AN APPLICATION OF NETWORK SECURITY MONITORING TO THE MIL-STD-1553B DATA BUS

UNE APPLICATION DES TECHNIQUES DE SÉCURITÉ RÉSEAUTIQUE AU BUS DE DONNÉES MIL-STD-1553B

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

Charles Bernard, BEng, P.Eng.
Second Lieutenant

In Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in Computer Engineering

September, 2019

Pour Memere.

Acknowledgements

I would like to thank my supervisor, Dr. Sylvain Leblanc for his unwavering commitment in guiding me to success throughout my studies at RMC.

To my spouse Katherine, our family and our friends, thank you for your ongoing support. This work would not have been possible without your love and sacrifice (and gentle nudging).

To the professors, staff and my fellow students in the Electrical and Computer Engineering department, particularly in the Computer Security Laboratory, thank you for your support, camaraderie and encouragement throughout the years.

To my colleagues at the Aurora Incremental Modernization Project, 36 Signal Regiment and 415 Long Range Patrol Force Development Squadron, thank you for recognizing the value in this research. Your encouragement and enthusiasm for this space provided motivation to carry on.

Finally, I would like to acknowledge the invaluable contribution of the Directorate of Technical Airworthiness and Engineering Support, both for supporting this research, and offering me the opportunity and flexibility to pursue it. Special thanks to Stephen Sterling and Patrice Bélanger, and the entire team in DTAES 8.

Abstract

The objective of this thesis is to demonstrate that the application of signature-based network security monitoring techniques to the MIL-STD-1553B data bus can be used to identify signs of undesirable or otherwise abnormal system activity. Detection of such traffic is critical to ensuring that data bus-connected devices continue to operate as designed, and are free from compromise or faults.

In addition to signature-based detection two anomaly-based detection techniques were selected for implementation: word repetition analysis and RT frequency analysis. Each of these has roots in techniques used to monitor IP-based networks for signs of compromise. Software implementing each of these functionalities was written in Python, including functions to provide dissection of MIL-STD-1553B data, as well as a command line user interface.

Initial functionality testing was conducted using a commercially-available MIL-STD-1553B data bus simulator commonly used for bus prototyping. The effectiveness of the detection techniques was demonstrated against data bus attack scenarios postulated by Stan et al. Two additional cases were studied, representing scenarios where a bus-connected device could be faulty or misconfigured.

In two of six cases, a successful detection was made using signature-based analysis. In two other cases, anomaly-based detection uncovered initial signs of compromise, which were further investigated using signature-based detection. In the final two cases, signature-based detection was not effective.

While many opportunities for future work exist in further refining and automating these techniques, implementing new detection strategies, and testing against different attacks, it was ultimately demonstrated that the application of signature-based network security monitoring techniques is a viable means of detecting indicators of undesirable activity on the MIL-STD-1553B data bus.

Résumé

L'objectif de cette thèse est de démontrer que l'application des techniques de sécurité réseautique du type détection de signatures au bus de données MIL-STD-1553B peut servir à identifier la présence d'activités non-désirables ou autrement anormales dans le système. La détection de ce trafic est critique pour assurer que les composantes se servant du bus opèrent tels que conçues, sans faute ou compromis.

En plus de la détection de signatures, deux méthodes de détection d'anomalies ont été sélectionnées et mises en oeuvre : l'analyse de répétition des mots, et l'analyse de la fréquence des RT. Chacune de celles-ci sont dérivées à partir de techniques servant à surveiller les réseaux IP. Un programme a été rédigé en Python pour implémenter ces fonctions ainsi que le support requis : la décortication du data MIL-STD-1553B ainsi qu'une interface d'utilisateur basée sur la ligne de commande.

Des tests de fonctionnement ont été conduits avec l'aide d'un simulateur de bus de données MIL-STD-1553B disponible sur le marché commercial. L'efficacité des techniques de détection ont été mises à l'épreuve contre des scénarios d'attaque proposés par Stan et al. Deux cas additionnels ont été étudiés, traitant de scénarios ou un RT serait touche par une faute technique ou mal configure.

Dans deux des cas étudiés, la détection par signature a réussi à découvrir des indices claires de compromise. Dans deux autres cas, la détection primaire était faite par l'analyse à base d'anomalies, mais la détection par signature a servi d'outil d'enquête. Dans les deux derniers cas, la détection par signature s'est avérée inefficace.

En utilisant des combinaisons des trois méthodes proposées, des indices claires de compromise ou de faillite technique ont été observées dans tous les scénarios sauf un. En fin de compte, ce travail démontre que l'application des techniques de sécurité réseautique est efficace pour trouver des indices d'activité anormale ou non-désirable sur le bus de données MIL-STD-1553B.

Contents

List of Tables

List of Figures

1 Introduction

From the earliest days of powered flight, aircraft avionics have been designed based on a presumed environment of trust — that every device connected to the system belongs, that each device does only the task it was designed to do, and that the device would not deliberately produce misleading information. This notion is largely a byproduct of the careful design required to successfully implement such a complex system such as an aircraft. Historically, each avionic component was crafted to carry out a single specific function, and when two devices needed to exchange data, they were connected with a discrete wire line.

The history of networked computing followed much the same model. As computers first began to exchange data in a systemic fashion within small local area networks, similar assumptions were made: there would be no reason for a computer within the network to deliberately have an adverse effect on a peer. However, these small networks began to grow and interconnect, expanding beyond the point of being administered by a single person or organization. This lack of direct administrative control rapidly eroded the assumed trust between connected devices. The connection of even a single device controlled by a careless or malicious user would have the potential to cause havoc, with administrators powerless to stop it.

Many of the networking protocols underpinning computer communications today were developed during the period where networks were still relatively small. Replacing these with more modern protocols that consider security from the outset would require a herculean effort and a considerable amount of resources. Instead, computer and network operators have developed and deployed a number of products and procedures with the goal of adding a measure of security to the existing protocols. Programs such as firewalls, intrusion detection systems and anti-virus are today considered *de rigueur* for networked computer operations.

Avionics saw similar growth, evolving rapidly in the period between World War II and the Vietnam era. Single-function devices displaying information directly to a user began to be replaced with more complex equipment capable of sharing data with other devices as required. This data exchange can be seen as analogous to the dawn of the networked computing era.

Eventually, the web of discrete lines required to interconnect avionic devices became unsustainable. To remedy this, the United States Air Force published a standard in 1973 for a serial data bus to be used to interconnect military aircraft avionics: MIL-STD-1553 [1]. This bus would provide a single data line connecting every avionic device to a central computer. This standard, published in 1973, was first implemented on the General Dynamics F-16 Falcon and has since become commonplace in both military and civilian aircraft.

