshark Summary	49
28.	Reconstructed Spectrum on MITM Machine	50
29.	Timing Round Trip Time	53
30.	Modification of UHD Code	69
31.	Modification of Echotimer Source Code	69
32.	Antenna Placement	71
33.	Table Setup	72
34.	Measuring Tape Setup	72
35.	Wireshark Display of Unmodified Output For Gain Input of Zero	78
36.	Wireshark Display of Modified Output For Gain Input of Zero	79
37.	Accuracy of Reconstructed IQ Data	81
38.	Sample Rate Modification Results	82
39.	Effects of Introducing Phase Shift	85

List of Tables

Table		Page
1.	Summary of Hardware and Data Generation in Examined Radars	9
2.	Global Dual Continuous Wave Flow-Graph Parameters	23
3.	Port Number Allocations	42
4.	Host Computer Specifications	59
5.	Analog Equipment	60
6.	USRP N210 Specifications	61
7.	MITM Computer Specifications	62
8.	Host Computer Software Specifications	62
9.	Man-In-The-Middle Computer Software Specifications	63
10.	Global Dual Continuous Wave Flow-Graph Parameter Values	64
11.	Derived Dual Continuous Wave Flow-Graph Parameter Values	67
12.	Unmodified Payload Values from UHD Library	79
13.	Modified Payload Values from UHD Library	79
14.	Sample Rate Modification Summary: Pre-Attack	83
15.	Sample Rate Modification Summary: During Attack	83
16.	IQ Data Modification Summary: Pre-Attack	85
17.	IQ Data Modification Summary: During Attack	86
18.	IQ Data Modification Summary: During Attack	86
19.	Baseband Transmission Gain Measurements	87
20.	Variable Gain Amplifier Transmission Gain Measurements	88

CYBER SECURITY ANALYSIS AND STRATEGY DEVELOPMENT FOR SOFTWARE-DEFINED RADARS

I. Introduction

1.1 Overview & Background

Software-defined signal processing systems are an emerging technology used in research, industry, and defense applications. A software-defined system offers flexibility that a traditional monolithic systems cannot. However, this flexibility introduces a possible security weak point. These systems must be thought of as a cyber asset when they are deployed into real world systems. It is critically important that designers analyze and secure these new technologies against cyber and traditional attacks. An emerging application of software-defined technology is radar. Software-defined radars are the subject of academic research including medical [1], automotive [2], and weather monitoring applications [3]. Software-defined radars are currently used by NASA [4] for topographical mapping. Software-defined radars are also actively being developed for defense applications [5].

This research supports the Department of Defense Cyber Strategy: Goal III, defending US interests from disruptive cyber attacks [6]. Specifically, this research provides guidelines on assessing and securing software-defined radar architectures. The emergence of this new technology into operational roles requires that resources be devoted to assessing, securing, and maintaining this technology as a cyber asset.

1.2 Problem Statement

There is a lack of research that discusses the security of software-defined radars as cyber systems. Work exists quantifying and assessing the performance of various software-defined radar systems, but there is little mention of how to quantify and assess these systems for trusted use as a cyber asset.

1.3 Research Goals and Hypothesis

This work analyzes a test-bed software-defined radar system and presents cyber attacks and mitigation techniques on the test-bed architecture. The attacks and mitigation techniques serve as a guideline to assess the security of a software-defined radar system. The test-bed architecture is an unclassified system that consists of open source components. The lessons learned from a test-bed architecture can then be extrapolated to inform and build guidelines to secure software-defined radar systems. The goal of the research is to investigate software-defined radar architectures using a test-bed as an example. The hypothesis of this study is that the system architecture is vulnerable to cyber attack, specifically the communication channel between the two major components.

1.4 Approach

There are various software-defined radar architectures. Current deployed software-defined radars are often times a custom system designed for a specific application. The potential for software-defined radar resides in the reconfigurability inherent to the software-defined architecture. Therefore, this research focuses on an architecture that best exemplifies the reconfigurable nature of software-defined radar.

This research was conducted by analyzing an open source software-defined radar

architecture. The companies responsible for the system components did not design them to be used in secure applications. However, the system serves as an academic example of a software-defined radar for the purposes of this study and the architecture shares features with deployed systems. The core of the system consists of GNU Radio and the Universal Software Radio Peripheral N210 (USRP N210). These components combine to form a functional radar that is capable of producing distance and velocity estimates. The effectiveness of each attack or defense is analyzed by using the output of the radar estimation as data points.

1.5 Assumptions & Limitations

There are a variety of software-defined architectures unique to specific applications. This research will focus on an open source software-defined radar architecture. This allows the research to approach the architecture with no a priori knowledge of the system but provides a general audit of a software-defined radar. Once the system is defined, the cyber attack tools will be limited to readily available software. This limits the scope of the study to a reasonable depth without requiring the development of custom hardware.

1.6 Research Contributions

This research presents attack, defense, and analysis tools on a test-bed software-defined radar. This research identifies vulnerabilities in the communication channel between components in the system. The analysis revealed that the architecture is vulnerable to Man-in-the-Middle (MITM) attacks. This work enumerates these attacks, deploys them on the system, and measures their effect. This portion of the thesis was submitted and accepted to the International Conference on Cyber Warfare and Security [7]. This work also creates defense strategies for the software on the host

machine. The add-ons include a intrusion detection system inside the host software and a packet dissector plugin for Wireshark.

II. Background and Related Research

The chapter begins with a review of the literature surrounding software-defined radars. The review covers software-defined radars as they exist in research and deployed applications. The chapter then covers the background necessary to understand the specific components used in the system under test. This section also covers the operation of all hardware and software necessary to understand the operation of the system.

2.1 Use of Software-Defined Radars

Software-defined radars are characterized as systems that implement a radar functionality in software that was traditionally done in hardware. Examples of these functions include signal generation, filtering, modulation, and mixing. The inherent ability in software-defined systems to modify the software that implements signal processing creates a flexible framework that can be reconfigured for use in multiple applications such as radio, global positioning systems, and radar.

Software-defined radar architectures are possible due to the decreased cost of near real time computing. The lowered cost and increased capability of software-defined radars offers scientists and engineers a flexible system with which to develop, deploy, and monitor radio frequency systems in multiple bands, applications, and configurations. Software-defined radars are most commonly found in research applications, but are emerging as the architecture of choice in real world applications.

2.1.1 Software-Defined Radar Research.

This section covers a subset of the software-defined radars as they appear in research. Software-defined radar research covers a range of topics from medical imaging

to weather monitoring applications. The research applications in this section serve to provide examples of the more interesting software-defined radars.

The architecture of each software-defined radar varies, but there are commonalities between each system. The sections below review each of the research systems that demonstrate an implementation of a software-defined radar. The systems in this section use at least one off-the-shelf component in the system, and serve as a proof of concept design.

Medical Imaging.

A team has adapted a low cost software-defined radio to function as a radar and detect foreign objects inside human tissue [1]. The software-defined radar utilizes a BladeRF software-defined radio and GNU Radio [8] to create the low cost radar. The BladeRF hardware has various models, all available for under 2000 dollars. This price point positions this configuration as a prototype for a mobile medical device [1]. The authors of the work envision a device that could be supplied to field teams to detect foreign objects in patients without having to transport the patient to a traditional medical center.

Automotive.

A team has built a prototype 24 GHz software-defined radar system capable of operating as a radar and communication device on vehicles [2]. This software-defined radar architecture uses direct digital synthesizers to create the analog waveforms and a digital signal processing chip to perform signal analysis on the sampled data. Their system functions as a frequency modulated continuous wave radar pseudo-noise code transmitter. This system demonstrates how multiple functionalities can be integrated into a single processing architecture. A centralized system capable of

handling multiple functions in a vehicle is one of the major advantages of using a software-defined radar.

Weather Radar.

The Multifunction Phased Array Radar program, undertaken by National Oceanic and Atmospheric Administration, requires many individual receiving channels for use in a distributed sensing array. A team researched this problem and designed a custom solution to decrease the cost of the distributed receivers [3]. This design uses the custom digital receiver to create a module capable of receiving and processing through eight channels at each geographic location. The design decreases the cost of each channel to approximately 200 dollars, far below the cost of off-the-shelf hardware solutions.

Intrusion Detection.

Researchers have implemented software-defined radar that functions as a home intrusion detection system [9]. The system utilizes a pseudo random signal source from a software-defined radio front end. The system then uses a classification algorithm to determine if there is an intruder within a certain perimeter. This is an example of how readily available software-defined radios can be utilized as software-defined radars within the price range of consumers.

2.1.2 Deployed and Developing Software-Defined Radar Systems.

This section describes software-defined radars that are in current use as deployed systems or are in active research. These systems are in recent or current development for civilian and defense applications. The hardware used in each system is generally custom made for each application. This is in contrast to many of the research radars

that use software-defined radios as a component.

The Ohio State University Software-Defined Radar.

Researchers at The Ohio State University have developed a software-defined radar system for the Air Force Office of Scientific Research Discovery Challenge Thrust [10]. The software-defined radar distinguishes itself with a 500 MHz instantaneous bandwidth, offering coverage of many bands of operation.

Synthetic Aperture Radar.

Software-defined radars find use in synthetic aperture radars [4]. NASA Goddard Space Flight Center has developed two synthetic aperture radars for use in future experiments. The software-defined radars are designed for use in measuring topographical features in support of studying carbon release into the atmosphere [4]. These systems were flown on their first test flight in Spring 2014. The radars utilizes Field Programmable Gate Array (FPGA) to generate and process data. The FPGAs are controlled by a host computer over an Ethernet interface.

Raytheon Flexdar.

In 2014, Raytheon announced a new software-defined radar system developed under the Office of Naval Research Integrated Topside program [5]. This project goal is to build a system that will encompass “software-defined digital re-configurability at the foundational level” [5]. This serves as an example of how software-defined radars are mature enough to be integrated in defense applications.

2.1.3 Summary of Architectures.

Each of the architectures presented perform at least one function in software that classifies the radar as software-defined. Table 1 displays a summary of the features of the aforementioned software-defined radars.

One characteristic common across all examined architectures is the use of FPGAs. The configurations that use off the shelf components generate their data in the host machine controlling the software-defined front end. The custom made hardware devices utilize FPGAs to generate or process data. The FPGAs generate and process data at least an order of magnitude faster than a CPU implementation of data generation.

Table 1. Summary of Hardware and Data Generation in Examined Radars

System	Components	Digital Data
Medical Imaging	BladeRF SDR(FPGA based), off the shelf	Generated and processed by host machine
Automotive	Custom signal generator and DSP processor	Generated from stored as look up tables on DDS
Weather	Custom hardware based off FPGA evaluation board	Receiver only, IQ data processed after storage
Intrusion Detection	Uses off the shelf SDR (FPGA based) front end	Samples generated and processed in host machine
OSU Radar	Custom system utilizing FPGAs and DSPs	Back end computer controls FPGA and DSP operation
Synthetic Aperture	Custom hardware with FPGAs	FPGAs generate and process data

One characteristic common across all examined architectures is the use of FPGAs. The configurations that use off the shelf components generate their data in the host machine controlling the software-defined front end. The custom made hardware devices utilize FPGAs to generate or process data. The FPGAs generate and process data at least an order of magnitude faster than a CPU implementation of data generation.

2.2 System Description

This section describes the operation of the major components in the system. The descriptions below cover the theory, software, and hardware that comprise the system. The core radar functionality was created and developed by Stefan Wunsch in an open source software module [11]. The system is fundamentally composed of a host machine communicating over Ethernet to USRP N210 software-defined radio modules. The software-defined nature of the system implies that it is comprised of both digital and analog components. Figure 1 displays the components of the system and the high level demarcation of analog and digital domains.

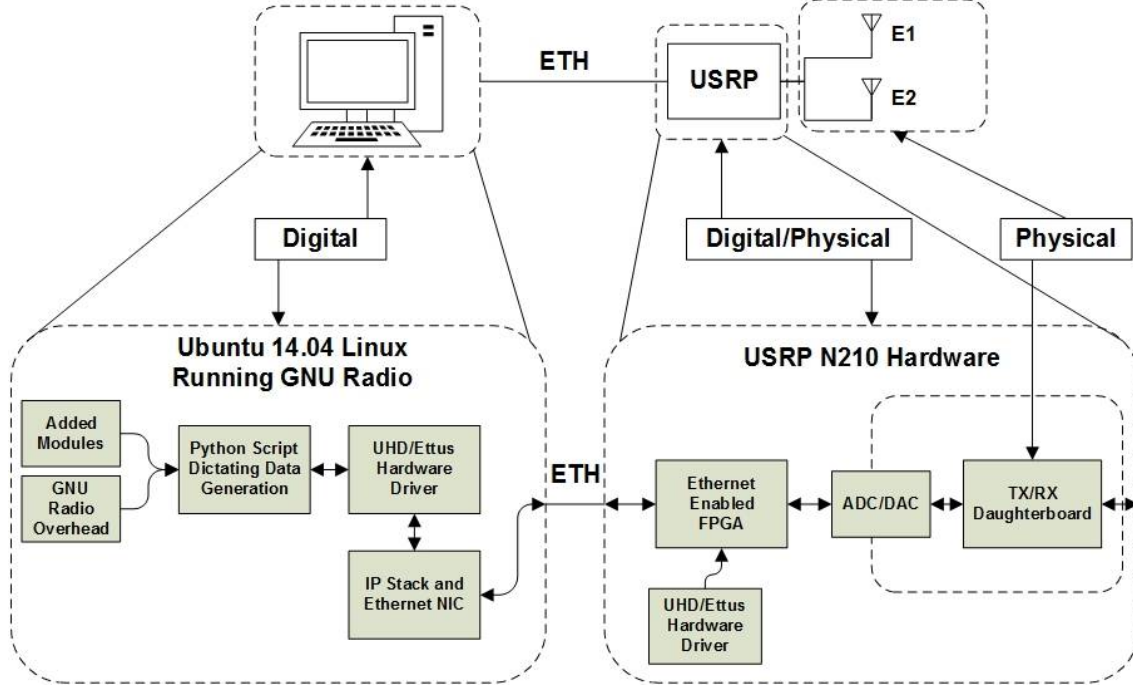


Figure 1. Architecture of Generic USRP and Host Machine System

The following sections describe the relevant components or processes used in the operation of the software-defined radar. The description of each of the components enables the construction of possible of attack and defense strategies targeted towards the system architecture.

2.2.1 Radar Operation.

The software-defined radar in this study is a dual continuous wave radar. This type of radar operates by transmitting a signal consisting of two single-tone frequencies f_1 and f_2 . The signal then propagates some distance R , reflects off a target, and returns to the radar aperture. During propagation, each tone acquires a phase shift as described in Equation 1 where ϕ is the phase acquired by a tone f_n , R is the distance to the target, and c is the speed of light.

$$\phi = \frac{4\pi f_n R}{c} \quad (1)$$

The range is estimated by comparing the phase acquired by each tone during propagation. Equations 2, 3, and 4 demonstrate the derivation of the range estimate. Each tone acquires some phase during propagation, as illustrated in Equation 2. The radar measures the phase difference as in Equation 3. The range is then determined by Equation 4 [12].

$$\phi_1 = \frac{4\pi f_1 R}{c}, \phi_2 = \frac{4\pi f_2 R}{c} \quad (2)$$

$$\Delta\phi = \phi_1 - \phi_2 = \frac{4\pi\Delta f R}{c}, \text{ where } \Delta f = f_1 - f_2 \quad (3)$$

$$R = \frac{\Delta\phi c}{4\pi\Delta f} \quad (4)$$

The maximum resolvable range of this radar type, R_{max} , depends on the difference of the two tone frequencies, Δf . The range limit occurs when $\Delta\phi = 2\pi$. The maximum resolvable range is determined by Equation 5.

$$R_{max} = \frac{2\pi c}{4\pi\Delta f} \quad (5)$$

2.2.2 GNU Radio.

GNU Radio is the software that controls the USRP N210 and performs all the signal processing computations in the system. The GNU Radio software runs on the host machine and performs data generation, hardware interface, and signal processing. The GNU Radio wiki best describes the GNU Radio software.

“GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems.” [8]

Below are descriptions of the GNU Radio Companion interface and the module processing blocks. The description of these modules provides the details of the software used to generate and process data within the software-defined radar.

GNU Radio Companion.

GNU Radio utilizes a graphical interface, GNU Radio Companion, to create signal processing chains. Figure 2 displays a simple flow-graph in GNU Radio Companion.

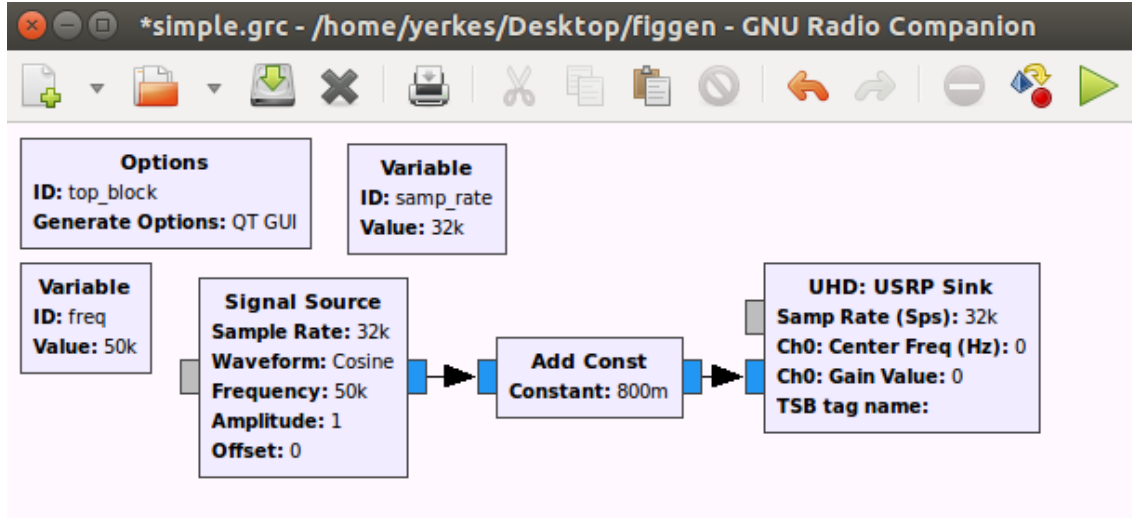


Figure 2. Simple GNU Radio Flow-Graph

Each block in the flow-graph represents a signal processing function. In this simple example, a signal source block is generating In-phase/Quadrature (IQ) samples for a 32 KHz wave and passing it through a block that adds a constant to each sample and then to a block that interfaces with a USRP N210. Flow-graphs can be assembled

from a variety of blocks inside GNU Radio to build signal processing chains that implement an array of communication systems. The flow-graph can interface with hardware, or run entirely on the host machine.