From the 1990s to the present day, there has been an increasing overlap in the worlds of avionics and computing. The ubiquitousness of communication systems in modern life and the increased availability of high-speed, low-cost data processing have led to the opening of data exchanges between the previously-isolated avionics network aboard aircraft, and the wider networked world. The Global Positioning System (GPS) and seatback entertainment displays are introducing external data into the previously sheltered aircraft system, while systems such as Automatic Dependant Surveillance - Broadcast (ADS-B) open data paths from the aircraft to the outside world. Systems such as Aircraft Communications Addressing and Reporting System (ACARS) and passenger Wi-Fi offer bidirectional communications, while general purpose computing devices such as tablet computers are increasingly pressed into service as Electronic Flight Bags (EFBs) and system maintenance tools.

While aircraft information systems are no longer isolated from external data systems, the MIL-STD-1553 data bus and its assumption of trust persist. As was the case for computer networking, a wholesale replacement of the data bus would require considerable effort and expense to re-engineer every bus-connected component currently in service. Instead, the development of security functionality that can be applied to the existing bus would be a simpler and more cost-effective option.

This thesis seeks to investigate the effectiveness of signature-based intrusion detection on the MIL-STD-1553B data bus.

1.1 Motivation

The motivation behind this thesis stems from work done to study the vulnerabilities inherent to the MIL-STD-1553B data bus. Two primary works in particular have demonstrated the need for a method to detect attempts to exploit these vulnerabilities.

The first of these is the work done by Captain Jeremy Paquet to study the vulnerabilities inherent to the MIL-STD-1553B data bus [2]. This study of the bus provided an initial insight into the MIL-STD-1553B protocol's lack of rigorous security features.

The second is the work of Stan et al., presenting a security analysis of the MIL-STD-1553B protocol, including an enumeration of assets, an attacker profile and a taxonomy of potential MIL-STD-1553B attack methods [3]. Stan et al. go on to demonstrate a sequence-based statistical detection method to identify anomalies in bus transmissions.

1.2 Aim and Scope

The aim of this thesis is to demonstrate an application of signature-based detection to MIL-STD-1553B data bus traffic to identify signs of undesirable system activity.

The undesirable activity to be detected is that which can be absolutely defined, such as the appearance of a data bus word matching specified parameters. While signature-based detection is a strong tool, it is not universally useful. Undesirable activity can also be defined in more relative terms, relying on an assessment of bus activity against a baseline. While this work focuses on the former, examples of the latter will also be studied for comparison.

While this work focuses on the MIL-STD-1553B data bus, the signature-based detection technique demonstrated could potentially be applied to other similar data buses, such as ARINC 429, STANAG 3910 or the Controller Area Network (CAN) bus.

The scope of this work is to produce a proof of concept implementing a signature-based detection routine tailored to the MIL-STD 1553 data bus. Elementary anomaly detection methods are also implemented in order to provide insight into scenarios where the signature-based detection is not suitable for purpose. Detailed implementation, including the actions to be taken upon detection, is beyond the scope of this work.

1.3 Potential Impact

The primary impact of this thesis is a method for detection of undesirable data bus activity, which could be an indication of a compromise of one or many data bus connected devices.

A potential secondary application of this work is in aircraft maintenance. With the ability to search through large amounts of data bus traffic for specific events or departures from a known baseline, maintainers could study the bus traffic for specific faults or failures.

1.4 Document Outline

The remainder of this document will be broken down as follows:

Chapter 2 will provide background information and will cover aspects of the MIL-STD-1553 data bus, the concept of network security monitoring and the current state of systems security engineering within the context of aerospace. This chapter concludes with an overview of the existing literature on the topic.

Chapter 3 will present a number of scenarios in which the MIL-STD-1553B data bus may be susceptible to abuse, and discuss methods to detect such tampering.

The design and implementation of an automated system for detecting signs of undesirable data bus activity will be discussed in Chapter 4. This will include design assumptions and constraints, algorithmic implementation of the signature detection method proposed in the previous chapter, and a summary of the development process. This chapter will conclude with a discussion of the testing done on the prototype software prior to proceeding.

Chapter 5 will evaluate the effectiveness of the proposed automated detection system against the scenarios defined in Chapter 3, followed by a discussion of the results observed.

Chapter 6 will discuss potential avenues for future work as well as make concluding remarks.

2 Background

As this work involves a combination of concepts from two distinct fields, namely the MIL-STD-1553B data bus and network security monitoring, background elements from each will be presented in this chapter. A third section discussing the concept of System Security Engineering in aviation will provide context for the importance of this work within the broader aerospace systems engineering field. This chapter will conclude with a review of the security work previously done in the data bus space and underscore the potential applications for this work.

2.1 MIL-STD-1553B Data Bus

The MIL-STD-1553 data bus standard was originally published in 1973, and later revised as MIL-STD-1553A in 1975 and again as MIL-STD-1553B in 1978. MIL-STD-1553B has also been adopted for commercial use as SAE AS-15531 [4]. While the differences between revisions are minor, this work will focus on the more recent revision B. Extensions and modifications to MIL-STD-1553B, such as FAST-1553, are outside the scope of this work, as they typically do not adhere to the MIL-STD-1553B signaling scheme. However, as discussed in section 1.2, the work presented herein is likely to be adaptable to other data bus types and signaling conventions.

2.1.1 Bus Architecture

All devices connected to a MIL-STD-1553B data bus are assigned one of three possible functions.

The role of the bus controller (BC) is to coordinate all bus activity by commanding other devices to send data to, or read data from the bus or report on their status [4]. The BC is also capable of sending its own data, or reading data from other devices for processing.

Remote terminals (RTs) are generally connected to, or embedded within, various aircraft components to enable bus communications [5]. Each is assigned a unique address, allowing the BC to direct actions to specific RTs. The concept of addressing will be discussed further in Section 2.1.2. An RT can either write the requested data to the bus to make it available to other bus-connected devices, or it can read the bus to consume data generated by other devices [4].

While a BC and one or more RTs are required for the bus to function, a bus monitor (BM) is entirely optional [4]. A BM is used to passively monitor the bus and capture data for later processing. The standard forbids a BM from writing data to the bus, and it is therefore not assigned an address [1].

The devices are interconnected using copper wiring. Two separate runs of wiring are typically used, providing redundancy in case of failure of one line. These two buses, referred to as A and B in most implementations, are independent: messages are sent over one bus or the other.

A version of the standard implemented over optical fibre exists, documented as MIL-STD-1773. The standard also includes instructions for interconnection between the two bus types [5, 6]. The standard will not be further discussed in this work as it is out of scope.

2.1.2 Signaling and Word Composition

In order to better understand the security concerns described in Chapter 3 and the implementation described in Chapter 4, some fundamental knowledge of how the MIL-STD-1553B standard passes information is required.

2.1.2.1 Signaling Characteristics

The data carried over the MIL-STD-1553B data bus is Manchester-encoded binary at a rate of 1 Mbps. These binary signals are organized into 20-bit words. Each word begins with the three-bit synchronization pattern defined by the standard, and ends with a single parity bit, leaving a 16-bit payload for the transmission of information. Transmissions requiring more than 16 bits can be sent as a series of words, which is referred to as a message [1].

Time-division multiplexing is used to structure bus communications by carving 20-bit time slots for each word. Given the data rate of 1 Mbps, this translates to a nominal 50,000 slots per second [5]. The true number of available slots will be slightly less, as a four microsecond gap is required between messages. This intermessage gap is not required, however, between words in the same message [1].

There are three types of words defined by MIL-STD-1553B, each with a specific function: command words, data words, and status words. These word types can be distinguished by the format of the synchronization pattern at the start of each. Within the context of this work, this distinction is automatically made by the hardware, as will be discussed in Chapter 3.

2.1.2.2 Command Words

Command words are used by the BC to pass instructions to RTs. Only the BC may issue command words [1].

The bit format of a command word is illustrated in Figure 2.1.

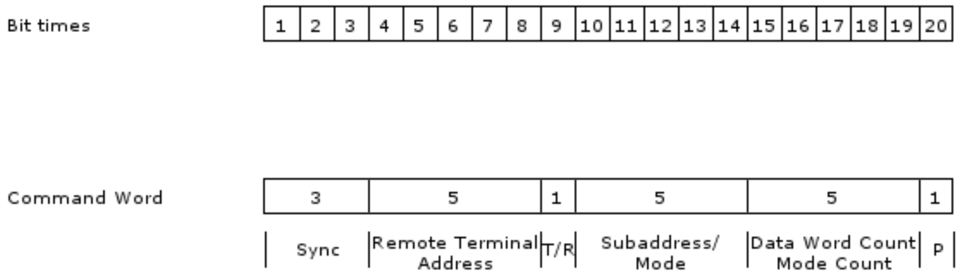


Figure 2.1: MIL-STD-1553 Command Word Format. Adapted from [1].

2.1.2.3 Data Words

Data words are used to pass data to other bus-connected devices. The standard does not define a structure for data words; all 16 payload bits are available to carry data and the formatting is left up to the designer of the RT [1]. The bus designer must take care to ensure that any device consuming data from a particular RT is able to interpret the data according to the defined scheme.

Figure 2.2 shows the breakdown of data words by bit time.

2.1.2.4 Status Words

Status words are generated by RTs to inform the BC or other RTs of the device's state. Status words are used to confirm receipt of commands, to indicate errors, or to otherwise demand the attention of the bus controller [5].

Aside from the RT address field, the status word consists almost entirely of single-bit flags. The breakdown is illustrated in Figure 2.3.

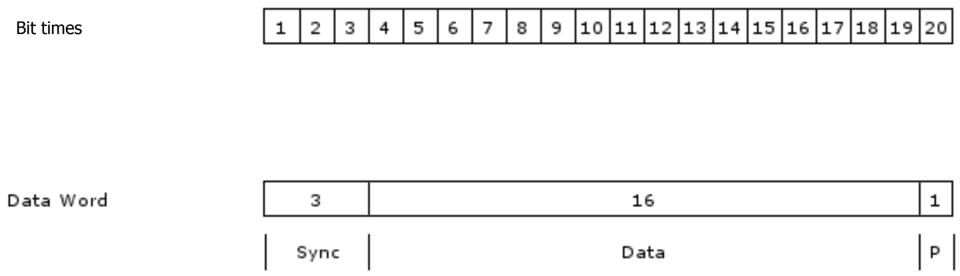


Figure 2.2: MIL-STD-1553 Data Word Format. Adapted from [1].

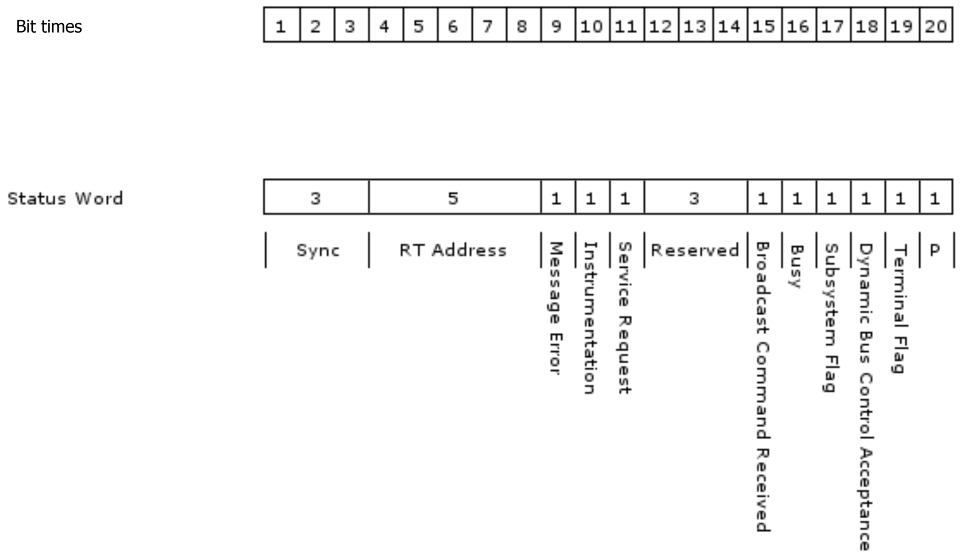


Figure 2.3: MIL-STD-1553 Status Word Format. Adapted from [1].

2.1.3 Message Formats

As mentioned previously, MIL-STD-1553B words are grouped to form messages. The bus standard uses the term “message formats” to describe the types of interactions between BCs and RTs [1]. As discussed earlier, all bus transmissions are commanded by the BC, regardless of the message source or destination.

There are four possible message formats. Three of these formats are used to govern data transfers between devices, and the fourth, the mode message, is used for data bus management.

2.1.3.1 BC-RT Data Transfer

The transfer of data from the BC to an RT is a three-step process [1, 5].

1. The BC sends a command word addressed to the RT, with the T/R bit set to "receive", and the data word count set to the number of data words to be transmitted. The targeted RT will now listen to the bus and capture the number of words indicated in the word count.
2. Immediately after sending the command word, the BC will send the specified number of data words, which will be captured by the RT.
3. Upon receiving the specified number of data words, the RT will send a status word to confirm receipt. If required, the status word will also indicate any errors by using the appropriate flags.

2.1.3.2 RT-BC Data Transfer

Data transfer from the RT to the BC is similar to the previous example, but in reverse [1, 5].