The GNU Radio companion software is simply a visualization tool. A GNU Radio Companion flow-graph is saved with a “.grc” file extension. The file is no more than an XML document that the GNU Radio Companion software parses and displays into a graphical representation of a signal processing chain. Figure 3 illustrates how the connections in the *.grc file correspond the visual connections in GNU Radio Companion.

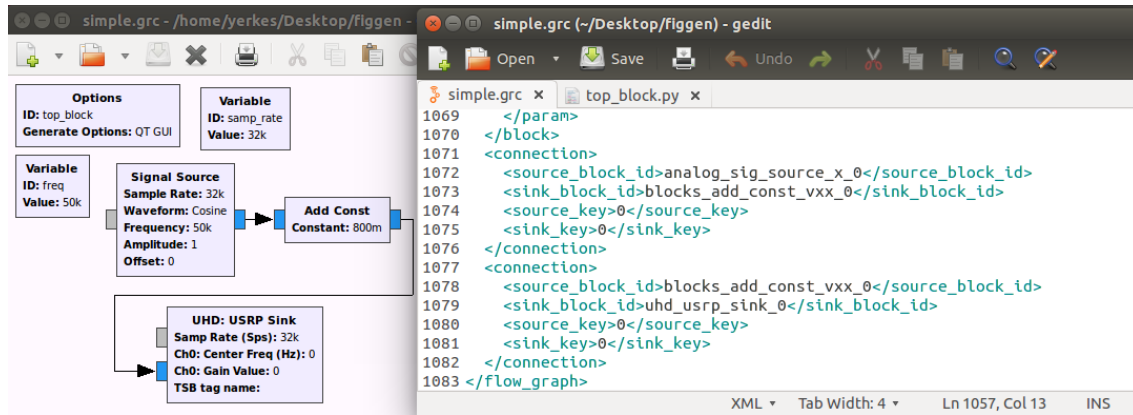


Figure 3. XML Generation From Flow-Graph

GNU Radio will parse the *.grc file into a Python script when the flow-graph is executed. This Python script is what executes and calls the modules within the flow-graph. Figure 4 illustrates the python script generated from the flow-graph in Figure 2.

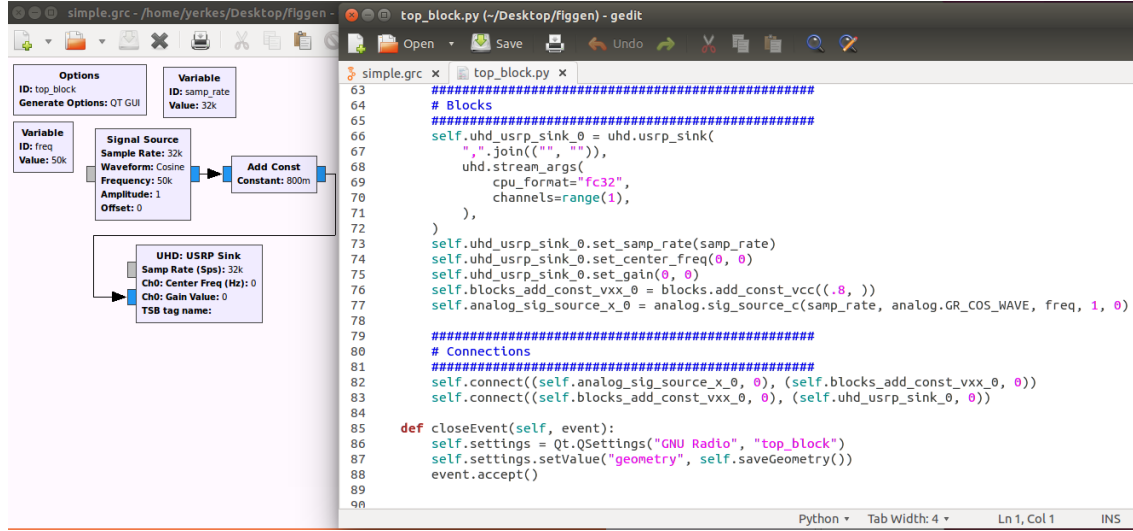


Figure 4. Python File Generated From Flow-Graph

Modules inside the flow-graph operate on data stored in a buffer passed to each module. The modules write any resulting data to an output buffer. The GNU Radio environment contains a scheduler that handles the specifics of module construction, flow-graph execution, and buffer management. The end result of this process is that samples are generated in sources and piped through to sinks within the GNU Radio flow-graph and managed by the scheduler.

Modules.

The modules and their connections define the processing that takes place inside a flow-graph. The modules act input output boxes that take in and perform some operation on input data as defined by their underlying code. The code that processes data inside a module can be written in either C++ or Python. This allows system designers to take advantage of the speed offered by C++, or quickly prototype a function in Python. Designers use a development engine supplied by GNU Radio, “gr mod tool”, to generate the files necessary to implement and import a new module into the GNU Radio software [13]. The modules written in C++ are compiled into

a shared libraries and added to the systems library path [14]. GNU Radio utilizes Simplified Wrapper and Interface Generator (SWIG) [15] to wrap and import the C++ compiled modules into the Python environment at runtime.

2.2.3 Universal Software Radio Peripheral N210.

The USRP N210 is a product developed by Ettus research, a National Instruments company, that operates as the re-configurable radio frequency front end in the system under test. Figure 5 displays the general USRP architecture as described by Ettus Research documentation [16].

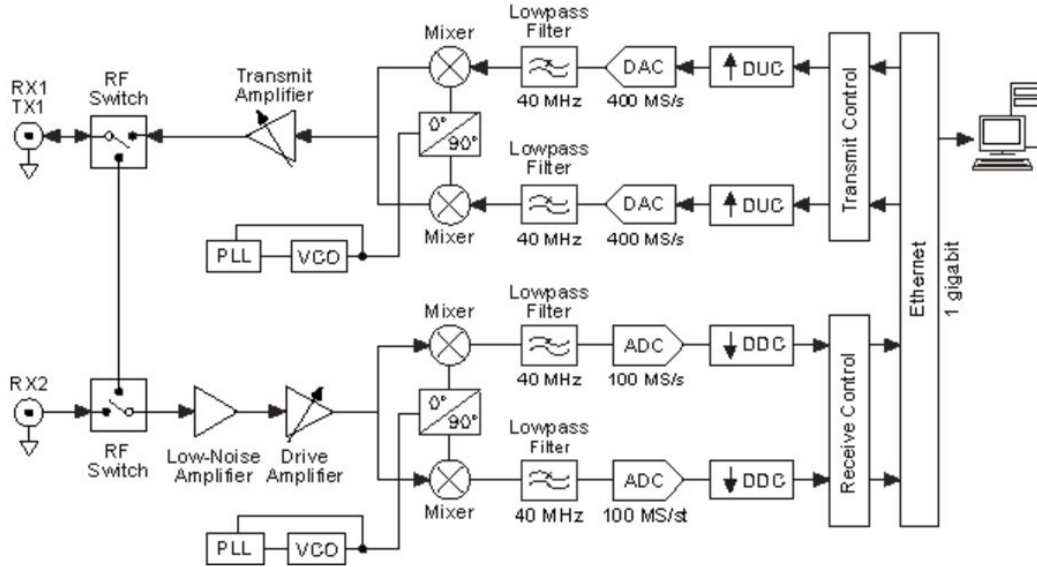


Figure 5. General USRP Architecture

The USRP series operates by streaming and receiving digital IQ samples to and from a controlling host computer. The host computer manages the operation of the USRP and generates any IQ data that the USRP series devices transmit. The USRP hardware processes and returns IQ data back to the host machine.

The general architecture of the USRP series presented in Figure 5 has the specific

implementation for the USRP N210 shown in Figure 6, the figure is part of the USRP documentation from Ettus [17].

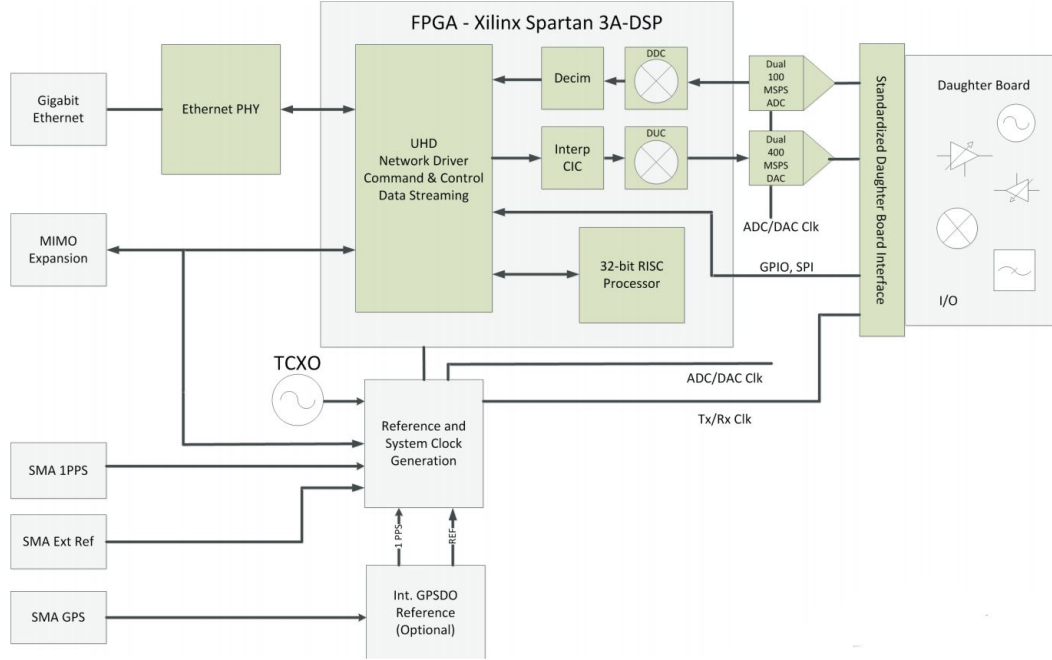


Figure 6. USRP N210 Architecture

The USRP N210 utilizes a Xilinx Spartan 3A-DSP FPGA as the primary processing module. The USRP N210 modules are designed to transmit and receive baseband IQ data when communicating with the host machine. The FPGA on board the USRP N210 handles all the signal processing and functions necessary to interface with the host machine. The FPGA also controls the hardware responsible for analog transmission and reception. The analog portion of the system is implemented by the radio frequency daughter-boards.

The daughter-boards are a series of interchangeable devices that interface with the USRP N210. The daughter-boards support various frequency bands, bandwidths, and gains. The daughter-boards communicate data to the FPGA through a Digital to Analog Converter (DAC) and an Analog to Digital Converter (ADC). The plug-and-play architecture of the daughter-boards allows the USRP N210 to operate in a

wide range of frequencies by switching daughter-boards.

2.2.4 Universal Hardware Driver.

The GNU Radio software does not directly control the USRP N210 hardware [18]. The GNU Radio software utilizes the USRP Hardware Driver (UHD) software to interface with the USRP N210. The UHD library software is responsible for managing the communication protocol between the host machine and USRP devices. The library translates application level data and commands into a format the USRP series will understand.

The UHD software includes the library on the host machine and the FPGA binary on board the USRP N210 that interprets data sent to the USRP N210. The software on board the host machine is coupled with the FPGA binary on the USRP N210 to enable the USRP to respond to commands issued by the host machine.

VITA 49 Radio Transport.

The UHD library will construct and send packets that conform to the general structure of the VITA 49 Radio Transport (VRT) protocol. The protocol is a suggestion to standardize data exchange between software-defined radios [19]. Figure 7 displays the various fields suggested by the VRT protocol as cataloged by Pentek [19].

Header (1 Word, Mandatory)
Stream Identifier (1 Word, Optional)
Class Identifier (2 Words, Optional)
Integer-seconds Timestamp (1 Word, Optional)
Fractional-seconds Timestamp (2 Words, Optional)
Data Payload (Variable, Mandatory)
Trailer (1 Word, Optional)

Figure 7. VITA 49 Radio Transport Fields

Each field in the protocol is a multiple of 32 bits and contains data relevant to various communication protocols. The VRT packet format is the basis for the data and command packets sent from the host machine to USRP N210.

Packet Formats.

The UDP packets sent to the USRP N210 can be grouped into two categories: control and data packets. The control packets are a subset of packets that configure the USRP N210 operation or maintain the protocol between the host machine and USRP N210. The data packets contain IQ data heading towards or coming from the USRP N210. Figure 8 displays the exact packet format for data and control packets.

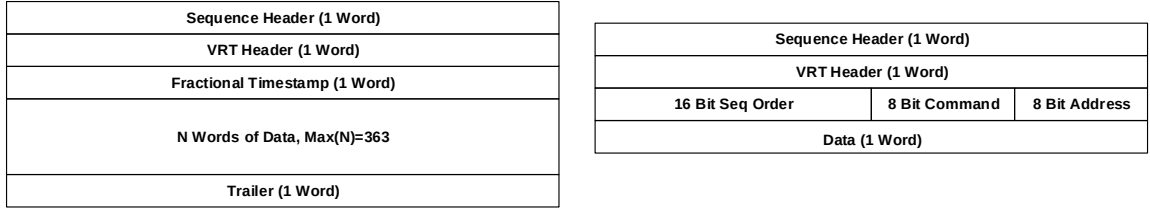


Figure 8. Data and Control Packet Format

The size of a data packet sent between the USRP N210 and host machine is limited by the Maximum Transmission Unit (MTU) of Ethernet, 1500 bytes. This implies there are 1472 bytes available for UDP payload data assuming a 20 byte IP header and an 8 byte UDP header. The UHD library will format data packets similar to the VRT packet format while also including a sequence header before the VRT payload.

Data packets sent to the USRP N210 contain a 32 bit sequence field, the VRT 32 bit header, 64 bit fractional timestamp, data field, and 32 bit trailer. The USRP N210 accepts and receives 16 bit IQ samples. Equation 6 displays the math that shows four bytes are needed for every 16 bit IQ sample.

$$\frac{16 \frac{\text{bits}}{\text{sample}} * 2}{8 \frac{\text{bits}}{\text{byte}}} = 4 \frac{\text{bytes}}{\text{sample}} \quad (6)$$

The maximum amount of samples per data packet can be calculated from the maximum UDP payload size, 1472 bytes. With a four byte sequence header, 4 bytes VRT header, 8 bytes fractional timestamp, and 4 bytes trailer there are 1452 bytes remaining for IQ data. With four bytes per sample, there is a maximum of 363 IQ samples sent per data packet.

Control packets contain the 32 bit sequence header, 32 bit VRT header, 32 bit command field, and a 32 bit data field. The FPGA on-board the USRP N210 interprets control commands and configures the hardware on the USRP N210 accord-

ing to the values of the control and data fields. A control word contains four bytes of sequence information, two bytes of command data, and two bytes of address data. The data field of a control packet defines the value of a specific parameter, such as sample rate or gain.

Over-the-Wire to Application Level Data Conversion.

The 16 bit sample format used by the USRP N210 requires that application data generated in another format be converted before being sent the USRP N210. The system under test generates application level IQ data as 32 bit floating point numbers. The UHD software converts the samples from 32 bit floating point (64 bits per sample) to 16 bit (32 bits per sample) as required by the USRP N210.

The UHD library utilizes the Streaming SIMD Extensions instruction set to accomplish this conversion. This instruction set controls 128 bit registers available on the processor to convert four application level floating point samples at a time to 16 bit format. It does this by multiplying each floating point number by a 32767.1035. This scalar is chosen so that a floating point value of one will map the the maximum value of a 16 bit integer, 0x7FFF=32767. The IQ samples generated by the flow-graph have amplitudes that are normalized between -1 and 1 facilitating the mapping to 16 bits. The conversion to application level data is the dual of this process, a 16 bit sample is divided by the scalar, and the value is cast as a floating point number. Understanding the mapping between application level data and wire format data enables the ability to interpret sent packets.

2.2.5 Gr-radar.

The gr-radar module is the open source GNU Radio add-on that implements the radar functionality of the system under test. The module was developed and

distributed by Stefan Wunsch and is available on Github [11]. The module supplies a set of flow-graphs, tools, and modules that implement the dual continuous wave radar system under test. Figure 9 illustrates the high level process of producing a range estimate.

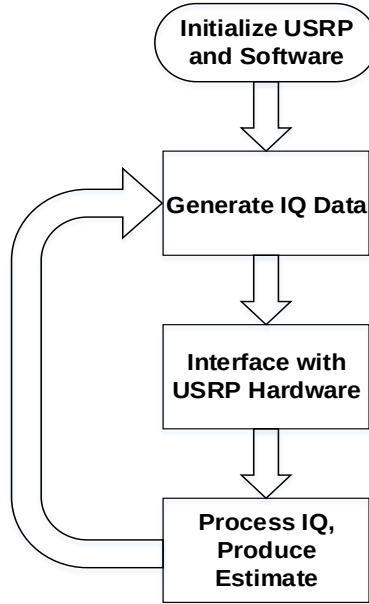


Figure 9. General Estimation Cycle

The flow-graph that implements the estimation cycle begins with the “Signal Generator CW” modules to produce the IQ data to be transmitted to the transmitting USRP N210. The interfaced phase takes place entirely within the “USRP Echotimer” modules. Every module following the Echotimer is used to process the data and produce a range estimate.

The important feature of the flow-graph is that the three phases of the estimation cycle are logically separated. The order of module execution within GNU Radio is determined by the connections between modules. The connections within the flow-graph dictate that the flow-graph execute the three phases of the estimation cycle in

order.

The flow-graph has a series of global parameters that dictate how the flow-graph functions. Table 2 reviews the function of each of global variables and the subsequent sections review the operation of each phase of the estimation cycle.

Table 2. Global Dual Continuous Wave Flow-Graph Parameters

Variable	Description
Sample Rate	The sample rate at which the flow-graph will generate data. The flow-graph will set the sample rate of the USRP N210 to this value.
Packet Length	The number of IQ samples produced during data generation and sent to the USRP N210. The USRP N210 will return the same number of samples.
Frequencies	The tuple representing the frequencies of the two tones. Each signal generator will generate one of these tones.
Center Frequency	The center frequency the host machine will request the USRP N210 modulate the data around.
Rx Gain	The entry representing the relative value the USRP N210 will set the Low Noise Amplifier and Variable Gain Amplifier to.
Tx Gain	The entry representing the relative value the USRP N210 will set the Base Band and Variable Gain Amplifier to.
Decimation Factor	The scalar value representing the decimation factor applied to received IQ data.
Threshold	The value in dB indicating the minimum received power level necessary for the estimation module to produce a range estimate.

Data Generation.

The data generation phase of the estimation cycle generates IQ data that represents the two-tone wave transmitted by the radar. Figure 10 illustrates the algorithm used to create the IQ floating point data for a single tone.

Input: F_s : The sampling frequency of the data to be generated, Φ_s : starting phase of the wave, f : frequency of the tone to be generated, N : Number of samples to generate

Output: f_t : N length array of IQ data for single tone

initialization: $\Phi = \Phi_s$;

```
for  $i$  from 0 to  $N - 1$  do  
     $f_t[i] = \text{re}\left(e^{\frac{j2\pi fi}{F_s} + \Phi_s}\right) + \text{im}\left(e^{\frac{j2\pi fi}{F_s} + \Phi_s}\right)$   
     $\Phi = \Phi + \frac{2\pi fi}{F_s}$   
end
```

Figure 10. Data Generation Algorithm

The “Signal Generator CW” modules implement this algorithm. The generator modules require an input frequency, amplitude, and number of samples to produce. The flow-graph generates a two tone wave by calling the generator modules with the same parameters except for the input frequency f . One module is called with an input parameter of $+f$ and the second module is called with a parameter of $-f$. The two tone wave results from summing the sequences generated from the $+f$ and $-f$ outputs of data generation modules. The flow-graph passes the N length IQ sequence to the USRP Echotimer module, initiating the Hardware Interface phase.

Hardware Interface.

The hardware interface portion of the estimation cycle takes place entirely inside the “USRP Echotimer” module. The Echotimer module was developed as part of the gr-radar toolbox plugin. The module interfaces with the UHD library to create a flow-graph block that sends and receives IQ data from the USRP N210. The Echotimer module maintains a list of parameters inside of the thGNU Radio Companion environment and initializes the USRP N210 with the specified values. Figure 11 displays a subset of the parameters the Echotimer uses during initializing a connection with the USRP N210.

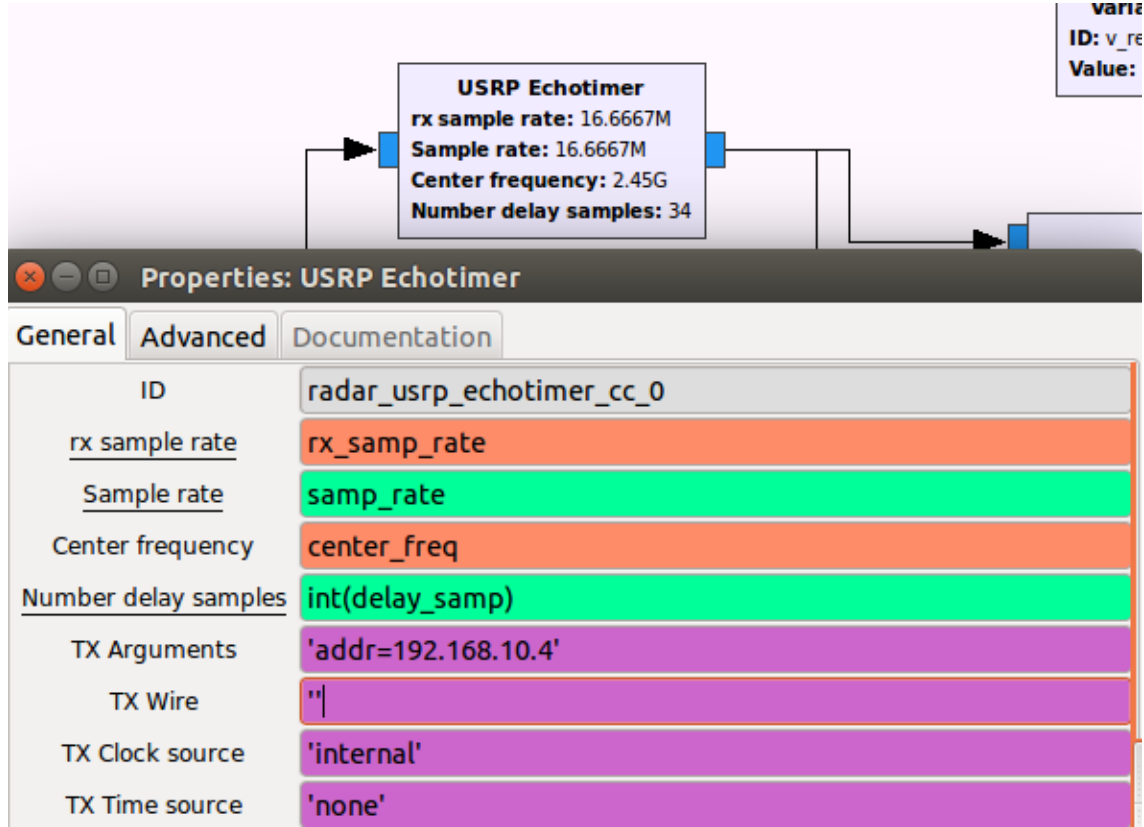


Figure 11. Echotimer Parameters

The module will pass these parameters to functions from the UHD API to set up the USRP N210 as specified by the field values. The Echotimer module is passed

the center frequency, IP addresses, clock setting, antenna settings, sample rate, delay samples, timing settings, and packet length.

During the estimation cycle, GNU Radio calls the Echotimer and passes in N floating point IQ samples. The Echotimer sends these samples through UHD methods that convert and stream the samples to the transmission USRP N210. The transmission USRP N210 receives the samples, pre-processes them, and sends them through the DAC converter and to the radio frequency daughter-board. The signal goes through the propagation process described in section 2.2.1. Then, the receiving USRP N210 receives the signal into through radio frequency daughter-board, samples it, and streams the samples back to the host machine. Figures 12 and 13 display the transmission and reception process respectively for IQ data.

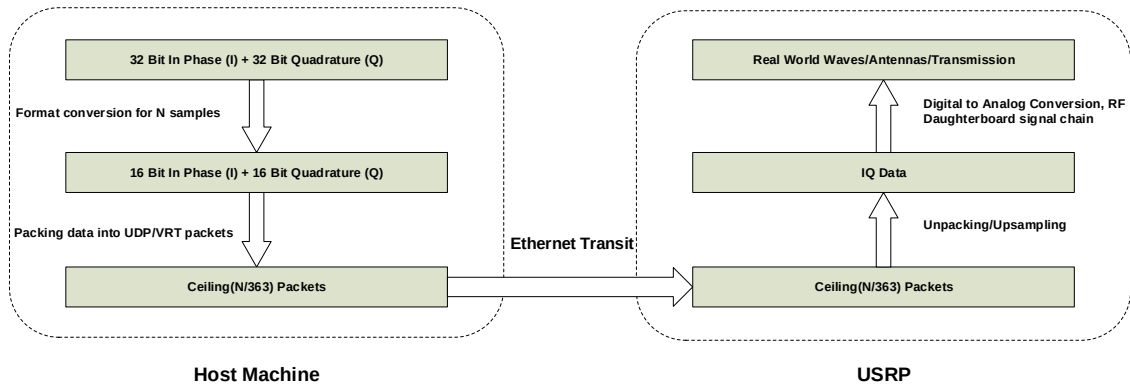


Figure 12. Transmission Process of IQ Data

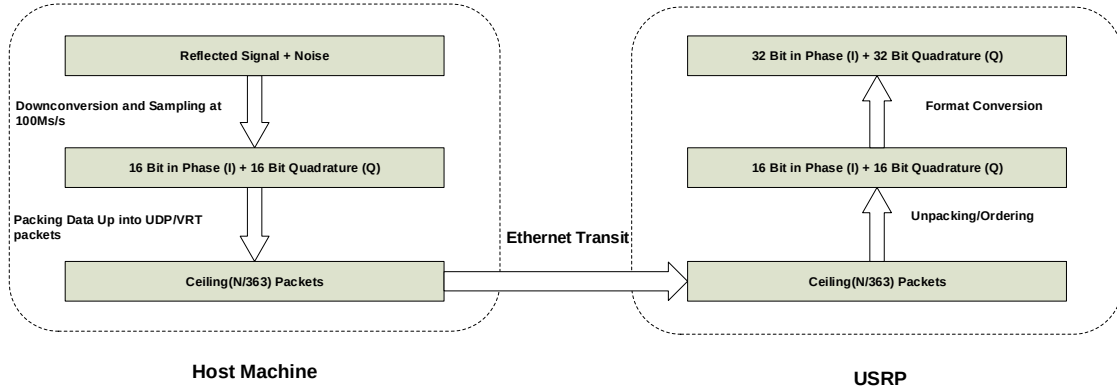


Figure 13. Reception Process of IQ Data

The Echotimer uses separate threads for the send and receive processes. The block creates a thread responsible for streaming IQ data to the transmission USRP N210 and a separate thread that waits for data to be returned from the receiving USRP N210. The Echotimer starts both threads and waits for both threads to end before checking to see that the correct number of samples was returned. The Echotimer then writes the samples to the output buffer, and returns control to the flow-graph. The flow-graph then begins the estimation process.

Estimation Algorithm.

The modules following the Echotimer implement the estimation algorithm that produces a range estimate. This section describes the signal processing on the received IQ data. The flow-graph first generates IQ data that represents a two tone wave. The signal generators create base band data that represents the wave. By summing the output of the $+f$ and $-f$ signal generators, the flow-graph creates a data sequence that represents the wave in equation 7.

$$e^{j\omega t} + e^{-j\omega t} \quad (7)$$

The IQ data representing the wave in Equation 7 is transmitted out the transmission USRP N210, propagates some distance, reflects off the moving target, and is returned the receiving antenna. The signal acquires a Doppler shift in the frequency of the tones, a phase from signal propagation, and a phase from the carrier demodulation. The IQ data returned to the flow-graph will be samples of a wave represented in Equation 8.

$$e^{j((\omega+\omega_{dopp})t+\Phi_{prop1}+\Phi_{carrier})} + e^{j((-\omega+\omega_{dopp})t+\Phi_{prop2}+\Phi_{carrier})} \quad (8)$$

The Doppler frequency shifts applied to each wave by reflecting off the target are approximated as equal. The USRP N210 does not perform carrier phase recovery. Therefore, each tone in the returned signal may have a phase shift from the analog mixing process.

The flow graph then performs discrete complex mixing by multiplying the returned signal by the complex conjugate of the sent data. The processing splits into two different channels at this point, as the signal is mixed with each tone f_1 and f_2 . This analysis designates these channels as c_1 and c_2 . Equations 9 and 10 displays the results of the mixing process for c_1 and c_2 .

$$c_1 = e^{j\omega t} (e^{j((\omega+\omega_{dopp})t+\Phi_{prop1}+\Phi_{carrier})} + e^{j((-\omega+\omega_{dopp})t+\Phi_{prop2}+\Phi_{carrier})}) = e^{j((\omega_{dopp})t+\Phi_{prop2}+\Phi_{carrier})} \quad (9)$$

$$c_2 = e^{-j\omega t} (e^{j((\omega+\omega_{dopp})t+\Phi_{prop1}+\Phi_{carrier})} + e^{j((-\omega+\omega_{dopp})t+\Phi_{prop2}+\Phi_{carrier})}) = e^{j((\omega_{dopp})t+\Phi_{prop1}+\Phi_{carrier})} \quad (10)$$

Strictly speaking, the frequency components that will be mixed to $\pm 2\omega$ will alias

back into the band of interest. For example, 12 MHz will be alias to -4.67 MHz. However, the algorithm that searches for spectral power will ignore peaks outside of a specified bandwidth that is far less than 4 MHz.

Channels c_1 and c_2 now contain only the Doppler frequency signal, propagation phase shift, and carrier phase. The flow-graph then decimates the signal in each channel, and performs a Fast Fourier Transform on each channel. The decimation decreases the size of the the Fourier Transform on the data, the spectral resolution of the frequency bins remains the same. The Fourier Transform of the decimated signals in the channels are designated as C_1 for channel c_1 and C_2 for channel c_2 .

The flow-graph then multiplies C_1 by the complex conjugate of C_2 . This implements a Cross Spectral Power Density operation. The Cross Spectral Power Density indicates if two signals share power in a frequency bin. The density also produces the phase difference between the frequencies of each signal. The signals c_1 and c_2 will share the same Doppler frequency of w_{dopp} , however the phase of each signal depends on what tone frequency was transmitted. Additionally, because $\Phi_{carrier}$ is the same for both c_1 and c_2 . The phase difference measured by the Cross Power Spectral Density, $c_1 - c_2$, will be the result difference between *the tone frequencies only*.

It is worth noting that this system does not perform any low pass filtering before decimation. This implies that higher frequency signals could alias into the band of interest. The transmitted waveform is tonal in nature and very narrow in the frequency spectrum. Therefore, any higher frequency content will not reside in the bandwidth the algorithm will narrow look for the Doppler frequency in.

The output of the Cross Power Spectral Density is fed to a module that will find the maximum peak of the spectrum. The module that finds the maximum peak only examines peaks within a one KHz window and ignores the zero Hz peak. This avoids the aliasing effect mentioned earlier. The value of the frequency, the phase

difference, and the power level is fed to a module that estimates the range. The range is estimated from the measured phase difference as is described in in Equation 4. A range estimate is only produced if the power of the received signal is above the threshold defined in Table 2. Finally, the flow-graph passes the range estimate to a GUI that graphs the results and optionally saves them to a file.

2.2.6 Ettercap.

Ettercap is a program available for Linux operating systems. The Ettercap man page best describes Ettercap [20].

“Ettercap was born as a sniffer for switched LAN (and obviously even “hubbed” ones), but during the development process it has gained more and more features that have changed it to a powerful and flexible tool for MITM attacks. It supports active and passive dissection of many protocols (even ciphered ones) and includes many features for network and host analysis (such as OS fingerprint).”[20]

While Ettercap has many capabilities, the Etterfilter engine and the ARP cache poisoning MITM attack are the relevant aspects of Ettercap in this study and are discussed below.

ARP Cache Poisoning with Ettercap.

Both the USRP N210 and the host machine maintain an Address Resolution Protocol (ARP) cache. The ARP associates an IP address with a hardware address [21]. Ettercap provides the capability to perform an ARP cache poisoning on a switched network. This attack modifies the ARP cache maintained on machines in the network to assign the hardware address of the MITM machine to the IP address of another computer. Figure 14 illustrates how the network traffic is redirected.

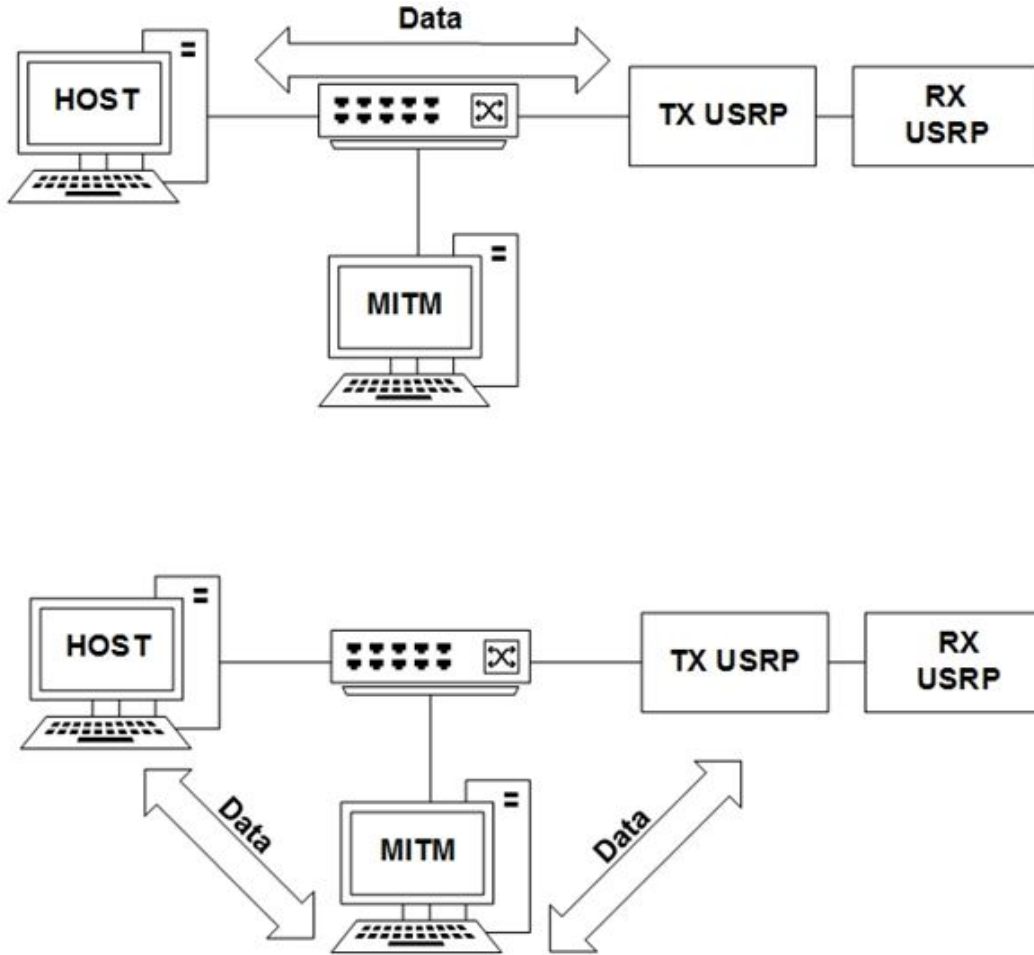


Figure 14. ARP Cache Poisoning Redirecting Traffic

By performing this attack on multiple machines, Ettercap can redirect all traffic between machines through the MITM machine, selectively modify it, and forward it. In this system, both of the USRP N210s respond to an ARP cache poisoning by redirecting the traffic through the MITM machine.

EtterFilter.

Ettercap can also modify the payload of packet piping through Ettercap. Ettercap offers the capability to modify packet data through a filtering engine, Etterfilter. Etterfilter is a compiler that produces binary files that the JIT interpreter in Ettercap

can understand [22]. Etterfilter supports the ability to examine IP, TCP, UDP, and Data fields. The filter engine does not support conditional modification of packets. That is to say, while Etterfilter can examine and replace specific bytes within a packet, it does not support any robust framework to monitor a protocol with changing field values. The Etterfilter files are limited to “if” statements and string comparisons of the data payload in a packet. The static nature of the filtering engine can be a limitation when attempting to follow and modify complex protocols. In order to dynamically modify packets, Ettercap offers a workaround wherein Ettercap saves packets to a file and calls an outside program to operate on the contents of the packet. However, this process does incur an extra processing delay due to file access time.

III. Investigative Process and Developed Tools

Chapter 2 detailed the high level concepts necessary to understand the operation of the system under test as found in gr-radar software module [11]. However, the gr-radar software required extensions to make the operation of the software suitable for testing. Additionally, new scripts and tools were developed to implement the attacks and defenses.