1. The BC sends a command word addressed to the RT, with the T/R bit set to "transmit", and the data word count set to the number of words it expects to receive.
2. Upon receipt of the command word, the RT will reply with a status word. This status word may be used to indicate any validation errors.
3. Immediately after sending the status word, the RT will send the number of status words requested in the original command word.

2.1.3.3 RT-RT Data Transfer

Because the BC must manage both a data transmission and data reception, RT-RT data transfers are slightly more complex than the previous cases [1, 5].

1. The BC sends a command word to the receiving RT ($\mathrm{RT_R}$), with the T/R bit set to "receive" and the data word count set to the number of words to be received.
2. The BC sends a second command word to the transmitting RT ($\mathrm{RT_T}$) with the T/R bit set to "transmit", and the data word count set to the same number of words.
3. $\mathrm{RT_T}$ replies with a status word, followed immediately by the requested number of data words. These words are captured by $\mathrm{RT_R}$, which is still waiting in receive mode.
4. When it has received all specified data words, $\mathrm{RT_R}$ will confirm receipt with a status message of its own.

2.1.3.4 Mode Messages

As alluded to in Section 2.1.2, mode messages are used by the BC to command RTs to "communicate with the multiplex data bus hardware, and to assist in the management of information flow, and not to extract data from or feed data to a functional subsystem" [1]. Examples include "Initiate Self Test", "Transmit Status Word", "Reset Remote Terminal" and "Transmitter Shutdown".

Mode messages take the form of command words with the subaddress/mode field set to either of the mode values: `0b00000` or `0b11111`. When either of these values are detected, RT will interpret the word count field as a mode code. These codes correspond to actions defined in the standard [1].

2.2 Network Security Monitoring

Network Security Monitoring (NSM) is defined as "the collection, analysis, and escalation of indications and warnings to detect and respond to intrusion" [7]. This definition has a number of practical implications: we must be able to see the traffic, understand it and make a determination as to whether or not an escalation is warranted — is the traffic benign, or is it anomalous or possibly malicious? [5].

Techniques for monitoring and interpreting MIL-STD-1553B bus traffic in order to conduct the analysis will be discussed in detail in Chapter 4; this section will provide some context for the techniques discussed therein.

There exists two broad families of analysis and detection algorithms: signature-based and anomaly-based [5]. While there have been attempts to further subdivide these into taxonomies of algorithms, there is no general consensus on a single breakdown [8, 9, 10, 11]. Rather than delving too deeply into any one taxonomy, this section will broadly present characteristics and examples within the context of the two large families.

2.2.1 Signature-based Detection

Signature-based intrusion detection is the most simplistic and intuitive detection method available. It compares the bus traffic observed to samples of traffic known to be associated with undesirable activity. Any traffic that matches the sample, known as a signature, results in an action, typically an alert to the operator or a log entry. [8]

Signature-based detection is implemented using one of two methods: string-matching and state modeling [8]. While state modeling can be a powerful tool, string-matching is more relevant to the present work.

In a string-matching implementation, the intrusion detection system (IDS) is given a list of strings of interest, and searches for them in the traffic traversing the network. This is useful in cases where the analyst has knowledge of what signs they are looking for.

Rigidly defined signatures can be used to detect traffic that exactly matches the string of interest. For network traffic that is known to contain defined constituent portions, e.g. header fields, flags, payloads, signatures can be defined parametrically. For example, a signature can be crated to raise a detection against any traffic containing a specified source addresses. If supported, the use of wildcards and regular expressions can further enhance the versatility of signatures.

Due to its relative simplicity, signature-based detection has the advantage of being easy to implement without requiring the more intensive computational resources of anomaly-based detection routines. This is particularly relevant if the detection engine is to be run in a resource-constrained environment, e.g. added to existing hardware or where power consumption is a critical design factor.

Computational simplicity means detection can be fast: high-speed data processing coupled with powerful pattern matching algorithms allow a signature-matching routine to process data on the order of tens of gigabits per second [12]. The comparatively slow 1 Mbps speed of MIL-STD-1553B can easily be analyzed.

Despite the advantages, signature-matching techniques have two major disadvantages. The first is the requirement for prior knowledge, both of the system's configuration and of the attacker's exploitation tactics. While known attack patterns are often well documented, novel attacks would not be detected. Because the focus is strictly on the traffic observed, and not the behaviour of the system as a whole, no alert will be raised if an attacker is able to compromise the system without using any of the strings in the list. This is true even if the system is showing obvious signs of compromise [5, 8].

To counter this, near-constant upkeep is required to ensure that the latest attack methods are characterized and added to the signature database. Active research into methods of compromise is also valuable in the effort to stay one step ahead of the attacker.

2.2.2 Anomaly-based Detection

While signature-based detection generally performs well in detecting known traffic characteristics, anomaly-based detection can be powerful in finding signs of compromise without the need for prior knowledge of the attacker's methodologies.

Rather than rely on a list of pre-written signatures, anomaly-based systems attempt to detect possible intrusions by determining whether or not the traffic observed is "normal" [8].

Normalcy can be defined by a number of metrics, such as overall traffic volumes, number of occurrences of a specific message, or frequency of messages addressed to or from a particular address. A wide range of metadata can be used to define a model for normal activity.

As traffic is observed, an anomaly-based IDS compares the selected parameters to the model. Deviations from the model cause the IDS to generate an alert. The IDS can be tuned by setting thresholds on the deviation, for example as a percentage above or below normal, or a number of deviations in a given time frame.

The main advantage of anomaly-based IDS is the elimination of the requirement for prior knowledge of the attacker's behaviour in order to generate a signature. Because alerts are generated based on deviations from systemic normalcy rather than detection of a specific signature, it can be possible to detect attacks not previously seen, or defend against attackers deliberately avoiding the use of strings likely to be part of the signature set.

Conversely, the traffic sample used to generate the initial model must absolutely be free of attack traffic. If malicious traffic is included in the initial model, it will not be seen as anomalous. It can be difficult to confirm that a traffic sample contains only non-malicious traffic, especially when capturing traffic from systems already in operation, where there is a risk that the system may have been compromised.

Anomaly-based IDS are also susceptible to defeat by slow-moving multi-step attacks, where each step deviates only slightly from the model. While any one of these deviations may not raise an alert, the sum total of their actions will result in malicious activity. These types of attack can be particularly insidious for IDS using machine-learning techniques, as the algorithms may actually roll attack traffic into the model used by the system, as discussed above.

2.2.3 Intrusion Prevention Systems

While IDS detect undesirable network activity and generate alerts for action by an analyst, Intrusion Prevention Systems (IPS) go one step further, attempting to automatically stop the detected intrusion [13].