This chapter describes the work done during the research process. The sections below describe the products of the research as well as the method of completing the work. This should serve as a guide to any future work re-creating these results. The tools developed during research enable the description of the implemented cyber attacks and defenses on the system under test. The chapter concludes with a guide on how to secure the system.

3.1 Implemented Tools and Extensions

There are three areas of tools developed during the research process: extensions to gr-radar, host side defense tools, and MITM attack tools. This section describes these different areas and tools in the context of a cyber system.

3.1.1 Host Side Software Tools.

The host side software tools are defined as the addition, creation, or modification of software running on the host machine. This section describes the the addition of a radar calibration tool and a method to verify host machine data generation.

Extending gr-radar.

The hardware inside the USRP N210 will induce some time delay in the IQ sequence returned to GNU Radio. This delay will manifest as a phase error when processing the returned IQ data, which will lead to a range estimate error. The processing chain used in the experiments employs a rudimentary method of correcting for hardware delay. The only calibration method available was a method that would subtract a static distance from the distance estimate output of the processing chain. However, this method cannot correct the phase error induced within the USRP N210 hardware.

Adding or subtracting a static value for range can produce estimates outside the resolvable range of the radar. For example, the flow-graph processing chain measured the phase difference between the two frequencies and estimated the distance was six meters, but the actual target was at two meters. Subtracting a static offset of four meters would correct the range estimate. However, if the target then moves to an actual distance of two meters, subtracting a static offset of four meters will result in an estimate of negative two meters. A negative distance, or a distance greater than the maximum resolvable range, are impossible results if the radar estimation chain is implemented correctly. The the only valid results should be between zero meters and the maximum resolvable range.

If the resolvable range of the radar is quite large, the static distance correction would serve as a sufficient corrective technique. However, the system under test maintains a maximum resolvable range of 12.5 meters. The static distance offset is unacceptable for this range resolution. The solution is to correct for the phase error before the range estimate is produced. This was accomplished by extending the estimator module. Figure 15 displays the estimator module location inside the flow-graph. Figure 16 illustrates the source code modification inside the module.

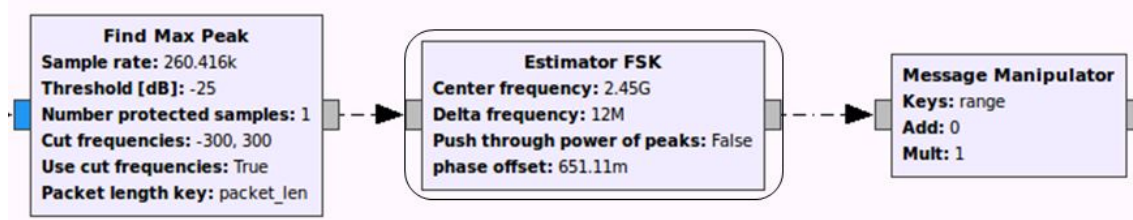


Figure 15. Estimator FSK Module

```
// Calc ranges and write to vector
d_range.clear();
for(int k=0; k<d_phase.size(); k++){
    std::cout<<"phase before k="<<k<<" phase="<<d_
    d_phase[k]+=d_phase_offset;
    std::cout<<" phase after k="<<k<<" phase="<<d_
    if(d_phase[k]>=0) d_range.push_back((d_phase[k]
    else if(d_phase[k]<0) d_range.push_back((2*M_
}
```

Figure 16. Calibration of Phase Inside Estimator-FSK

The modified code takes in a parameter that alters the received phase difference of the waves, modulo 2π , before estimating the range. This approach eliminates the possibility of estimating values outside the resolvable range of the radar. Modifying the source code accomplishes changing the actual functionality of the module, but it does not properly set up the module to interact with GNU Radio. Figure 17 displays the GNU Radio interface with the estimator module.

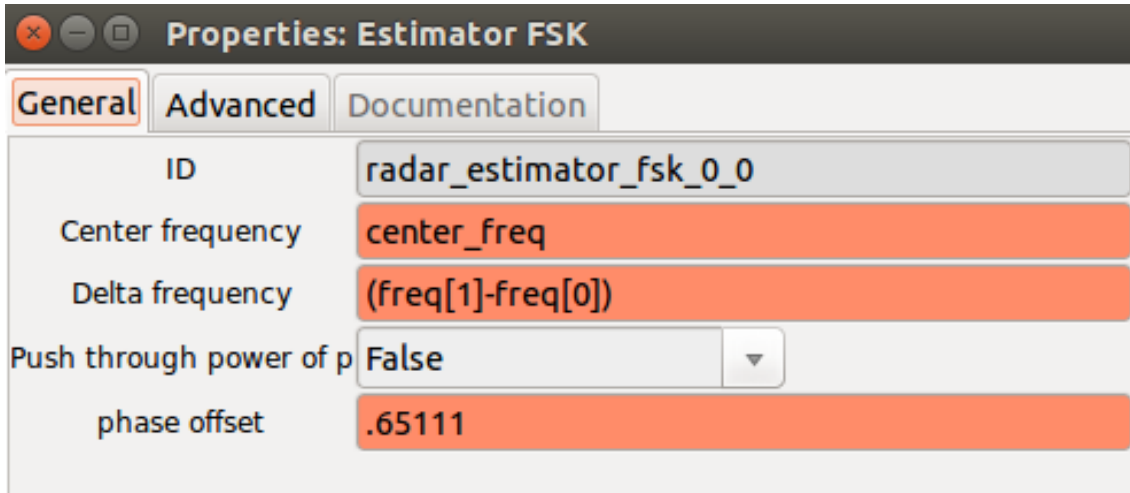


Figure 17. Estimator-FSK Inside GNU Radio Companion

The field “phase offset” was added to the XML description of the module. The extended XML description is illustrated in Figure 18.

```
<block>
  <name>Estimator FSK</name>
  <key>radar_estimator_fsk</key>
  <category>RADAR</category>
  <import>import radar</import>
  <make>fadar.estimator_fsk($center_freq, $delta_freq, $push_power,$phase_offset)</make>
```

Figure 18. Extended XML For Estimator-FSK Module

The new XML description is parsed by the GNU Radio software and passes the module parameters in Figure 17 to the underlying C++ constructor in Figure 19.

```
/*
 * The private constructor
 */
estimator_fsk_impl::estimator_fsk_impl(float center_freq, float delta_freq, bool push_power, float phase_offset)
: gr::block("estimator_fsk",
  gr::io_signature::make(0,0,0),
  gr::io_signature::make(0,0,0))
{
```

Figure 19. C++ Constructor For Estimator FSK

The modifications above allow a parameter that is changed within the GNU Radio Companion flow-graph to be passed to the underlying C++ modules. Tuning the radar is then a matter of varying the phase correction value until a suitable accuracy

is reached. This method assumes that the phase error is constant over the course of an experimental run. However, time, temperature, and drift can affect the phase characteristics of the USRP N210. It was observed that this method of calibration produced stable data over the course of experiments lasting approximately one hour.

Verifying Sent Packets.

During the initialization phase of the software-defined radar, the GNU Radio Companion flow graph will send a burst of packets that configure hardware settings on the USRP N210. The GNU Radio program utilizes the UHD library to accomplish the set up. However, there is no easy way of deciphering the stream of packets sent out of the host machine. Standard packet capture software such as Wireshark can display the sent and received traffic, but there is limited utility inside Wireshark to interpret the packet capture.

If an attacker managed to corrupt the library that interfaces with the USRP N210, they would be able to alter the control or data packets destined for the USRP N210. Additionally, they could corrupt any data coming back from a USRP N210. To mitigate this attack, the host machine must verify the data being sent or received from the USRP N210.

One method of verifying the sent data is to extend Wireshark to interpret the commands sent to the USRP N210. Wireshark supports the ability to add in packet dissectors to interpret different packet structures. Dissectors may be written in either C or LUA [23]. LUA is a lightweight scripting language that supports a variety of programming paradigms [24]. C offers faster dissection, but LUA offers a flexible structure that can be modified without recompiling.

The dissector implemented in this research utilizes LUA to dissect the command packets destined to the USRP N210. Figure 20 displays the dissector in action.

No.	Time	Source	Destination	Protocol	L
14	0.183569425	192.168.10.2	192.168.10.4	USRPN210 FIFO CTRL	

▶ Frame 14: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on i

▶ Ethernet II, Src: WistronI_69:2d:b8 (54:ee:75:69:2d:b8), Dst: National_6

▶ Internet Protocol Version 4, Src: 192.168.10.2, Dst: 192.168.10.4

▶ User Datagram Protocol, Src Port: 35241 (35241), Dst Port: 49159 (49159)

▼ N210 FIFO-XCVR: HOST->TX USRP

Word 1: Seq Header: 00000160

Word 2: VRT Header: 40000003

▼ CTRL Word whole : 01600116

Seq Out : 0160

CMD : 01

ADDR : 16

Word 4: VGAsset : 0000dc00 set VGA gain to 5

Figure 20. LUA Dissector Interpreting Packets Within Wireshark

The dissector is coded to generically parse command packets sent to the USRP N210. If the dissector recognizes the packet type, it parses the data field inside the packet and displays the value that the UHD driver is actually sending. The dissector currently supports dissection of transmission gain, reception gain, and sample frequency control packets. Figure 21 displays the code that recognizes the packets that set the gain used during transmission.

```

--this section of code searches for specific bytes or entire words in the command field of a packet sent to the USRP
--BB Gain Set Check--
if buffer:range(13,1):bytes()==_68 then
  pinfo.cols.info:set("setting baseband gain")
  if buffer:range(14,1):bytes()==_89 then
    words:add(buffer(12,4),"Word 4: BBset :      " .. buffer(12,4).. " set BB gain to 0 ")
  elseif buffer:range(14,1):bytes()==_99 then
    words:add(buffer(12,4),"Word 4: BBset :      " .. buffer(12,4).. " set BB gain to 1.5 ")
  elseif buffer:range(14,1):bytes()==_a9 then
    words:add(buffer(12,4),"Word 4: BBset :      " .. buffer(12,4).. " set BB gain to 3 ")
  elseif buffer:range(14,1):bytes()==_b9 then
    words:add(buffer(12,4),"Word 4: BBset :      " .. buffer(12,4).. " set BB gain to 4.5 ")
  else
    words:add(buffer(12,4),"Word 4: BBset :      " .. buffer(12,4).. " unknown BB gain set value")
  end
end
--VGA Gain Set Check --
elseif bit.band(buffer:range(14,1):uint(),_0F)==0x0c then
  pinfo.cols.info:set("setting VGA TX gain")
  local gval=0
  local readVal=buffer:range(13,2):uint()
  while readVal ~= 0x003c do
    readVal=readVal-0x0020
    gval=gval+1
  end
  words:add(buffer(12,4),"Word 4: VGAsset      :  " .. buffer(12,4).. " set VGA gain to " .. gval)

```

Figure 21. Code For Dissecting Transmission Gain Packets

Transmission gain is the product of two gain stages, base band and variable gain amplifier. The values for this script were determined by modifying the UHD library to display the register values before they packets are sent. The mapping process consists of sending application layer control commands while observing the console displaying what value was sent. Figure 22 displays the modification to the UHD source code that displays the register information.

```

/*****
* Primary control and interaction private methods
*****/
UHD_INLINE void send_pkt(wb_addr_type addr, boost::uint32_t data, int cmd){
    managed_send_buffer::sptr buff = _xport->get_send_buff(0.0);
    if (not buff){
        throw uhd::runtime_error("fifo ctrl timed out getting a send buffer");
    }
    boost::uint32_t *trans = buff->cast<boost::uint32_t *>();
    trans[0] = htonl(++_seq_out);
    boost::uint32_t *pkt = trans + 1;

    //load packet info
    vrt::if_packet_info_t packet_info;
    packet_info.packet_type = vrt::if_packet_info_t::PACKET_TYPE_CONTEXT;
    packet_info.num_payload_words32 = 2;
    packet_info.num_payload_bytes = packet_info.num_payload_words32*sizeof(boost::uint32_t);
    packet_info.packet_count = _seq_out;
    packet_info.tsf = _time.to_ticks(_tick_rate);
    packet_info.sob = false;
    packet_info.eob = false;
    packet_info.has_sid = false;
    packet_info.has_cid = false;
    packet_info.has_tsi = false;
    packet_info.has_tsf = _use_time;
    packet_info.has_tlr = false;

    //load header
    vrt::if_hdr_pack_be(pkt, packet_info);
    //std::cout<<"num_header_words "<<packet_info.num_header_words32<<std::endl;

    //load payload
    const boost::uint32_t ctrl_word = (addr & 0xff) | cmd | (_seq_out << 16);
    if(addr==SPI_CTRL){
        std::cout<<"send_pkt ctrl"<<std::endl;
        std::cout<<"num_header_words "<<packet_info.num_header_words32<<std::endl;
        std::cout<<"addr "<<std::hex<<int(addr)<<std::endl;
        std::cout<<"cmd "<<std::hex<<int(cmd)<<std::endl;
        std::cout<<"seq_out "<<std::hex<<int(_seq_out)<<std::endl;
        std::cout<<"num_header "<<std::hex<<int(packet_info.num_header_words32)<<std::endl;
        std::cout<<"data "<<std::hex<<int(data)<<std::endl;
        std::cout<<"entire control "<<std::hex<<int(ctrl_word)<<std::endl;
    }
}

```

Figure 22. Modifying UHD to Examine Values

The UHD library is recompiled to display the console data only during the LUA development process. The process requires that information be printed to the console for purposes of mapping and dissection. The library does not display console information during tests that produce range data. It is worth noting that having access to the source code makes the task of command mapping easier, but it is not necessary. The UHD API exposes methods to GNU Radio to configure the USRP N210. An attacker or designer could determine this mapping by using the application layer to

set the hardware settings and monitoring the traffic in Wireshark.

The packet content dissection was performed using the USRP N210 and XCVR radio frequency daughter-board. The LUA dissector currently only supports this configuration of hardware. The flexible nature of the LUA script should enable any additional dissection parameters to be easily extended onto the dissector.

3.1.2 Man-In-The-Middle Tools.

The MITM attacks in this study are conducted using Ettercap running inside a Kali Linux operating system. Once the MITM successfully deploys an ARP cache poisoning attack, all traffic between the host machine and USRP N210 is available to programs on the Kali machine. Naturally, sniffing programs such as Wireshark or tcpdump can log all the throughput. The sections below describe process behind generating the Ettercap filters used to modify data in transit. In some cases the appropriate filter parameters could be gleaned without knowledge of the UHD protocol. Observations as to the port numbers, packet structure, and protocol flow served to guide the exploration of the UHD code base.

USRP N210 Ports.

The communication between the host machine and USRP N210 uses UDP. The ports that the USRP N210 use are statically defined and serve different purposes. It is possible to discern the use of these ports through packet captures on the MITM machine with a few educated guesses and some knowledge of the operation of the host machine. Tools such as Wireshark or tcpdump can be used to examine the traffic between the host machine and USRP N210. The IP addresses can be determined with standard network mapping tools such as NMAP. Once an attacker has the IP addresses, they can easily determine some port allocations. For example, the port

45196 can be inferred as the transmission data port because of the large amount of data streaming to this port on the USRP N210 that transmits data. Similarly, the port 45197 is the port the USRP N210 returns IQ data from because of the large amount of traffic coming from this port. The port 49159 can then be assigned as the control data port.

These initial guesses led to a targeted search through the UHD code base for the specific port numbers. On Linux, the “grep” command offers a recursive feature to examine all file contents within a directory. This process revealed the port allocations as described in Table 3 from the file “fw_common.h”.

Table 3. Port Number Allocations

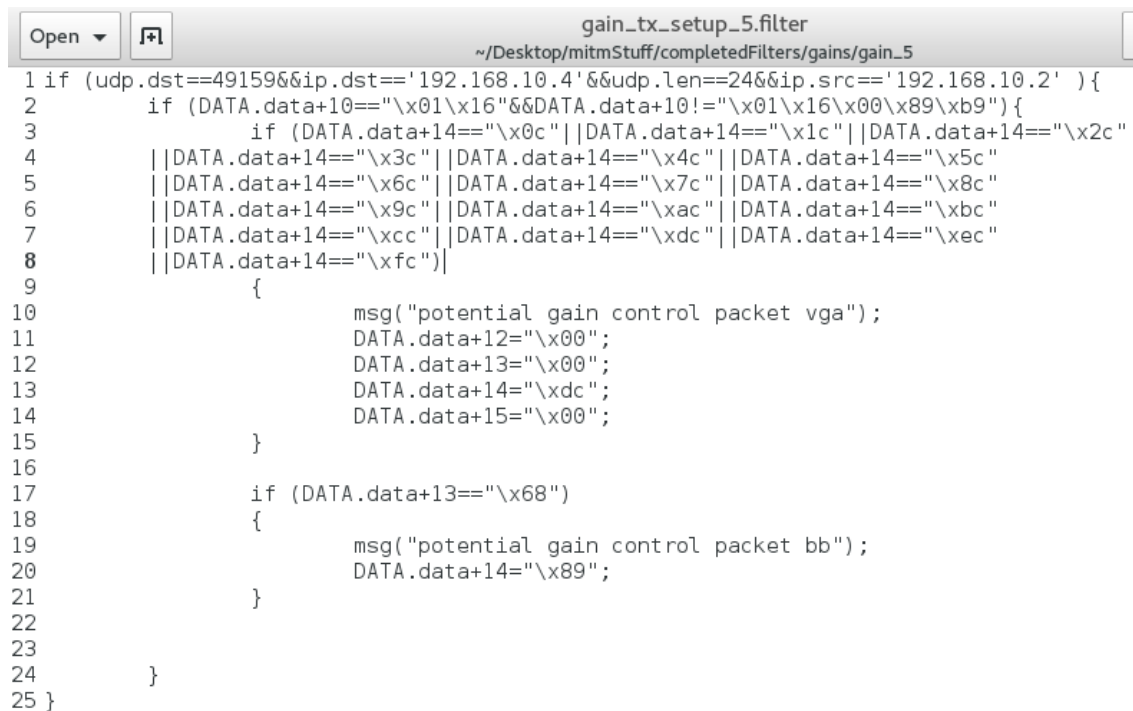
Port Number	Description
49152	UDP control port
49156	UDP Reception IQ return port. IQ packets are streamed from this port back to host machine
49157	UDP transmission IQ port. IQ packets are streamed to this port for transmission.
49158	UDP reception port 2, not utilized in this study.
49159	UDP FIFO control port. Host machine sends control packets to this port.

These port numbers can be found by recursively searching the UHD library code base for the port numbers observed in a Wireshark packet capture.

Configuration Command Generation.