Upon detection of a potential compromise through either a signature- or anomaly-based technique, an IPS will attempt to interrupt the attack. On IP-based networks, this can be done by creating a firewall rule to block the malicious traffic, or logging off the user. In some cases, an IPS can also attempt to shut down the affected system before the compromise can spread.

The deployment of an IPS introduces a large amount of risk to system availability. While false alarms on an IDS are a minor inconvenience, the consequences of a false alarm on an IPS means blocking legitimate traffic [13]. IPS are therefore only suitable for systems where there confidentiality is highly prized above availability.

Because an aircraft's MIL-STD-1553 data bus is typically mission- or flight-critical, maintaining high availability is essential. The potential availability risks introduced by an IPS-style solution are unacceptable. A IDS system conducting passive monitoring and alert generation, however, introduces minimal risks to the system and would be the preferred option for a data bus application.

2.3 Aircraft System Security Engineering

The rapid emergence of data transmission paths to and from aircraft, coupled with the increasing demand for real-time data within the aircraft, from advanced flight systems to passenger entertainment systems, have brought the topic of security of aircraft systems to prominence in recent years [5]. This section briefly summarizes the state of the field of Aircraft System Security Engineering, where the impacts of this work may ultimately be felt.

As briefly discussed in the introductory remarks in Chapter 1, aircraft systems have evolved from discrete data lines to multiplexed high-speed data buses, but the security model has remained unchanged. The most commonly cited turning point in aviation system security is the design of the Boeing 787, which sought to combine previously isolated subsystems for aircraft control, maintenance. and passenger entertainment into a single network. The concern of interconnecting a public-facing system with one carrying flight-critical data such as the data bus gave rise to concerns surrounding safety and airworthiness.

In evaluating this portion of the 787's design, the United States Federal Aviation Administration issued Special Condition (SC) 25-356-SC, to address the issue of data separation [14]. SCs are raised to address "novel or unique design features", and and are intended to "contain such safety standards for the aircraft, aircraft engine or propeller as the FAA finds necessary to establish a level of safety equivalent to that established in the regulations" [15].

This SC is notable as the first instance of information security requirements being codified. Continued development of this concept has led to the issue of standards for aeronautical systems security by the Radio Technical Commission for Aeronautics (RTCA) and European Organization for Civil Aviation Equipment (EUROCAE).

In October 2018, the United States Government Accountability Office (GAO) made public a report to the U.S. Senate Committee on Armed Services describing the state of cybersecurity as it applies to weapon systems, including ships, aircraft and land vehicles. This document provides a detailed look into the threats and vulnerabilities of platforms, and the potential effects of poor cybersecurity practice [16]. While this report provides no definitive recommendations, it does highlight a number of concerns with regards to the challenges encountered by the U.S. Department of Defense in identifying platform vulnerabilities and mitigating the associated risks.

While these efforts are certainly laudable, FAA, RTCA and EUROCAE's requirements primarily apply to new and heavily modified designs. This does not apply to the thousands of aircraft currently in service around the world, nor does it provide much guidance on the use of legacy systems such as MIL-STD-1553B devices. Although the GAO's report does provide some retrospective and hints at the importance of considering legacy systems, it does not contain recommendations or practical guidance.

With the emphasis on new aircraft and weapons systems coming online, there is a clear lack of guidance on the implementation of security features for platforms already in service, many of which contain systems designed decades ago. The ability to apply network security monitoring functionality to a critical communication pathway such as the MIL-STD-1553B data bus has the potential of going a long way towards closing this gap.

2.4 Previous Work

The concept of adapting network security monitoring techniques to an aircraft data bus is a fairly new one, and the body of directly relevant previous work is small. There are however a number of works related to the study MIL-STD-

1553B as well as to the concept of security applied to aviation systems are relevant to this space. Additionally, the study of embedded systems, particularly as applied to the automotive industry, can lend inspiration to aerospace applications. The principal works in these areas are discussed below.

2.4.1 MIL-STD-1553B

As discussed in Section 1.1, the work to study the inherent vulnerabilities of the MIL-STD-1553B data bus by Captain Jeremy Paquet is the starting point to the exploration of this research space and a primary motivation to this work [2].

Captain Blaine Losier has also undertaken efforts to implement a system to automatically detect tampered MIL-STD-1553B bus communications by measuring a number of parameters describing the bus's activity and comparing these observations to baseline data measured on the same bus in a known-good state. These parameters are RT response time, inter-message gap, data throughput, bus utilization and periodicity [17].

Losier's preliminary findings motivated the work of Lieutenant Sebastien Genereux, Lieutenant Alvin Lai and Sub-Lieutenant Craig Fowles in developing MAIDENS: a MIL-STD-1553 Anomaly-based Intrusion Detection System. MAIDENS improves on Losier's work by reducing the false alarm rate and providing more accurate information on the time of detection. This was done by optimizing the width of the bins used to sort the buffered data and adding a sliding window to the bus data parser to improve time resolution [18]. While this anomaly-based detection approach was demonstrated to be successful in detecting signs of compromise, it discounts signature-based detection.

Thuy D. Nguyen's paper on MIL-STD-1553 covert channel analysis presents a threat model and hypothetical attack scenarios where the data bus's vulnerabilities can be used to create a covert channel to surreptitiously store and extract information [19]. Nguyen identifies several data storage and transmission paths enabled by "the lack of security requirements for hosted payload space applications".

Nguyen's work was the inspiration for the work of Orly Stan et al., where they conduct a security analysis of the MIL-STD-1553B data bus. This work was discussed previously in Section 1.1 and is this work's other primary motivation.

While the objective of Stan et al.'s paper is to present a sequence-based statistical detection method to identify anomalies in bus transmissions, they also present a detailed security analysis of the MIL-STD-1553B protocol. The evaluation begins by enumerating assets and grouping them into three categories [3]:

Connectivity Assets: Transceivers, couplers, and physical bus wiring.

Data Assets: Data in transit over the bus, data stored in the memories of the transceivers and subsystem (i.e. firmware), and subsystem data (e.g. the aircraft position as derived by a GPS system).

Computational Units: The physical computing devices generating and consuming data.

For examples of each asset, Stan et al. present a list of potential consequences, should the asset suffer a loss of either confidentiality, integrity or availability. In doing so, this common triad is reframed into the narrow effect categories of data leakage, data integrity violation, and DoS attack [3].

The attack vectors are also grouped into two broad categories [3]:

Message Manipulation: Modification of legitimate command, status, and data words transmitted over the bus.

Behaviour Manipulation: Altering the behaviour of a compromised component, e.g. transmitting arbitrary messages of the attacker's choosing, including using unusual timings or sequences.