The USRP N210 accepts two categories of commands: control and data. The control packets are commands that will configure the USRP N210 to operate with

specific parameters such as sample rate, gain, and transmission channel. Altering configuration commands is a simple matter of producing an Ettercap filter that targets the byte sequence unique to each type of command. The filter argument structure targets packets heading from the host machine to the USRP N210 control port. A specific hardware setting is targeted by altering the value in the register field of a control packet. Figure 23 displays an example of this structure for modifying the gain of the transmission USRP N210.



```

1 if (udp.dst==49159&&ip.dst=='192.168.10.4'&&udp.len==24&&ip.src=='192.168.10.2' ){
2     if (DATA.data+10=="\x01\x16"&&DATA.data+10!="\x01\x16\x00\x89\xb9"){
3         if (DATA.data+14=="\x0c" || DATA.data+14=="\x1c" || DATA.data+14=="\x2c"
4             || DATA.data+14=="\x3c" || DATA.data+14=="\x4c" || DATA.data+14=="\x5c"
5             || DATA.data+14=="\x6c" || DATA.data+14=="\x7c" || DATA.data+14=="\x8c"
6             || DATA.data+14=="\x9c" || DATA.data+14=="\xac" || DATA.data+14=="\xbc"
7             || DATA.data+14=="\xcc" || DATA.data+14=="\xdc" || DATA.data+14=="\xec"
8             || DATA.data+14=="\xfc")
9             {
10                 msg("potential gain control packet vga");
11                 DATA.data+12="\x00";
12                 DATA.data+13="\x00";
13                 DATA.data+14="\xdc";
14                 DATA.data+15="\x00";
15             }
16
17             if (DATA.data+13=="\x68")
18             {
19                 msg("potential gain control packet bb");
20                 DATA.data+14="\x89";
21             }
22
23     }
24 }
25 }

```

Figure 23. Ettercap Filter For Gain

The gain and sample rate packets can be delineated from each-other by the values of the control fields. Constructing filters that examine specific byte positions within the control word allows the filtering engine to replace the data payload with different byte values that correspond to a different parameter value than what the host machine sent.

Constructing different filters that specify different values is then simply a matter

of tabulating and recording what the host machine sends for different requests from the application layer. The mappings for transmission gain, reception gain, and sample rate settings are used in the LUA dissectors.

IQ Data Generation.

The next tool developed was a script to generate Ettercap filters that target the IQ data packets. The first task is to find a way to determine the order of the packets. This is necessary because the IQ data sent and received from the USRP N210 is time ordered. The IQ packet structure includes header information such as a sequence number and VRT header. However, due to static nature of Ettercap filters cannot examine these fields to dynamically replace the IQ payloads.

The solution to static Etterfilter restriction is to save the packet to a file and execute an outside program on the file. However, initial tests revealed this method to be incapable of maintaining line speed. Therefore, the assumption was made that the IQ sequence sent by the USRP N210 was the same for every estimation cycle. This is a fair assumption to make in the context of software-defined radar because some architectures use look up tables for the IQ data and do not generate it through a CPU. This method sends the same data for every estimation cycle and was found to produce accurate estimations when averaging the sample outputs. This is done to work within the limitations of Ettercap, but an FPGA or custom software implementation would be capable or working around this constraint.

The IQ sequence of a packet sequence headed to the USRP N210 was captured on the MITM machine using Wireshark. Then another packet capture is produced by exporting one burst of IQ data packets to the USRP N210 into a separate capture file. A Python script was created that examines the bytes within the packet at certain positions. The script can then determine the order of the data packets by finding

combinations of byte positions that are unique to a packet in the sequence. Figure 24 displays the packet capture used during parsing.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
2	0.000006	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
3	0.000008	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
4	0.000011	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
5	0.000014	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
6	0.000017	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
7	0.000020	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
8	0.000023	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
9	0.000025	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
10	0.000028	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
11	0.000042	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
12	0.000043	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
13	0.000044	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
14	0.000044	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
15	0.000045	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472
16	0.000046	192.168.10.2	192.168.10.4	UDP	1514	33552 → 49157 Len=1472

▶ Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
 ▶ Ethernet II, Src: WistronI_69:2d:b8 (54:ee:75:69:2d:b8), Dst: WistronI_69:2c:78 (54:ee:75:69:2c:78)
 ▶ Internet Protocol Version 4, Src: 192.168.10.2, Dst: 192.168.10.4
 ▶ User Datagram Protocol, Src Port: 33552 (33552), Dst Port: 49157 (49157)
 ▶ Data (1472 bytes)

Figure 24. Packet Capture Used to Generate Filters

Figure 25 displays the algorithm used to parse the packet capture and determine the unique byte position.

Input: *a*: iterable object of packets read from pcap from Figure 24, *Positions*: array of three positions to check

Output: A printout indicating whether specified positions have unique values in *a*

initialization: *l*= empty list of 3x1 tuples, 1byte per entry

```

for packet in a do
    make temporary tuple, tempTuple, comprised of the bytes at positions
    specified in Positions
    if tempTuple in l then
        print out error message
        exit
    end
    add tempTuple to l
end

```

Figure 25. Data Parsing Algorithm

The next step is to use these byte positions to generate Ettercap filters that will replace the IQ data during transit. A script is employed to facilitate producing scripts in a time efficient manner.

The script does the menial work of punching out the various fields inside each filter. This is essential for two reasons. The first is that Etterfilter compiler is very sensitive to syntax. The second is that this method of producing filters can generate filters for an arbitrary amount of data packets. The first part of the script generates the Etterfilter header that targets the IP address and port number of interest. The header examines the destination IP and destination port address to target data packets destined to the transmission USRP N210.

The next step is to generate the body of the Ettercap filters. The body of the

filter is responsible for replacing the IQ data of the payload. The IQ data is generated using a similar method used to generate data in host machine, but contains a phase difference between the two transmitted tones. A script then generates a header and body for the Etterfilter. The beginning of one filter is shown in Figure 26.

```
if (ip.dst=='192.168.10.4' && udp.dst==49157) {
    if (udp.len==1480 && DATA.data+16=="\x7f" && DATA.data+17=="\xff" && DATA.data+21=="\x69" ){
        DATA.data+16="\x00"; DATA.data+17="\x00"; DATA.data+18="\x00"; DATA.data+19="\x00";
        DATA.data+20="\x9d"; DATA.data+21="\x62"; DATA.data+22="\x00"; DATA.data+23="\x00";
        DATA.data+24="\x7d"; DATA.data+25="\xba"; DATA.data+26="\x00"; DATA.data+27="\x00";
        DATA.data+28="\xc2"; DATA.data+29="\x58"; DATA.data+30="\x00"; DATA.data+31="\x00";
    }
}
```

Figure 26. Ettercap Data Filter

The first condition of the filter checks to see if the packet is headed towards the data port on the USRP N210. The second check examines each byte position and then begins to replace all 363 samples with new data.

Finally, the number of filters generated is directly proportional to the Packet Length parameters in the flow-graph. In the case of this study, there are 362 packets generated that require 1448 data replacement commands inside the body of each Filter. The version of Ettercap used in this study crashed while using a single large filter for all 362 packets. The solution is to break up the Filters into smaller files that can all be loaded individually. This specific application requires the filter be broken into 16 separate filters that change the IQ payload for 362 different packets.

3.2 Implemented Attacks

This section details the implemented attacks possible on the architecture of the system under test. Each attack in this section would be applicable to other software-defined radars with a similar architecture. The architecture centralizes the data production and processing in the software and transfers the data to a front-end responsible for the transmission and reception.

3.2.1 Host Side Software Manipulation.

The software in the software-defined radar is ultimately a collection of libraries on the host machine that encapsulate the functions necessary to accomplish the signal processing. The system under test uses libraries that are a Linux Shared Object library (.so file extension). The nature of a Shared Object file allows other programs to import and use the binary instructions inside the Shared Objects. The GNU Radio software does not verify the library loaded into the environment. Therefore, one valid attack is to simply replace the library with an altered library. This technique is applicable to different operating systems as well. For example, the Windows equivalent of the Shared Object is a Dynamic-Link Library.

The new library can replace, replicate, or add any functionality to the original library. The effects of this attack are limited only by the ability of the attacker to correctly replace the library. There are two readily available methods for an attacker to create a modified version of the library. First, the attacker could recompile the library with a modified version of the source code. This implies the attacker has access to the source code. This type of threat would be representative of an insider compromising the system by leaking code. The attacker could also acquire the library from the host machine and modify it with traditional reverse engineering tools such as IDA Pro [25]. In either case the attacker requires privileged access on the host machine to install the new library in the system library path.

Waveform Sniffing.

The communication between the host machine and USRP N210 is not encrypted. Ettus makes no claim that this system is encrypted or secure. This represents systems as they would be built with this open source protocol. This enables an attacker to capture any data in transit to the host machine and interpret it real time or store it

for later. If the attacker understands the data packets or can make educated guesses about the content of the data they can reconstruct the waveform being sent from the host machine. In this system, the ports used for data and control are static. It is reasonable to assume an attacker could determine the data channel based off the amount of traffic going to a specific port. In fact, this research surmised the data port, before finding the port allocations, by examining packet captures. Figure 27 shows the Wireshark traffic summary of the IQ data exchange.

Wireshark · Conversations · standardflow_lowpower										
Ethernet · 4		IPv4 · 3		IPv6 · 1		TCP		UDP · 22		
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Packets B → A	Bits/s A → B	Bits/s B → A	
192.168.10.2	57113	192.168.10.4	49157	3,547	4939 k	3268	279	3252 k	12 k	
192.168.10.2	53811	192.168.10.6	49156	2,923	4410 k	2	2921	72	3216 k	
192.168.10.4	49152	192.168.10.2	46712	662	51 k	331	331	19 k	19 k	

Figure 27. Wireshark Summary

It is clear that the data ports are the ports with the highest level of traffic at around 3 Mbits/s. The attacker can then use a script to parse the packets and reconstruct the sent IQ data. Even if the attacker did not know the data packet format, simply examining the data packets gives a good indication of what fields are changing between packets and where the data resides. The expected spectrum would contain two frequency peaks at 6 and -6 MHz. Figure 28 displays the spectrum of the signal as it is reconstructed on the attackers machine.

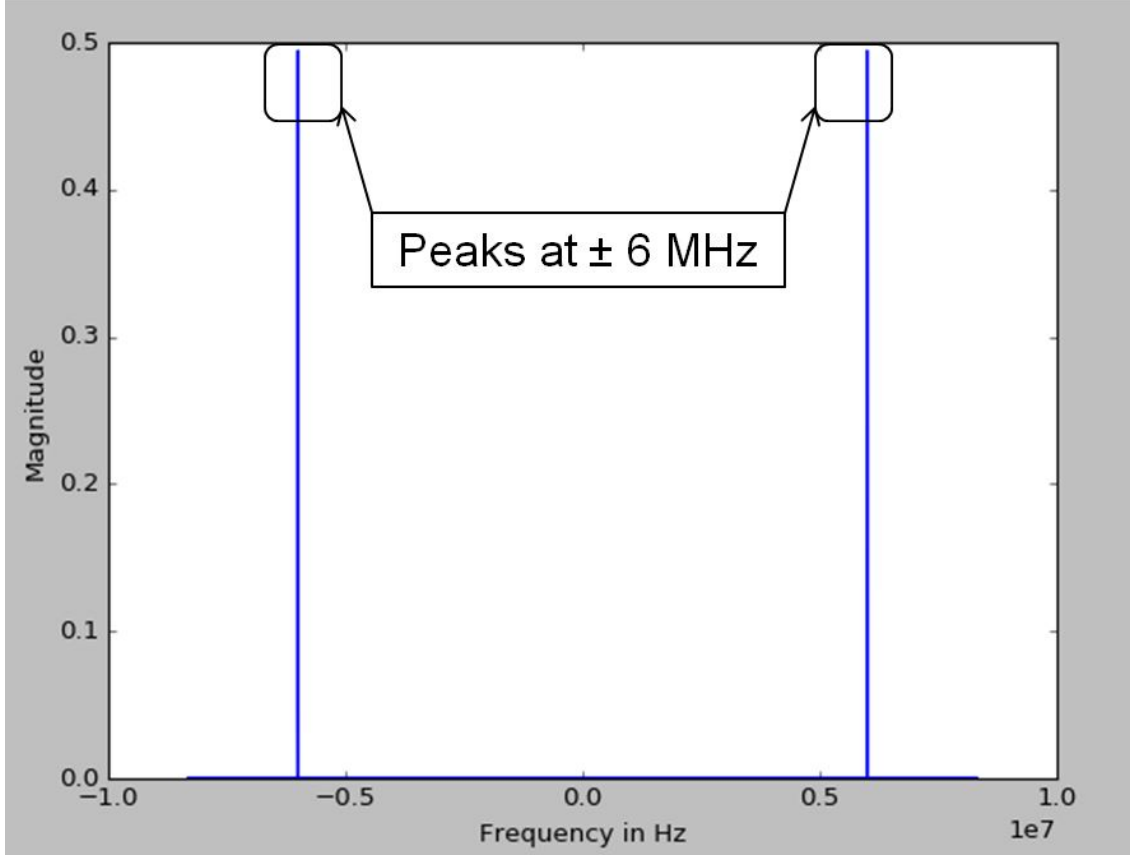


Figure 28. Reconstructed Spectrum on MITM Machine

The magnitude plot of the spectrum reveals that the MITM machine successfully detected the two frequency peaks of the transmitted signal. If the attacker knows the sampling frequency, they can display the exact spectrum of the transmitted signal by correctly assigning the frequency bin values. The IQ data packets are roughly VRT compliant. Therefore, Wireshark can parse every field except for the 32 bit sequence header in the IQ data packets. The time series data is constructed by parsing and ordering all 2^{17} IQ samples into a vector. The spectrum is computed via a standard complex Discrete Fourier Transform of the time sequence. The frequency bins are acquired by assuming a sample frequency. It is clear that the attacker could view the peaks on the spectrum to begin gathering data on the sent waveform.

3.2.2 Configuration Command Setting Manipulation.

The USRP N210 supports various settings for operational parameters (e.g. sample rate, transmission gain). The USRP N210 sets these values in hardware by interpreting the values sent by control packets and then setting the registers inside the USRP N210. The register values stored on-board the USRP N210 dictate how the USRP N210 will operate. The host machine sets these values by sending control packets that instruct the USRP N210 to set a specific register to a specific value. Therefore, if an attacker can identify specific control packets, they can modify the data field within the packet. The data field affects what value the USRP N210 set a register value to, and ultimately the value of a hardware parameter.

An attacker may either “replay” the data within a control packet or change the value of the data field. The advantage of replaying the data to the USRP N210 is that the attacker need not know what the value inside the packet is. The attacker need only know how to recognize the control word of a command packet and then replace the data payload of the packet with the previously captured value.

An attacker that wants to set the registers to a targeted value inside the USRP N210 may instead replace the data field of a command packet with a new value. If an attacker has some knowledge of what the value of that field maps to within the USRP N210, they can set a register to a specific value. This allows the attacker to set specific hardware parameters. For example, an attacker could modify the data fields of control packets to set the sample rate and gain to different values on the USRP N210 than what was requested by the host machine.

Ettcap filters are capable of modifying the control packets in the context of this system. These filters are loaded into Ettcap during the operation of the software-defined radar while the MITM attack is occurring. As the host machine sets up the USRP N210, the filters modify the gain and sample rate settings before the system

generates and receives IQ data.

3.2.3 IQ Data Manipulation.

The IQ data manipulation attack utilizes the filters that modify the IQ data sent to the USRP N210. The filters are loaded into Ettercap during the operation of the software-defined radar after the initial setup. Sets of filters were produced that changed the IQ data to have $\pi/2$ and $\pi/4$ differences between the two tone frequencies. These filters change the IQ data on the way to the USRP N210 before the data is converted into an analog signal. The transmitting USRP N210 will transmit these phase shifted signals. If the radar produces distance estimates by computing phase difference between the two waves, then any phase difference received will be interpreted as the effect of propagation. However, phase difference was added before the wave ever propagated.

This attack requires that the attacker have some knowledge of the estimation process taking place inside GNU Radio. This attack exploits the fact the distance estimation is the result of the phase difference between the two tones transmitted by the radar. By inducing a phase difference before the wave is transmitted, the position estimate of the radar will be altered by an amount proportional to the induced shift in the new IQ sequence.

3.3 Implemented Defenses

This research also implemented defenses against cyber attack vectors. The sections below detail what and how these defenses operate.

3.3.1 Delay Check.

A simple line of defense is to perform a timing analysis on the round trip time of the communication with the USRP N210. The MITM attack involves the attacker machine receiving, processing, and forwarding all data between the host machine and USRP N210. This process incurs delay in the round trip time between when the host machine sends and receives IQ data. Figure 29 displays the code inside the Echotimer that tracks the round trip.

```
uint64_t diff;
clock_gettime(CLOCK_MONOTONIC,&start);
d_noutput_items_send = noutput_items;
d_thread_send = gr::thread::thread(boost::bind(&usrp_echotimer_cc_impl::send, this));

// Receive thread
d_out_recv = &d_out_buffer[0];
d_noutput_items_recv = noutput_items;
d_thread_recv = gr::thread::thread(boost::bind(&usrp_echotimer_cc_impl::receive, this));

// Wait for threads to complete
d_thread_send.join();
d_thread_recv.join();
// end timer
clock_gettime(CLOCK_MONOTONIC,&end);
```

Figure 29. Timing Round Trip Time

The round trip time is determined by calculating the difference (in nanoseconds) between the start of the send data thread and the end of the receive data thread. During the construction of the module, the timing code opens a file that contains 3 entries: *windowSize*, *Average*, and *ValueAbove*. The *windowSize* parameter dictates the window size the code will keep a windowed average value of the round trip time. This allows an engineer to increase the window size if the variance is large. The *Average* represents the round trip average and is computed beforehand and imported during initialization. Finally, *ValueAbove* indicates the level above the *Average* that will trigger an alert that MITM attack may be taking place.

The code then checks to see if average round trip delay is above the *Average* plus the *ValueAbove*. If it is, the system displays a message that there may be an intruder

in the system. This method requires that the system be measured and calibrated while knowing that there is no attacker in the system. The *windowSize* and *Average* can then be measured and input into the initialization file. The *ValueAbove* parameter is chosen such that no single measurement in the calibration phase exceeds this value.

3.3.2 Software Modification Mitigation.

Host side software modification is defined as modifying the software used in the host machine. The manipulation can changed any data or commands issued by the host machine. One mitigation technique is to capture the packets being sent by the host machine, and compare the data and commands to the application level commands being sent. This technique requires third party capture software to record the packets sent from the host machine. The LUA script described previously is capable of parsing commands issued by the UHD library. By examining the commands that set the hardware values and comparing them with what the GNU Radio script is trying to set, one can validate the operation of the UHD library. For example, the designer can inspect the packet capture to ensure that transmission gain of requested by GNU Radio is actually sent to the USRP N210.