Stan et al. go on to postulate a number of attack methods, grouped by attack vector and effect, with attack vectors broken down even further into categories by asset. While this categorization process could be seen as somewhat convoluted, it does culminate in a table of threats and attack methods, showing the attack vectors for each. An adapted version of this table is shown in Table 2.1.

Table 2.1: Adapted from the Threats and Attack Categorization proposed by Stan et al.

	DoS Attack	Data Leakage	Data Integrity Violation
Command Word	2 (Message and Behaviour)	2 (Message and Behaviour)	1 (Message)
Status Word	1 (Message)	2 (Message and Behaviour)	1 (Message)
Data Word	1 (Message)	1 (Message)	1 (Message)
Transmission Timings	1 (Behaviour)	1 (Behaviour)	0 (None Reported)
BM Impersonation	0 (None Reported)	1 (Behaviour)	0 (None Reported)
TEMPEST	0 (None Reported)	1 (Behaviour)	0 (None Reported)

Table 2.1 summarizes the possible attacks proposed by Stan et al. For example, there are two proposed DoS attacks using command words: one using message manipulation and one using behaviour modification. Of the sixteen attacks identified, four will be revisited in detail in Chapter 3 and further developed into scenarios to validate the detection system implementation described in Chapter 4.

Stan et. al. go on to propose and demonstrate a statistically driven anomaly-based detection method, applying Markov models to predict changes in the state of the bus [3]. As with MAIDENS, this approach is successful, but does not consider the utility of a signature-based approach.

2.4.2 Other Aviation Buses and Systems

The concept of applying network security monitoring techniques to an aircraft data bus has been postulated before by the team of Nagaraja Thantry and Ravi Pendse. However, this work is focused on ARINC 664 data bus standard, which effectively an implementation of the IEEE 802.3 Ethernet standard. While Thantry and Pendse do make a compelling argument in favour of applying security controls to the aircraft data bus, the attacker profile, bus vulnerabilities and mitigations proposed are specific to a data bus implementation based on TCP/IP over Ethernet. The paper also proposes a network monitoring solution, but one informed by additional data sources, such as the cockpit voice recorder or other strategically-placed sensors [20]. While the specific technologies proposed in that work may not be applicable to MIL-STD-1553B, the arguments and conclusions made do further illustrate the relevance of information security applications to data buses.

2.4.3 Automotive Systems

Looking beyond the context of aviation, parallels can be drawn with other embedded systems, particularly those related to transportation. Some of the most compelling work done in this space has been in the automotive sector.

The Controller Area Network (CAN) bus is present on nearly every modern passenger vehicle and serves as the primary data bus. Much like MIL-STD-1553B, it has no native security features, assuming instead that all messages on the bus are legitimate.

A frequently cited example of automotive hacking is the work done by Charlie Miller and Chris Valasek, which gained mainstream attention when it was presented in Wired Magazine in July 2015 [21]. By exploiting an exposed mobile data connection, they were ultimately able write data to the vehicle's CAN bus. This gave them remote control of nearly every electronic device in the car, from windshield wipers and climate control to throttle position and braking. Miller and Valasek replicated their initial work a year later, this time using a compromised bus-connected device to send spoofed commands [22].

Similar work by Julien Savoie takes this a step further, discussing the potential for attacks specifically intending to damage the vehicle by exceeding the design limits of components, or to injure the vehicle occupants by tampering with throttle, brake and steering inputs [23].

On the defensive side, the work of Adrian Taylor et al. more formally characterizes attacks against the CAN bus into a cyberattack framework [24]. This framework was devised in order to provide a structure for attack simulation, which enables comprehensive development and testing of a CAN bus-specific IDS. This IDS uses recurrent neural networks and multivariate Markov chains to identify malicious traffic.

While the present work does not specifically consider the CAN Bus, these attacks underline the potential gravity of data bus compromise. It also presents a potential alternate application for this work in the future.

3 MIL-STD-1553B Security Concerns and Detection Methods

This chapter will first outline six scenarios affecting the security of the MIL-STD-1553B data bus: four derived from the work of Stan et al. [3], and two developed by the author [25]. A brief overview of each scenario will be given accompanied by a discussion of the potential impact on the MIL-STD-1553B data bus.

The second portion of this chapter will propose three methods for detecting signs of compromises flowing from these scenarios. The chapter will conclude by revisiting each of the scenarios and illustrating how they can be detected using one or many of the proposed detection methods.

While this chapter proposes methods for detecting possible attacks, the design and implementation of these into an automated system will be discussed in Chapter 4.

3.1 Scenarios

As discussed in Chapter 2, Stan et al. have conducted a security analysis of the MIL-STD-1553B communication protocol [3]. Their work includes asset enumeration, attacker profiling, identification of attack vectors and a list of potential attack methods.

In their paper, Stan et al. classify their proposed attack methods into three broad categories: Denial of Service (DoS), Data Leakage, and Data Integrity Violation. These attacks are further subdivided by attack vector: Message Manipulation and Behaviour Manipulation [3]. This work was discussed in detail in Section 2.4 of this document.

Message manipulation attacks are predicated on the attacker's ability to modify values in the MIL-STD-1553B words sent by a device. This implies that the attacker must compromise a bus-connected device. Taking control of a BC allows the attacker to modify command words, while compromising an RT would primarily affect data or status words [3].

Behaviour manipulation attacks leverage features of the bus's design in order to cause an adverse effect. This includes the generation of false command words, as opposed to the modification of otherwise legitimate command words seen in message manipulation attacks. An attacker could also adjust message response times to cause word collisions on the bus, or exploit covert data transmission channels [3].

Of the sixteen attack methods proposed by Stan et al., we selected a number of these to support the development of an automated detection method. While the paper contains a brief conceptual description of each attack, no comparisons are made between them [3]. We used three factors to categorize these attacks: criticality, feasibility, and diversity in vectors and effect.

Criticality is the impact of the attack on the bus, and ultimately the system. While it would be difficult to definitively rank these without thorough testing, the attack description gives a reasonable first-order approximation of the impact on the safe operation of the aircraft. The objective was to select attacks with high criticality to ensure the detection system is effective against potent threats.

Feasibility refers to the ease with which an attacker could implement the attack. This was evaluated by estimating the level of effort required to replicate the attack in a laboratory setting using commercial MIL-STD-1553B simulation equipment. Attacks that do not require particularly deep technical knowledge to implement, or that do not rely on a precise set of conditions are more versatile. Feasible attacks would likely be an adversary's tools of first choice. While Stan et al. do demonstrate only three attacks in their paper (two DoS attacks and one spoofing attack), little is offered in the way of implementation guidance [3]. Instead, the implementation of previous MIL-STD-1553 work was used as a reference point [2].

Finally, diversity was used as a selection factor in order to demonstrate the effectiveness of the detection system against a variety of attack types, not only variants of a single type. The three scenarios implemented by Stan et al. were also set aside, in favour of selecting scenarios not previously implemented [3].