Another layer of security would be to ensure that the hash of the library is correct before the library runs. Commercial companies such as Carbon Black Inc [26], formerly Bit9, offer application white listing services. This could also be accomplished by simply running a hashing algorithm on each library used in GNU Radio before the radar begins to operate to ensure the library is not modified.

3.4 Securing The Software-Defined Radar

This research focused on finding and quantifying vulnerabilities in an example software-defined radar system. Naturally, the next step would be to extend the open

source hardware and software to include the option to authenticate the data between the host machine and USRP N210s.

3.4.1 Tools Required.

This section details the tools necessary to extend the functionality described in the following sections. The description of the tools below should serve as a guide to what equipment is required to secure the system.

The UHD library is written in C++. The library already has utility to extend functionality through the Make system in Linux [27]. As long as the dependencies of this system are met, the only C++ development tools needed are text editors and a console with root access to compile and install the library.

The FPGA image used on the USRP N210 is built using the Xilinx ISE [28] and GNU Make. The code for the FPGA images are available from the Ettus UHD Github repository. The Xilinx ISE is the design suite used to generate the cores used on the FPGA. GNU Make is software used in the Linux environment to make executables.

3.4.2 Encryption.

The most straightforward solution to securing the system is to provide encryption between the host machine and the USRP N210. There are devices on the market providing end-to-end authentication that can handle Gigabit Ethernet speeds that could provide this encryption [29]. However, a more holistic solution involves implementing security between the host machine and USRP N210 in the FPGA and UHD code. This would ensure that a solution could be distributed and patched into any system that required it.

Currently, the UHD library is open-source. This implies that the proposed solutions be rigorous enough to handle the case where either the host machine software

of the USRP N210 operation is modified.

3.4.3 Symmetric Key Encryption.

The easiest encryption method to implement would be a simple symmetric key encryption system that encrypts the payload of the UDP packets sent to the USRP N210. This system would encrypt all communication between the host machine and USRP N210. The most flexible implementation of this system would be to allow the USRP N210 to be programmed with a user specified key. This does not require a third party to keep the key secret. The USRP N210 and host machine then use this key to encrypt and decrypt all network traffic through an encryption scheme.

This could be accomplished by ensuring the USRP N210 hardware was directly connected to the host machine and using a utility to burn the key onto the USRP N210 memory. The USRP N210 FPGA image can then be modified to encrypt and decrypt with this key. The downside of this architecture is that if an attacker gains access to the host machine key, they can decrypt all the network traffic. This implies that the host machine and USRP N210 must be hardened against cyber attacks that can read the keys from the memory.

3.4.4 Public Key Infrastructure.

A more robust solution would be to use a Public Key Infrastructure (PKI) scheme to encrypt all network data. This solution would require the USRP N210 and host machine to utilize infrastructure that maintains the public keys. while encrypting all network traffic would be ideal, the amount of data produced during an estimation cycle may be prohibitive to encrypting all network traffic.

The PKI scheme can be improved upon by using the public and private keys to exchange symmetric keys. The symmetric key is then used to encrypt and decrypt

all traffic and is unique to each session. This decreases the workload on the system to encrypt data via PKI.

3.4.5 Message Authentication Code.

The USRP N210 and host machine need to verify the integrity and authenticate the content of any message sent over the link. One cryptographic solution to this problem is a message authentication code (MAC). In this scheme, both the sender and receiver maintain a copy of a private key that is used as the input of a cryptographic process performed on a packet. This process produces a MAC that is sent with the packet. When receiving a packet, the receiver of the message repeats the MAC production process and compares the MAC produced to the sent MAC. If the two MACs are equal, the message is authenticated. This scheme however cannot verify the sender of the message because the key is symmetric.

3.4.6 Digital Signature.

The proposed solution is to use a modified version of a digital signature to verify the data sent between the USRP N210 and host machine. Unless the entire packet payload needs to be encrypted, it is a waste of resources to encrypt every transmission. A scheme where the USRP N210 and host machine sign the data they send provides a good middle ground between complete encryption and no encryption. One solution is to encrypt a hash of the data with a private key and send it along with the payload data. Ideally, this would be a multiple of 32 bits to fit within the VRT packet format. The party receiving the data performs a hashing algorithm on the bits. The receiver then hashes the received data and compares output to the sent hash received hash. The receiver verifies the integrity when the two hashes are equal.

The public and private key would still have to be maintained in some repository

or maintained on each machine in this scheme. However, the payload data could still be interpreted from packet captures. Third party software could then inspect the packets if there was ever a concern of library tampering. The advantage of having a signature encryption is that the protocol remains open and can be monitored by auditing entities.

IV. Research Methodology

This section details the experimental methodology used to test the attacks and defenses from the previous section. First, the system specifications are described in order to facilitate recreation of any results. Then, the methodology of each attack is described.

4.1 Hardware Specifications

This section details the specific hardware used in the experiments. The subsections below cover the computing hardware, analog equipment, USRP N210 specifications, and network equipment.

4.1.1 Computer Specifications.

The host machine and the MITM machine are laptop computers. Table 4 displays the hardware information of each laptop.

Table 4. Host Computer Specifications

Component	Description
Brand and Model Number	Lenovo W541
Processor	Intel I7-4910 2.9GHz/3.9 Turbo
RAM	32 GB DDR3 SDRAM
Hard Drive	500 GB 7200 RPM Hard Drive

4.1.2 Analog Equipment.

Table 5 displays the information about the analog components used in the system. There are two antennas, two cables, two adapters, a multiple input multiple output (MIMO) cable, and two mic stands used for the analog portion of the system.

Table 5. Analog Equipment

Component	Description
Antenna	Tupavaco Panel Antenna, 2.4-2.483 GHz, 20 dBi, N female connector.
Cables	SMA male-male 3 foot cables.
Adapter	N male to SMA female RF coaxial cable adapter jack converter connector.
Mic Stand	Tripod PA speaker stands, antennas are mounted to these stands.
MIMO Cable	A synchronization cable connecting the two USRP N210s. The cable allows clock synchronization between the two modules.

4.1.3 USRP N210 Specifications.

The USRP N210 specifications are displayed in Table 6. The specifications are drawn from the specification sheet on Ettus webpage [30].

Table 6. USRP N210 Specifications

Parameter	Description
ADC Sample Rate	100 MS/s
ADC Resolution	14 bits
ADC Wideband	SFDR 88 dBc
DAC Sample Rate	400 MS/s
DAC Resolution	16 bits
DAC Wideband SFDR	80 dBc
Host Sample Rate (8b/16b)	50/25 MS/s
Frequency Accuracy	2.5 ppm
w/ GPSDO Reference	0.01 ppm
SSB/LO Suppression	35/50 dBc
Phase Noise (1.8 Ghz)	
10 kHz	-80 dBc/Hz
100 kHz	-100 dBc/Hz
1 MHz	-137 dBc/Hz
Power Output	15 dBm
IIP3	0 dBm
Receive Noise Figure	5 dB

4.1.4 Network Equipment.

The USRP N210s are networked to the host machine through an Ethernet switch with an Ethernet cable. Table 7 displays the information regarding network equipment.

Table 7. MITM Computer Specifications

Component	Description
Switch	Cisco SG100D-08P 8 Port PoE Gigabit Ethernet Switch
Cables	1x8 Foot CAT 5E cable, 2x5 foot CAT 6 Cable

4.2 Software Specifications

This section contains the details of the software used in this study. The version numbers or equivalent information are supplied for each major software component of the system.

4.2.1 Host Computer Specifications.

The host computer is the device running GNU Radio and the processes associated with it. Table 8 display the relevant software version information.

Table 8. Host Computer Software Specifications

Software	Description
Operating System	Ubuntu 14.04.3, Release 14.04, trusty
GNU Radio	Version 3.7.9.2
gr-radar	Downloaded May 6 2016
UHD	Version 003.010
Wireshark	Version 2.0.4

4.2.2 Man-In-The-Middle Specifications.

The MITM machine software specifications are displayed in Table 9.

Table 9. Man-In-The-Middle Computer Software Specifications

Software	Description
Operating System	Kali Linux 4.3.0
Ettercap	Version 0.8.2
Python	Version 2.7.12rc1

4.3 GNU Radio Companion Flow-Graph Parameters

All experiments in this study use the same GNU Radio Companion flow-graph during the testing process. The sections below detail the flow-graph parameters used in experiments that produce range data. A subset of the parameters are set by the designer. These parameters define the values of derived parameters.

4.3.1 Chosen Flow-Graph Parameters.

Table 10. Global Dual Continuous Wave Flow-Graph Parameter Values

Variable	Value
Sample Rate	16,666,667 Samples/sec
Packet Length	$2^{17}=131072$ Samples
Frequencies	-6,+6 MHz
Center Frequency	2.45 GHz
Rx Gain	20
Tx Gain	30
Decimation Factor	$2^6=64$
Threshold	-25
Phase Offset	.65111 radians

Sample Rate.

The sample rate was chosen such that the next supported sample rate below 16.666667 Ms/s still met the bandwidth requirements for our the transmitted signal. The bandwidth of the transmitted signal is nominally 12 MHz. The USRP N210 only supports certain sample frequencies, and it is necessary to pick a sample frequency slightly above the absolute minimum of 12 MHz to account for any spectral leakage. The next lowest sample frequency is 14.285714 Ms/s. The next lowest after that is 12.5 Ms/s.

Choosing 16.666667 MegaSamples per second (Ms/s) as the sample rate enables the study of an attack where the MITM modifies the sample rates used by the USRP N210 to 14.285714 Ms/s, which is not set below the minimum bandwidth but is

different than the rate the application requested. Setting the sample rate below the minimum sample rate causes aliasing, and produces invalid estimates resulting from the estimation chain.

Packet Length.

The peak traffic is linearly proportional to how many IQ samples are produced and received for each estimation cycle. The MITM machine cannot forward and modify high rate traffic. The original packet length produced a traffic rate of approximately 300 Mbits/s. This speed could not be maintained by the MITM and would cause the UHD software to throw errors related to timing. Therefore, the packet length was modified to suit the speed limits of the MITM machine.

The packet length of $2^{17}=131,072$ samples was chosen by halving the original sample size until the traffic rate produced by the flow-graph was slow enough for the MITM machine to handle without the UHD protocol causing a timeout error. Preliminary range results with the lower packet length were obtained and found to be sufficiently stable to use during experimentation.

Frequencies.

The two frequencies of +6 and -6 MHz were used to keep the maximum range of the radar to a small value. The maximum range for these frequencies is 12.5 meters. This short range makes finding a testing ground easier, antenna placement less critical, and single person testing practical.

Center Frequency.

The daughter-board used in this study can broadcast and receive in the 2.4-2.5 GHz and 4.9-6.0 GHz bands. The lower band was chosen because it was less noisy in

the testing environment. The center frequency of 2.45 GHz was chosen because it is in the center of the WiFi band. The center frequency can be modified as long as the bandwidth of the signal does not extend beyond the operational limits of the radio frequency daughter-boards.

Decimation Factor.

The decimation factor dictates by what number the IQ data will be decimated before performing the Discrete Fourier Transforms. The decimation factor of 64 was chosen to make the length of the discrete Fourier transform 2048 bits long. This transform length allows the flow graph to produce data faster than longer transform lengths while still maintaining a suitable degree of accuracy in the transform results.

Rx Gain, Tx Gain, and Threshold.

The reception gain, transmission gain, and threshold values were obtained by having an engineer stand six meters from the antenna apertures and oscillate the copper sheet while the radar is operating. This process will either produce or not produce range samples in the estimation chain. Whether a sample is produced depends on 4 things: reception gain, transmission gain, threshold, and the speed of oscillation of the sheet.

The oscillation speed of the sheet must be fast enough to induce a Doppler shift in the wave that causes power to reside in at least on discrete Fourier transform bin outside of zero. The frequency resolution of this system is 127.157 Hz. The frequency shift induced on a wave by a moving target can be measured by the Doppler shift formula as in Equation 11

$$\Delta f = \frac{\Delta v}{c} f_0 \tag{11}$$

where Δf is the frequency shift induced in the wave, Δv is the relative velocity of the object in reference to the observer, c is the speed of light, and f_0 is the frequency of the original wave. The two tones will be at $2.45 \text{ GHz} \pm 6 \text{ MHz}$. Their frequencies are approximately equal for the purposes of determining the speed the sheet must travel to induce a 127.157 Hz frequency shift. With $f_0 = 2.45 \text{ GHz}$, $c = 299792458 \frac{\text{meters}}{\text{sec}}$, and $\Delta f = 127.157 \text{ Hz}$ the required velocity is $15.56 \frac{\text{meters}}{\text{sec}}$. The sheet must be moving at least this speed to induce a 127.157 Hz shift in the wave incident on the sheet.

The other parameters were then adjusted and tuned to attempt to maximize the amount of valid estimates produced. The tuning process consisted of a cyclic adjustment of the parameters until the radar would produce valid estimates only when the target was present and moving. If the threshold is too low, background noise will be seen as a valid signal. Similarly, if the reception gain is too high then the receiving USRP N210 will amplify background noise above the threshold. If the transmission gain is too high the returns from targets beyond the intended target can produce estimates.

4.3.2 Derived Flow-Graph Parameters.

There are two flow graph values derived from the above parameters. They are displayed in Table 11 and described in detail after.

Table 11. Derived Dual Continuous Wave Flow-Graph Parameter Values

Variable	Value
Frequency Resolution	127.157 Hz
Range Resolution	12.5 meters

Frequency Resolution.

The frequency resolution is the spacing between Discrete Fourier Transform bins. The frequency resolution is a result of the sample rate and packet length. Equation 12 displays the equation where df is the frequency resolution, f_s is the sampling frequency, and N is the packet length.

$$df = \frac{f_s}{N} \quad (12)$$

It is worth noting that a larger number of samples yields a tighter frequency resolution for any given sampling frequency. This implies that the longer we take samples, the closer spaced the frequency bins will be, and the more sensitive the radar will be to velocity changes.

Range Resolution.

The range resolution is a result of the frequency choice and maximum range described in the background section. Using frequencies of +6 and -6 MHz provides a maximum range of 12.5 meters.

4.4 Attack Experiments

This section details the experimental process undertaken to quantify the effectiveness of the attacks implemented on the system under test.

4.4.1 Host Side Software Manipulation.

This experiment examines if the GNU Radio software is susceptible to an attack that alters the libraries used by GNU Radio during operation. The assumption of this experiment is that that attacker has either acquired the source code of the UHD li-

brary or has reverse engineered and altered the library from the binary file. The result is a new library the attacker generates that functions differently from the original.

The test is performed by replacing the copy of the compiled UHD library loaded by GNU Radio with a library that will perform in an altered way. Specifically, this test was performed by altering the the code that sets the transmission gain to ten above what was requested for gain request. Confirming that the experiment worked is accomplished by analyzing a packet capture of the exchange and noting that the gain requests are always set ten units above what what requested.

To facilitate this, the gr-radar code for the Echotimer module is modified to included three set gain requests every time the module is called. By placing these calls after the receive thread completes, the gain change request are easily tracked within the resulting packet capture. Figure 30 displays the modification to the UHD source code that produces the effect of always requesting that the transmitting USRP N210 be set to a value ten above what was requested.

```
void set_tx_gain(double gain, const std::string &name, size_t chan){  
    //std::cout<<"set_tx_gain inside usrp2_impl.cpp called"<<std::endl;  
    gain=gain+10;
```

Figure 30. Modification of UHD Code

The experiment is successful if the sent command packets are altered from the correct value for the zero, five, and ten gain requests. Figure 31 displays the modification to the source code that creates the change gain requests in each call of the Echotimer.

```
set_tx_gain(0);set_tx_gain(5);set_tx_gain(10);
```

Figure 31. Modification of Echotimer Source Code

Three requests are necessary because the UHD library may not request a gain

change if the state-tree inside the library indicates that a new request doesn't change the value on board the USRP N210. The UHD software on the host machine maintains a structure that tracks what values have been sent to the USRP N210. If the application requests the same setting twice the software does not send packets that set the value again. This reduces the traffic in the system.

4.4.2 Waveform Sniffing.

This experiment tests the capability of a MITM attacker to sniff the waveform sent from the host machine to the transmission USRP N210. The attack setup begins by connecting the host machine to the transmission and reception USRP N210s. The MITM machine then initiates an ARP cache poisoning attack to redirect the traffic through the MITM machine.

The MITM machine captures the packets during the data exchange with Wireshark. The Wireshark pcap file is then pre-parsed and handed to a Python script capable of traversing each UDP packet in the sequence and extracting the 16 bit IQ samples. The Python script then converts the 16 bit IQ to floating point samples.

The verification step in this experiment involves comparing the reconstructed floating point samples to floating point samples saved directly from the GNU Radio application. This is done by calculating the individual and average error of all floating point samples in the sequence.

4.4.3 Experimental Setup For Configuration and IQ Sequence Tests.

The physical setup for the configuration command and the IQ data manipulation experiments are identical. The maximum range resolution of the system is 12.5 meters. A suitable testing ground with a flat, straight, concrete walkway 15 meters long was selected as the testing ground. The two computers, two USRP N210s, net-

work switch, and associated connectors are placed on a table near the antennas. The antennas are mounted on mic stands and placed directly next to each other.

The start of a 300 foot measuring tape is then secured to the concrete with tape, and laid out flat along the concrete walkway. Tape markings for one through twelve meters are then taped onto the measuring-tape so that a constant reference is available. Figures 32 through 34 display photos of the testing setup.



Figure 32. Antenna Placement



Figure 33. Table Setup



Figure 34. Measuring Tape Setup

4.4.4 Sample Rate Configuration Command Setting Manipulation.

The next experiment observes the effect of changing the sample rate on the USRP N210s to a different value than what was requested by the host machine. This experiment utilizes an Etttercap filter to change a sample rate request from 16.666667 Ms/s to 14.285714 Ms/s. This ensures that the bandwidth of the signal is still within the sampling rate, but changes the rate the USRP N210 uses.

First the software-defined radar is started and set to save every distance estimate to an output file. Then a test engineer stands at five distances for three minutes each. The engineer stands at distances in the order of six, two, three, four, and five meters. In each location, the engineer is oscillating a copper sheet that serves as a target for the transmitted signal. In each three minute interval, the estimation process inside GNU Radio will either produce or not produce range samples. A valid sample is one where the radar produces a distance output at the end of the estimation chain. Invalid samples occur if the target is not moving, did not return sufficient power, or background noise caused ambiguity in the signal returned to the radar.

After each distance is measured, the radar estimation chain applies a -100 meter offset to every distance estimate and lowers the threshold to the minimum value. This ensures that the estimation process will always produce a result caused by a low threshold, but it will be between -100 and -87.5 meters. The radar continues to run while the filters are loaded, this ensures that the setup remains the same. The -100 meters offset helps delineate the data during processing.

The next step is to change the sample frequency the USRP N210s are using. The man in the middle loads the filters that target the sample rate change request packet. The filters are designed to change the payload of the packet to 14.285714 Ms/s. The test engineer then forces the Echotimer to use a function callback that initiates when the user inputs a new sample rate into the user interface. The callback generates the

packets that set the sample rate.

The distance offset and threshold are restored to their previous values, the negative value samples indicate when the engineer was implementing the sample rate change. The engineer then repeats the process of standing at six, two, three, four, and five meters for three minutes at a time. The output of this experiment is a series of range samples representing data taken before and during the attack. Data analysis compares the range results of these two data series.

4.4.5 IQ Data Manipulation.

This section details the experimental process of Ettercaps ability to modify IQ data in transit. The section first details the assumptions and the necessity of predictable IQ sequence. Then the experimental process is described.

IQ Sequence Predictability.

An Ettercap filter is capable of targeting the IQ data packets based on the destination based on port number, size, and payload data content. An Ettercap filter can be constructed to zero out all IQ data or replace every IQ data packet with the same payload. However, this approach has the effect of not producing any meaningful estimates in the GNU Radio Companion flow-graph. To create a targeted effect, it is essential to be able to order all 362 packets.

The VRT protocol has a sequence number field that could be used to help identify and order the packets. However, the sequence number can only hold values of zero through fifteen. There are 362 packets produced during each estimation cycle. Therefore, the sequence number cannot be used to uniquely identify and order all 362 packets. To mitigate this, the assumption is made that the host machine sends the same waveform for every estimation cycle. If this is the case, then the order of the

362 IQ data packets can be determined by the content as described in chapter 3.

Experimental Process.

This experiment observes the effect of altering the IQ data in transit to the transmission USRP N210. This experiment utilizes a series of 16 Ettercap filters that modify the IQ content of all 362 IQ packets. This experiment was conducted for four sets of filters that applied a different phase difference to the transmitted tones. For each set of phase difference filters the process below was conducted.

First, the software-defined radar is started and set to save every distance estimate to an output file. Then, a test engineer stands at six meters for three minutes oscillating a copper sheet. In the three minute interval, the estimation process inside GNU Radio will either produce or not produce range samples, similar to the sample rate change test.

Then the radar estimation chain applies a -100 meter offset to every distance estimate and lowers the threshold to the minimum value. This ensures that the estimation process will always produce a result, but it will be between -100 and -87.5 meters. The distance offset again serves to delineate data in post processing.

The next step is to load all 16 filters on the man in the middle machine that will change the IQ data. This process take some time, but the data produced by the radar during this time will be marked by being negative.

The distance offset and threshold are restored to their previous values, the negative value samples indicate when the engineer was implementing the IQ change. The engineer then repeats the process of standing at six meters for another three minutes. The output of this experiment is a series of range estimates before and during the attack. Data analysis compares these data series to each other.

4.5 Defense Experiments

This section details the process of performing the defense related experiments.

4.5.1 Software Modification Mitigation.

This experiment determines if the library replacement attack detailed in section 5.1.1 can be detected with tools outside the UHD and GNU Radio software. The tool that verifies that the software is sending the correct packets in this experiment is a LUA dissector imported into Wireshark. This experiment verifies that the LUA dissector developed can parse a series of commands setting the gain of the transmission USRP N210.

The Echotimer source code is modified to request a gain change on the transmission USRP N210 over the entire range of gain values. This is accomplished by modifying the Echotimer block source code. This will produce gain control packets during every execution cycle. The Wireshark capture is then examined to see if the LUA dissector successfully decoded the gain control words to the correct values. The experiment is a success if the LUA dissector successfully decodes all gain control values. If the LUA dissector can correctly interpret the gain control packets, it proves the effectiveness of an unbiased observer that the application level commands can be compared against. An third party observer can observe unencrypted traffic and validate the operation of the UHD code base.

4.5.2 Delay Check Intrusion Detection.

The delay check experiment determines if an application layer extension of the Echotimer code can detect if there is an attacker in the system. This experiment is run by first recording the average values of delay for every ten estimation cycles for five minutes without the MITM. The extended Echotimer code functions by importing

an average value, d_{av} , and a maximum value above the average d_{max} . The calibrating run determines what values those parameters should be set to.

The MITM attack is then initiated. The MITM forwarding process will cause an increased delay in the system. The flow-graph is then restarted and run for another five minutes. If at any point, the total delay from transmission of IQ data to reception of IQ is above $d_{av} + d_{max}$, then the system displays a warning that there may be an intruder in the system. The experiment validates the delay detection if it displays the “possible intruder” message if and only if a MITM attack is currently taking place for the five minute intervals.

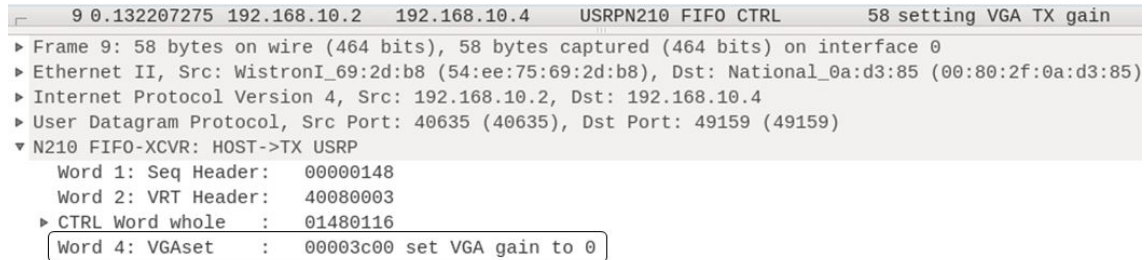
V. Results and Analysis

5.1 Attack Experiments

This section details the results from the experiments described in the methodology section. The data processing and results are described for each experiment in the sections below.

5.1.1 Host Side Software Manipulation.

The results of manipulating the UHD driver are illustrated by interpreting the results of a Wireshark packet capture. The three gain values used in the experiment were zero, five, and ten. These values are interpreted by the UHD software and parsed into two commands that set a base-band and Variable Gain Amplifier (VGA) gain. Figure 35 shows the unaltered packet command that sets the VGA gain to zero.



9	0.132207275	192.168.10.2	192.168.10.4	USRPN210 FIFO CTRL	58 setting VGA TX gain
▶ Frame 9: 58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface 0					
▶ Ethernet II, Src: WistronI_69:2d:b8 (54:ee:75:69:2d:b8), Dst: National_0a:d3:85 (00:80:2f:0a:d3:85)					
▶ Internet Protocol Version 4, Src: 192.168.10.2, Dst: 192.168.10.4					
▶ User Datagram Protocol, Src Port: 40635 (40635), Dst Port: 49159 (49159)					
▼ N210 FIFO-XCVR: HOST->TX USRP					
Word 1: Seq Header: 00000148					
Word 2: VRT Header: 40080003					
▶ CTRL Word whole : 01480116					
Word 4: VGAsset : 00003c00 set VGA gain to 0					

Figure 35. Wireshark Display of Unmodified Output For Gain Input of Zero

Figure 36 displays the VGA packet values for the modified UHD software that always requests the value ten above the application level input USRP N210, 15 in this case.

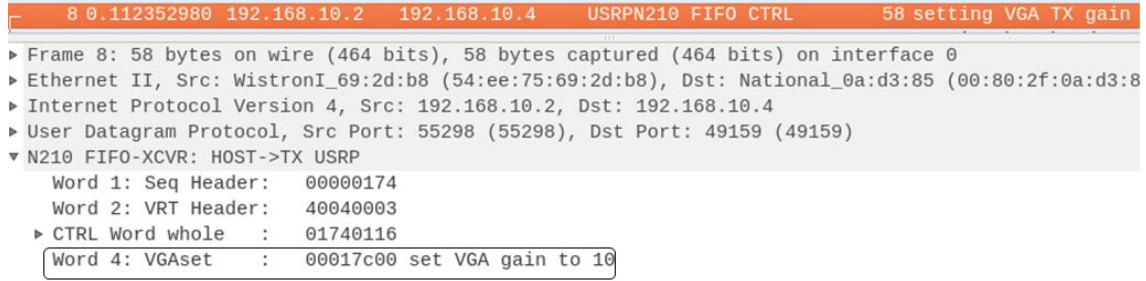


Figure 36. Wireshark Display of Modified Output For Gain Input of Zero

This is an example of how the library produces the incorrect control packet for an input of zero. Table 12 displays the data payloads of the packets generated from the unmodified UHD library and Table 13 displays the packet values from the modified library for all tested gain requests.

Table 12. Unmodified Payload Values from UHD Library

Input Gain	BB Packet	VGA Packet
0	0	0
5	0	5
10	0	10

Table 13. Modified Payload Values from UHD Library

Input Gain	BB Packet	VGA Packet
0	0	10
5	0	15
10	0	20

It is clear that the modified UHD library always requests a modified transmission gain ten above the intended value. This is a trivial example of how a similar architec-

ture must secure the software used in a software-defined system. The architecture of the system under test places the responsibilities of signal generation, hardware interface, and signal processing in the software of the host machine. The level of control the software maintains over signal processing responsibilities must be met with equal amounts of effort towards securing the software system.

5.1.2 Waveform Sniffing Results.

The ability of the MITM to reconstruct the waveform being sent by the host machine can be measured by how accurately the MITM reconstructs the 32 bit floating point IQ samples from the 16 bit wire format IQ samples. Therefore, the validation step compares the samples as they are reconstructed on the MITM machine to the saved samples from the host machine.

The GNU Radio Companion flow-graph was modified to include a module that saves the 32 bit IQ samples as raw binary data. The binary samples in this file are compared to the samples in a binary file generated on the MITM machine. The binary file generated on the MITM machine contains the 32 bit floating points samples resulting from reconstructing the samples from the packets. Reconstructing the samples is the reverse of converting the samples. Each 16 bit number is cast as a float and divided by 32,767.08203. This is the reverse process of mapping the -1 to 1 values to the 16 bit range. This puts the values back in the -1 to 1 range. Each IQ sample as generated in the flow-graph, IQ_1 , is then against the corresponding reconstructed sample, IQ_2 . The accuracy of each sample can be determined as Equation 13. Figure 37 displays the point-wise accuracy for all 2^{17} samples.

$$accuracy = \frac{|IQ_1 - IQ_2|}{|IQ_1|} \quad (13)$$

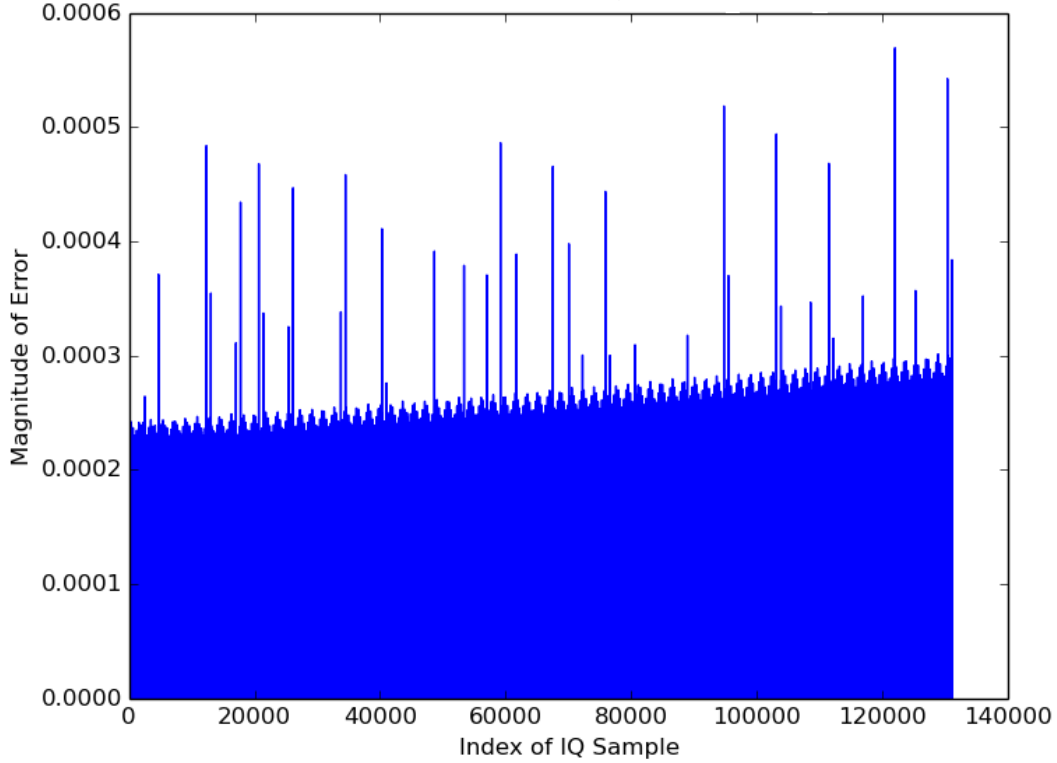


Figure 37. Accuracy of Reconstructed IQ Data

The accuracy is within .1% for all samples, indicating that the MITM machine can accurately reconstruct the sent waveform. The average error of all samples is 2.32508×10^{-5} , well within any reasonable error bound for waveform recovery. This analysis shows that recovering the sent waveform is possible on the MITM machine.

5.1.3 Sample Rate Configuration Command Setting Manipulation Results.

The results of the sample rate alteration can be analyzed by comparing the average distance measured before and during the attack. The data was analyzed by parsing the range estimates saved by GNU Radio Companion flow-graph into a comma separated value file. This file was then passed to the statistical software, Minitab [31],

and graphed. Figure 38 displays the average values and confidence intervals before and after modifying the sample rate.

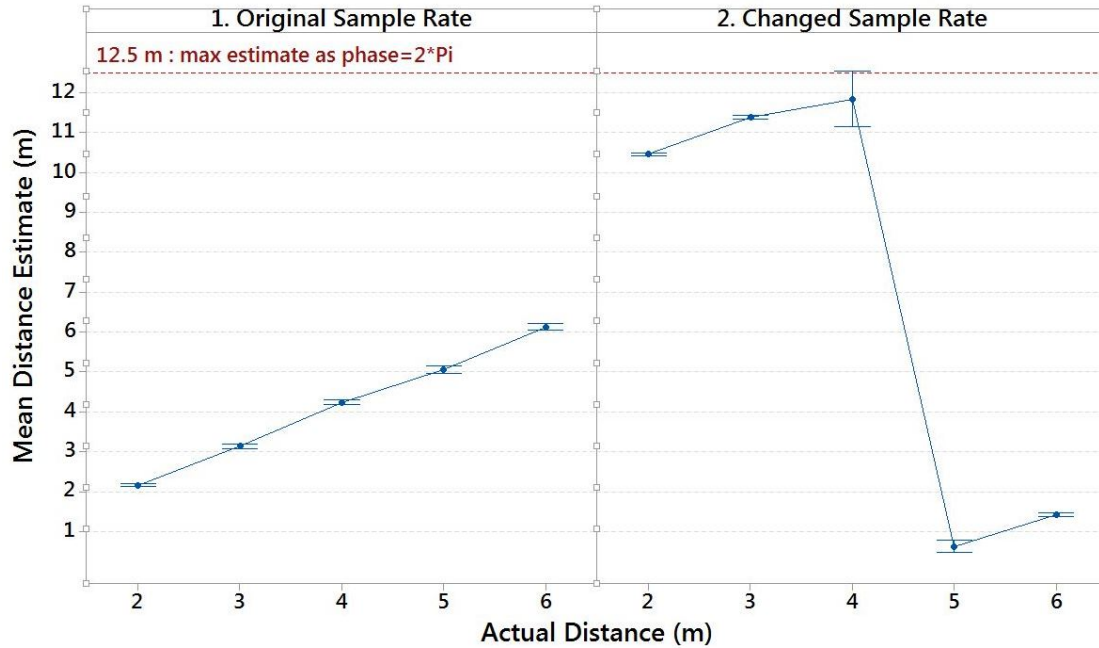


Figure 38. Sample Rate Modification Results

The average values of each distance is well outside the confidence interval for every measured distance. Additionally, the data for 4 meters displays a much larger confidence interval than any other point. This is due to the modulo nature of the range estimates. The range is aliasing between 12.5 and 0 meters, causing a wider variance in the sample data. Tables 14 and 15 displays the summary of the graphed data.

Table 14. Sample Rate Modification Summary: Pre-Attack

Range(m)	$\mu_{before}(\text{m})$	99% Interval(m)	n_{before}
6	6.13310	6.05112 to 6.21058	124
5	5.06735	4.97770 to 5.1570	161
4	4.25026	4.19494 to 4.30558	159
3	3.14316	3.08823 to 3.19809	165
2	2.17402	2.13877 to 2.20207	167

Table 15. Sample Rate Modification Summary: During Attack

Range(m)	$\mu_{after}(\text{m})$	99% Interval(m)	n_{after}	% Error
6	1.43017	1.39193 to 1.46842	49	76.6
5	.635539	.48132 to .785946	35	87
4	11.8509	11.1501 to 12.5517	65	178
3	11.39230	11.3516 to 11.4329	116	262
2	10.46320	10.4379 to 10.4885	151	384

The values μ_x and n_x refer to the average value and number of samples obtained during testing. The change of sample rate manifests in two areas. First, the number of valid samples obtained for six, five, and four meters is significantly less than before the sample rate change. This effect could be an result of imperfect target characteristics such as moving too slow. However, the distance of six meters is tested first both before and after the sample rate change, and two meters shows no significant drop in valid samples. This implies that the effect of reducing the valid number of samples is a result of the lowered sample rate.

The most striking disparity between the normal and post attack results is the

distance shift. The sample rate change has the effect of shifting all the distance estimates produced by the radar. The sudden skip between four and five meters is likely result of the modulo 2π aliasing of the radar. The results illustrate that simply altering a configuration setting creates inaccurate results. However, it was not possible to create a targeted result with this method. These results were part of a publication submitted to the International Conference on Cyber Warfare and Security [7].

5.1.4 IQ Data Manipulation Results.

The IQ data modification data processing is very similar to the data processing from the Sample Rate modification experiment. Minitab is used to process and display the range estimates from a comma separated value containing the values. This experiment differs in that the distance shift for each phase difference induced by the filters can be predicted and added to the analysis. The expected range shift is a function of Equation 4. The expected distance shift is ± 3.125 meters for a phase shift of $\pm \frac{\pi}{2}$ radians and ± 1.5625 meters for a phase shift of $\pm \frac{\pi}{4}$ radians. Figure 39 display the results for the $\pm \frac{\pi}{2}$ and $\pm \frac{\pi}{4}$ phase difference.