Ultimately, four of the scenarios were identified for further development and study, one of which has two distinct variants:

- Transmission Timings Denial of Service (Behaviour Manipulation):
  - Command word flooding variant;
  - Status word flooding variant;
- Status Word Data Integrity (Message Manipulation);
- Command Word Denial of Service (Behaviour Manipulation); and
- Status Word Data Leakage (Message Manipulation);

In addition, two other scenarios were developed by the author to reflect situations where anomalous bus activity could be the result of a configuration error or a maintenance issue; such anomalous behaviour was not considered by Stan et al. They are:

- Malfunctioning RT; and
- Unassigned RT Address.

3.1.1 Transmission Timings Denial of Service (Behaviour Manipulation)

Stan et al. propose that an attacker with the ability to transmit words over the bus can time transmissions to collide with legitimate traffic, and that these collisions may lead to errors or degraded performance of bus-connected components [3].

The proposed attack can be taken one step further by sending a continuous stream of words, thereby flooding the bus. Such an attack would certainly be expected to cause collisions with legitimate traffic, but it would also fill any otherwise available time slot, restricting the opportunities for legitimate traffic to pass, eventually leading to data starvation.

This work will examine flooding scenarios involving both command and status words. Stan et al. make no distinction between the types of words that can be used to cause flooding, but this decision does have knock-on effects [3].

Status words are used to confirm the receipt of a command word and to indicate errors to the BC. Depending which flags are set, the status words used to flood the bus could cause the BC to take an unnecessary action such as repeating the previous command. While this may not appear to be detrimental, the repeated messages do consume additional bus time slots and further exacerbate the congestion.

Command words are used to order actions from RTs. Because of this, flooding with arbitrary command words would not only congest the bus, but could also cause RTs to take unexpected action. If an arbitrarily-formatted command causes a critical RT to consume data at the wrong time, or if it causes the RT to enter an undesired mode, the effect could be catastrophic. Moreover, most command words will cause the receiving RT to return some amount of traffic, either data or a status word. This return traffic adds to the congestion, amplifying the attack.

Due to the arbitrary formatting of data words, the data word flooding scenario has not been studied further. While this could be examined in future work, it is expected that data word flooding would simply congest the bus, while occasionally providing erroneous information to any RTs who might be consuming the data words.

3.1.1.1 Criticality and Feasibility

The criticality of this attack is sufficiently high to warrant further examination. The dual-faceted effects of data starvation as a direct result of the denial of service as well as the potential execution of arbitrary commands make this case particularly interesting.

In terms of feasibility, an attacker would simply need to introduce a device to the bus (or compromise an existing one) and have it continuously transmit command or status words. These words can be sent at any time and do not depend on timing or require a particular bus state.

3.1.1.2 Attack Indicators

Detection of a transmission timings DoS is complicated by the fact it can be carried out using any word type, at a rate selectable by the attacker, and that it does not necessarily require the use of legal words, i.e. words in compliance with the standard. Therefore, any proposed detection method must have the flexibility to cope with this [25].

Let us first consider the case where the bus is flooded by repetition of a single word. The repeated appearance of identical command or status words in a short time frame could be an indicator of anomaly. Tracking the number of times any single word appears on the bus over a given period and raising an alert when it exceeds a given threshold could detect such an attack [25].

Additionally, if the composition of the word used in the flood is known from intelligence or is discovered in the course of analysis, monitoring the bus and raising an alert if this word is seen would be desirable [25]. This approach, however, may only be feasible in cases where all flooding words are identical.

If the flooding is caused by multiple different words, either randomly generated or following a prescribed pattern, the appearance of large volumes of traffic containing unassigned RT addresses could also be an indicator [25].

3.1.2 Status Word Data Integrity (Message Manipulation)

In this scenario, Stan et al. posit that "any threat agent connected to the bus [with BC or RT capabilities] can corrupt status words transmitted back to the real BC and send fake statuses as if it is the transmitting RT" [3].

In normal bus operations, the BC gathers data by sending a command word to a specific RT. For most commands, the RT will reply with a status word to confirm receipt of the command and return any requested data. If this status word is corrupted by an attacker and followed by a spoofed status word, the BC would accept the latter as being legitimate. While Stan et al. do not further discuss the ramifications of this, it is reasonable to assume that this false status word can be crafted to show errors, which could then cause the BC to assume communication has failed and drop any received data.

3.1.2.1 Criticality and Feasibility

The criticality of this attack depends entirely on which RT's communication is being disrupted and the frequency of the disruption. However, because it can be used to target critical RTs for an indefinite period, this method merits further investigation.

While the attacker would be required to monitor the bus for a target status word to disrupt and spoof, this attack is still very much feasible using standard bus-connected equipment.

3.1.2.2 Attack Indicators

The hallmark of this scenario is the appearance of two status words sent in response to a command: one from the RT, and a second one sent by the attacker as a fake, after corrupting the legitimate status word [3].

For the attack proposed by Stan et al. to be effective, the false status word would be expected to appear soon after the legitimate status word. Monitoring the bus in search of this double status word, the real one and the falsely generated one, could successfully detect such an attack [25]. It should be noted that this method might be prone to false alarms if the RT targeted typically generates a relatively large number of status words in a short period of time.

A second possible detection method is a statistical analysis of the frequency of RT addresses seen on the bus. The difference between the number of command and status words containing the targeted RT address would be expected to deviate from the traffic pattern observed in the known-good baseline traffic [25].

3.1.3 Command Word Denial of Service (Behaviour Manipulation)

The scenario postulated by Stan et al. suggests that bus communications can be disrupted by issuing a “fake command” [3]. While a number of possible commands are suggested, shut-down commands are particularly powerful.

Transmitter shutdown commands are implemented as mode messages, which can be addressed to any RT by the BC. Upon receipt of the command, the RT will simply stop transmitting. For an attacker with bus access, this attack is both simple, requiring only one command word, and precise, as it is directed at a specific RT. Depending on which system is targeted, the impact of such an attack can range from benign to catastrophic.

The command word flooding discussed in Section 3.1.1 can also be seen as a form of Command Word Denial of Service attack through Behaviour Modification. If the command words used to flood the bus contain an assigned RT address, the RT will attempt to react as if the message was legitimate. This may have detrimental effects on the bus’s operation, including the shut-down command example above [3]. If the command word is not properly formatted, the RT will attempt to send a status word with an error flag, consuming yet another time slot and adding to the flooding.

As command word flooding was examined as part of the Transmission Timings scenario presented in Section 3.1.1, this scenario will focus on targeted attacks using the shut-down command directed to a specific RT.