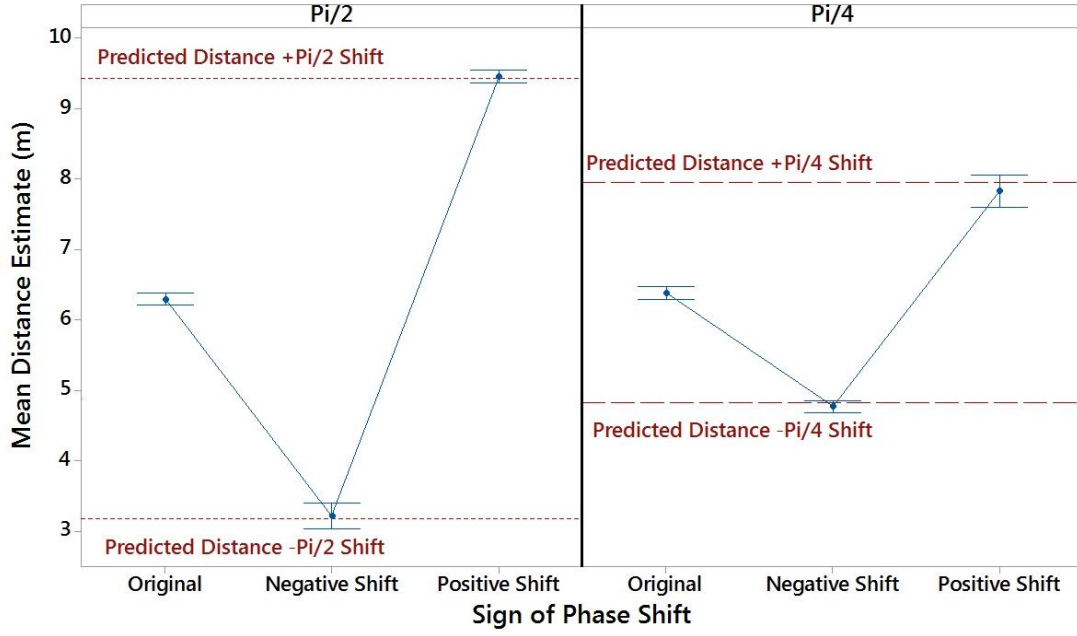


Figure 39. Effects of Introducing Phase Shift

It is clear from the confidence intervals and the average values that the IQ manipulation induced the predicted distance error. Tables 16 and 17 summarize the data from the experiment.

Table 16. IQ Data Modification Summary: Pre-Attack

Phase Difference(rad)	$\mu_{before}(m)$	99% Interval(m)	n_{before}
$-\frac{\pi}{2}$	6.29834	6.21291 to 6.38377	117
$\frac{\pi}{2}$	6.29834	6.21291 to 6.38377	117
$-\frac{\pi}{4}$	6.39070	6.29857 to 6.48256	112
$\frac{\pi}{4}$	6.39070	6.29857 to 6.48256	112

Table 17. IQ Data Modification Summary: During Attack

Phase Difference(rad)	μ_{after} (m)	99% Interval(m)	n_{after}
$-\frac{\pi}{2}$	3.21983	3.04016 to 3.3995	128
$\frac{\pi}{2}$	9.45692	9.36515 to 9.5487	103
$-\frac{\pi}{4}$	4.76985	4.68054 to 4.85917	98
$\frac{\pi}{4}$	7.83461	7.60444 to 8.06477	95

The values μ_x and n_x refer to the average value and number of samples obtained during a testing run. Table 18 displays the summary of how effective the induced phase shift was in producing the desired distance offset.

Table 18. IQ Data Modification Summary: During Attack

Difference(rad)	Expected Shift(m)	Measured Shift(m)	% Error
$-\frac{\pi}{2}$	-3.125	-3.07851	1.49
$\frac{\pi}{2}$	3.125	3.15858	1.08
$-\frac{\pi}{4}$	-1.5625	-1.62085	3.73
$\frac{\pi}{4}$	1.5625	1.44391	7.59

The percent error in Table 18 is calculated as

$$\%Error = \left| \frac{(\mu_{after} - \mu_{before}) - ExpectedShift}{ExpectedShift} \right| * 100 \quad (14)$$

Each measured distance shift from six meters is within 10% of the predicted shift value. It is clear that inducing a phase difference between the two transmitted tones creates a targeted distance shift in the output of the radar estimation chain. These results were part of a publication submitted to the International Conference on Cyber

Warfare and Security [7].

5.2 Defense Experiments Results

This section details the results for the two defenses implemented in this study. A timing analysis and packet dissector are presented.

5.2.1 Software Modification Mitigation Results.

Table 20 displays the parsed VGA values and compares them to the known value the UHD library will set the gain to for each application level gain request. Similarly, Table 19 displays the parsed base-band gain values for each gain request and how they compare to the known baseband value for each application level input gain.

Table 19. Baseband Transmission Gain Measurements

Input Gain	Expected BB	Measured BB
0 through 31	0	0
32	1.5	1.5
33 through 34	3	3
35+	4.5	4.5

Table 20. Variable Gain Amplifier Transmission Gain Measurements

Input	Expected	Measured	Input	Expected	Measured
0	0	0	1	1	1
2	2	2	3	3	3
4	4	4	5	5	5
6	6	6	7	7	7
8	8	8	9	9	9
10	10	10	11	11	11
12	12	12	13	13	13
14	14	14	15	15	15
16	16	16	17	17	17
18	18	18	19	19	19
20	20	20	21	21	21
22	22	22	23	23	23
24	24	24	25	25	25
26	26	26	27	27	27
28	28	28	29	29	29
30+	30	30	x	x	x

The LUA dissector successfully decoded the transmission gain packets for the VGA and base band amplifier mappings for all valid gain inputs.

5.2.2 Delay Check Intrusion Detection Results.

The calibration phase of the experiment indicated that the maximum windowed-average delay was never greater than 57,900,000 nanoseconds. The initialization file

was supplied an baseline average value of 57,807,637 nanoseconds with a threshold increase of 2,000,000 nanoseconds. Using these values, the man in the middle experiment detected a possible intruder for every estimation cycle while the MITM attack was occurring.

5.3 Summary

The attacks conducted on the system under test were successful in altering the operation of the system. In any software-defined radar there will be some machine running some type of software. Attacking this software is limited only by the attackers ability to acquire access to the computing resource. In this case the software was trivially modified to illustrate how this specific architecture does not verify loaded libraries. This lesson applies to the software component of any software-defined radar. Namely, the software must be secure for the system to be secure.

The other category of attack focuses on the communication link between the two devices in the system. The system under test is vulnerable to MITM attacks. The attacker can sniff, modify control, and modify data after data is redirected through the attacker machine. The attacker machine is capable of accurately recreating samples from sniffed traffic. The attacker can also employ readily available tools to modify traffic. Altering the sample rate created statistically significant error in the distance estimates, but could not produce a targeted effect. Altering the IQ data in transit created a predictable and targeted distance shift. In any software-defined radar system there will be a communication channel. The results of these attacks indicate that this channel must be secure for the radar to be secure.

The defenses implemented for the system under test were capable of detecting an intruder and deciphering network traffic. The first line of defense against a possible MITM attack is to monitor the timing characteristics of the system. A rudimentary

timing analysis system was capable of determining if there was a MITM attacker redirecting traffic. A Wireshark dissector was developed and shown to be capable of accurately interpreting command packets in transit to the USRP N210.

VI. Conclusion and Recommendations

6.1 Research Conclusion

The open source architecture examined in this study is not secure against cyber attack. The weak points fall into two categories: host side software and hardware interface. These vulnerabilities apply to any architecture where control is separated from the physical hardware. Many of the prototype software-defined radars are based on a similar architecture. It is imperative that a production radar based on these prototypes address the issue of security.

6.1.1 Host Side Software.

The software in a software-defined radar will vary depending on the architecture of the system. The exact nature of the software used on a host machine depends on the software environment, responsibilities of the software, and hardware language requirements. This software must be secure if the entire system is to be secure. This implies that every traditional means of securing a networked device should be employed on the host machine as a major step in securing any software-defined radar system. Examples include application white listing, log auditing, and operator feedback from using the software.

6.1.2 Hardware Interface.

Another potential weak point in a software-defined radar is where the software interfaces with hardware. There will always be some interface between software and hardware in any software-defined radar system. The nature of this interface varies depending on the architecture. Some architectures communicate waveform data and control commands through the interface while others may transfer only data or only

commands.

The system under test uses the hardware as a slave in a master-slave configuration. The host machine sends both IQ data and control information to the USRP N210s. This architecture allows for flexible data generation at the cost of requiring the host machine to generate data and process data.

Other architectures encapsulate the data generation in a single module and only utilize the software on the host machine to control the module. This architecture is used for higher performance software-defined radars such as those found in the synthetic aperture radar and distributed sensing applications. These architectures require that the module be pre-programmed with either data look-up tables or the algorithms necessary to generate the digital waveform data.

Both architectures have some channel between a host machine running software and the hardware performing the radar functionality. This channel should be the first thing examined when attempting to secure the system. Authentication between the endpoints is essential as the first step to securing the system. This is especially relevant if the channel is on a bus architecture where any device can observe the traffic between the software and hardware.

6.2 Current State

Software-defined radar is quickly maturing to a level where the technology is competitive with traditional radar systems. In some cases, open source software, hardware, and algorithms position this new technology for future use. However, these new applications have not been assessed from a security perspective.

6.3 Research Contributions

First, this research summarizes a variety of software-defined radar architectures as they exist currently. This survey provides a high-level view of software-defined radars found in research, commercial, and defense applications. The summary notes that the use of FPGAs is common across all architectures.

The scope of the research was limited to an available open-source software-defined radar. This radar shares key characteristics with all the software-defined radars found in background research, namely the use of FPGAs and a software control mechanism. The examination of this radar led to the conclusion that the first step in evaluating the security of software-defined radars is to examine the communication channel between the host machine and hardware. A communication channel from the host machine is common across all architectures and is potentially the most vulnerable part of the system. The effects of a compromised communication channel were published in the International Conference on Cyber Warfare and Security [7].

This research presents attack tools, defense tools, and plan for securing a radar based on the USRP N210. The MITM tools offer a glimpse into what is possible with readily available software. A major contribution is the LUA dissector plugin for Wireshark. This dissector presents the a general framework and method for adding in any functionality to decipher all USRP N210 traffic. The plan to secure a software-defined radar is left as a general guideline. The task of securing a cyber system is specific to each architecture. The guidelines provided offer high level suggestions based on the assumption that the target architecture is similar to the system under test.

6.4 Recommendations for Future Work

The system under test in this research is suited for an academic first step into developing a rigorous security evaluation method for software-defined radar systems. Creating guidelines, requirements, and doctrine for cyber assets is crucial for security. These requirements would create an environment wherein designers are required to implement security from initial design to deployment.

Future research should focus on the state-of-the-art software-defined radars available and focus on their deployment as cyber systems. The communication channel between the machine running the software control and the physical hardware is the first point to examine. But there is an entire field devoted to securing software as well. Future research should focus on the software inherent in these architectures. Future research should also place special emphasis on the interaction methods between the hardware and software radar components. Hardware fingerprinting and verification offer another layer of security against modifying the FPGAs and analog hardware used in the system.

6.5 Concluding Thoughts

This research was limited to a readily available open source architecture. The architecture shares characteristics with many research and deployed software-defined radars. The scope of the system is academic in nature, but nonetheless provides lessons on what areas of this new architecture are vulnerable and how they should be secured. The communication link and software of any software-defined radar are the components most critical to system security and should be the first things examined when auditing a software-defined radar. This assertion is supported by the results of this research.

Bibliography

1. J. Marimuthu, K. S. Bialkowski, and A. M. Abbosh, "Software-defined radar for medical imaging," *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 2, pp. 643–652, 2016.
2. Z. Hui, L. Lin, and W. Ke, "24GHz software-defined radar system for automotive applications," *European Microwave Week 2007 Conference Proceedings, EuMW 2007 - 10th European Conference on Wireless Technology, ECWT 2007*, no. October, pp. 138–141, 2007.
3. J. Meier, R. Kelley, B. M. Isom, M. Yearly, and R. D. Palmer, "Leveraging software-defined radio techniques in multichannel digital weather radar receiver design," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 6, pp. 1571–1582, 2012.
4. R. Rincon, T. Fatoyinbo, B. Osmanoglu, S. K. Lee, K. J. Ranson, G. Sun, T. Bolian, N. Goddard, and S. Flight, "Development of Next Generation Digital Beam-forming Synthetic Aperture Radar Architectures," pp. 2109–2111, 2016.
5. Raytheon. Raytheon Developing the Worlds Most Advanced Digital Radar. [Online]. Available: <http://raytheon.mediaroom.com/index.php?item=2556>
6. US Department of Defense. The dod cyber strategy. [Online]. Available: https://www.defense.gov/Portals/1/features/2015/0415_cyber-strategy/Final_2015_DoD_CYBER_STRATEGY_for_web.pdf
7. B. Yerkes, B. Ramsey, M. Rice, J. Pecarina, and S. Dunlap, "Security analysis of a software-defined radar." International Conference on Cyber Warfare and Security, 2017, (to appear).
8. GNURadio.org. What is GNU Radio and Why Do I Want it. [Online]. Available: <http://gnuradio.org/redmine/projects/gnuradio/wiki/WhatIsGR>
9. D. M. Chinnam, J. Madhusudhan, C. Nandhini, S. N. Prathyusha, S. Sw, R. Ramanathan, and K. P. Soman, "Intrusion Detection Using Software Defined Noise Radar," *Proc. 2010 Second International conference on Computing, Communication and Networking Technologies Implementation*, pp. 1–6, 2010.
10. OSU. Software Defined Radar Systems — ElectroScience Laboratory. [Online]. Available: <https://electroscience.osu.edu/radar-systems>
11. S. Wunsch. gr-radar: Gnu radio radar toolbox. [Online]. Available: <https://github.com/kit-cel/gr-radar>

12. B. Mahafza, “Radar Systems Analysis and Design Using MATLAB,” in *Radar Systems Analysis and Design Using MATLAB*, 1st ed. Boca Raton: Chapman and Hall/CRC, 200, ch. 3.
13. GNURadio.org. Welcome to GNU Radio. [Online]. Available: <http://gnuradio.org/redmine/projects/gnuradio/wiki>
14. tldp.org. Program library howto. [Online]. Available: <http://www.tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html>
15. swig.org. Welcome to swig. [Online]. Available: <http://www.swig.org/>
16. kb.ettus.com. About usrp bandwidths and sample rates. [Online]. Available: https://kb.ettus.com/About_USRP_Bandwidths_and_Sampling_Rates
17. Ettus. USRP N210. [Online]. Available: <https://www.ettus.com/product/details/UN210-KIT>
18. ——. USRP Hardware Driver and USRP Manual: Table Of Contents. [Online]. Available: <http://files.ettus.com/manual/>
19. R. Normoyle, “VITA 49 enhances capabilities and interoperability for transporting SDR data.”
20. linux.die.net. Ettercap linux man page. [Online]. Available: <https://linux.die.net/man/8/ettercap>
21. S. Bhatti and R. Atkinson. Address Resolution Protocol (ARP) for the Identifier-Locator Network Protocol for IPv4 (ILNPv4). [Online]. Available: <https://tools.ietf.org/html/rfc6747>
22. linux.die.net. Etterfilter linux man page. [Online]. Available: <https://linux.die.net/man/8/etterfilter>
23. Wireshark. Lua/Dissectors - The Wireshark Wiki. [Online]. Available: <https://wiki.wireshark.org/Lua/Dissectors>
24. R. Ierusalimschy, W. Celes, and H. d. F. Henrique de Figueiredo. (2016) Lua: about. [Online]. Available: <https://www.lua.org/about.html>
25. Hex-Rays. IDA: About. [Online]. Available: <https://www.hex-rays.com/products/ida/>
26. Carbon Black Inc. Carbon Black Inc. [Online]. Available: <https://www.carbonblack.com/>
27. gnu.org. Gnu make. [Online]. Available: <https://www.gnu.org/software/make/>

28. Xilinx. ISE Design Suite. [Online]. Available: <https://www.xilinx.com/products/design-tools/ise-design-suite.html>
29. Thales E-Security. Network Encryption Platforms: Datacryptor. [Online]. Available: <https://www.thales-esecurity.com/products-and-services/products-and-services/network-encryption-appliances>
30. Ettus, “Usrc n200/n210 networked series.” [Online]. Available: https://www.ettus.com/content/files/07495_Ettus_N200-210_DS_Flyer_HR_1.pdf
31. Minitab. Minitab Statistical Software - Minitab. [Online]. Available: <http://www.minitab.com/en-us/products/minitab/>

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) 23-03-2017			2. REPORT TYPE Master's Thesis		3. DATES COVERED (From — To) Sept 2015 — Mar 2017	
4. TITLE AND SUBTITLE Cyber Security Analysis and Strategy Development for Software-Defined Radars					5a. CONTRACT NUMBER	
					5b. GRANT NUMBER	
					5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Yerkes, Blake, E 2d Lt, USAF					5d. PROJECT NUMBER 17G310	
					5e. TASK NUMBER	
					5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENG-MS-17-M-083	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of Homeland Security ICS-CERT POC: Neil Hershfeld, DHS ICS-CERT Technical Lead ATTN: NPPD/CSC/NCSD/US-CERT Mailstop: 0635 245 Murray Lane, SW, Bldg 410, Washington, DC 20528 Email: ics-cert@dhs.gov phone: 1-877-776-7585					10. SPONSOR/MONITOR'S ACRONYM(S) DHS ICS-CERT	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. THIS MATERIAL IS DECLARED A WORK OF THE U.S. GOVERNMENT AND IS NOT SUBJECT TO COPYRIGHT PROTECTION IN THE UNITED STATES.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT This research investigates a rudimentary software-defined radar. The research presents a cyber attack and defense strategies on the test-bed architecture. The cyber attack and defense vectors are contextualized within the current state of software-defined radar systems by reviewing and summarizing various applications of software-defined radars currently in use. The system under test is open-source but representative of many software-defined radars found in current use. The operation of the system is described from the application level to the hardware operation. The system under test is found to be vulnerable to man-in-the-middle attacks. This attack vector is capable of modifying packet data to change configuration settings onboard the system front end and modifying the waveform data between the two nodes. Manipulation of the waveform data produces a predictable effect in the distance estimates produced by the radar. The results of the configuration command and waveform data manipulation are quantified and presented. The research presents strategies for securing the system under test. The research also presents tools for securing the system.						
15. SUBJECT TERMS Software-defined Radar, Cyber Security, Man-in-the-Middle						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Barry Mullins, AFIT/ENG	
U	U	U	U	109	19b. TELEPHONE NUMBER (include area code) (937) 255-3636, x4555; barry.mullins@afit.edu	