3.1.3.1 Criticality and Feasibility

Of all the attacks proposed by Stan et al., this attack is perhaps the most dangerous. The ability to specifically target individual RT addresses or subaddresses for shutdown at the time of an attacker’s choosing is a very powerful tool. In addition, the RT’s transmitter function can be restored by simply issuing a second mode message, offering some measure of cover to the attacker.

This is particularly concerning when one considers its sheer simplicity: a single command word is all that is required to potentially shut down a critical system, with absolutely no dependencies on the state of the bus or need to consider timing. For an attacker assumed to already have a foothold on the bus, this is as easy as it gets.

3.1.3.2 Attack Indicators

The obvious sign of this particular implementation of the command word DoS scenario is the appearance of the transmitter shutdown command targeting a particular RT, as suggested by Stan et al. [3]. The shutdown command consists of a command word formatted as a mode message, with the corresponding mode code. The attacker may also choose to send a second mode message to re-enable the targeted RT's transmitter and end the DoS [25].

Transmitter shutdown and start-up commands are rarely seen. The intended use of these commands is to shut down an RT that is malfunctioning or damaged and producing unintelligible data. This is expected to be an uncommon occurrence in normal bus operations.

By monitoring the bus for the appearance of transmitter shutdown and start-up commands, it would be possible to detect this attack. By detecting both shutdown and start up, an analyst would be able to identify a specific period during which communications from the targeted RT were impaired.

Even in the case where the commands are the result of normal bus operations and rather than an attack, transmitter shutdown commands are significant enough that an analyst should be informed of their occurrence. They may be the only indication that a component has malfunctioned and requires a maintenance action. Therefore, no distinction will be made between malicious traffic from benign traffic in this scenario. The study of this distinction may be an avenue for future work.

3.1.4 Status Word Data Leakage (Message Manipulation)

As described in Chapter 2, the status word bitfield contains three bits reserved for use in future revisions of MIL-STD-1553. Until then, MIL-STD-1553B dictates that these bits must be set to zero in order for the status word to be valid [1]. Stan et al. suggest that these three bits can be used to leak data surreptitiously between "cooperating threat agents" [3].

Interestingly, Stan et al. also highlight "a lack of status word monitoring" as an enabler for this attack [3]. While status words with non-zero reserved bits are not compliant with the standard and should be rejected, this may not necessarily be enforced in the design of the BC. Acceptance of these words as valid would allow bus operations to continue uninhibited, without indication of data leakage.

3.1.4.1 Criticality and Feasibility

In terms of criticality, this method gives the attacker a comparatively high bandwidth channel (3 of 16 bits per status word) to pass arbitrary data. Moreover, because the data contained reserved field is routinely discarded by bus-connected devices, there is not likely to be a disruption to regular bus traffic or an obvious effect on the system. The existence of such a covert communication channel is a serious threat to confidentiality and also provides the attacker a dedicated communication path between bus-connected devices.

This attack does require some specialized knowledge to implement, but is by no means beyond the capacities of a dedicated adversary. Presuming the attacker has control of an RT, some reprogramming would be required to allow setting the reserved bits before the word is transmitted. This could also be done as part of a supply chain attack, building such capabilities in to an RT before it is installed. Also, because no monitoring of the reserved field is done, this routine need not be especially elegant in order to succeed.

Diversity was also considered in selecting this attack. While the other attacks selected from Stan et al.'s list focus on denial of service or data integrity, this one provides a covert path for the attacker to pass or extract data. Because the reserved bits are not typically processed by MIL-STD-1553 devices, there is not likely to be an obvious indication that something is awry, making detection even more critical.

3.1.4.2 Attack Indicators

This scenario will assume that the BC accepts status words that are not compliant to the standard without raising an alert of its own.

An examination of all status words traversing the bus to confirm that the reserved bits are indeed set to zero would make it possible to generate an alert for words where this is not the case. Successfully doing so would eliminate a key enabler to this scenario, namely the "lack of status word monitoring" mentioned by Stan et al. [3].

3.1.5 Malfunctioning RT

While the scenarios proposed by Stan et al. highlight potential vulnerabilities in the MIL-STD-1553B protocol that can be exploited to carry out a deliberate attack, malicious actors are not the sole source of threats. Accidental or negligent actions by well-meaning individuals such as technicians or operators may also cause undesirable bus activity.

In the malfunctioning RT scenario, a notional bus-connected device is stricken by a fault. This error causes the device to transmit words either more or less frequently than it would ordinarily be expected to under similar conditions.

3.1.5.1 Malfunction Indicators

In order to detect a malfunctioning RT exhibiting this behaviour, its traffic volume must be compared to traffic observed on the bus prior to the suspected malfunction.

The number of command and status words containing the RT's address as a ratio of overall bus traffic can be calculated for the baseline and compared to the ratio present in a captured traffic sample. A significant deviation in the ratios of the two samples may be indicative of a malfunctioning RT, producing either more or fewer words than ordinarily expected [25].

A change in the RT's response rate, i.e. command words without a corresponding status word, may also be considered indicative of anomalous behaviour [25].

3.1.6 Unassigned RT Address

The misconfiguration of a bus-connected device is a second example of potential bus disruption from an accidental threat source.

This scenario proposes a situation where a bus-connected device is observed utilizing an RT address that is not assigned in the design of the bus. This could be the result of an RT configuration error, of anomalous BC behavior, or of a device surreptitiously added to the bus in order to gain access, as postulated by Stan et al. [3].

3.1.6.1 Malfunction Indicators

The detection of an unassigned RT can be made by observing all bus traffic and capturing the RT address field of all command and status words. This list can be compared to one yielded by the same bus in a known-good state under similar conditions. Addresses appearing in the traffic sample but not in the baseline state would be suspect [25].

If the analyst has a list of the assigned RT addresses, the task is simplified. Rather than tracking each RT address seen, an observer can simply raise an alert if a word is seen bearing an RT address not on the known list [25]

3.2 Detection Methods

In reviewing the attack and malfunction indicators listed for each of the scenarios presented in Section 3.1, it becomes clear that signature detection is not useful in all cases. A number of attack scenarios are more likely to be detectable using anomaly-based methods.

In addition to a signature-based detection routine, two rudimentary anomaly-based detection algorithms were implemented as part of this work. These are Word Repetition Analysis and RT Frequency Analysis. If a detection is made using either of these techniques, it can be assumed that richer implementations such as MAIDENS would also be successful.

Further, although signature-based IDS may not able to make an initial detection, anomaly-based techniques may provide information that can then be used to guide the creation of a signature for secondary analysis.

This section will present a description of each of these three detection methods, with the objective of implementing them as part of the automated detection system described in Chapter 4. These functionalities are:

3.2.1 Signature-based Detection

Recall from Chapter 2 that signature-based detection involves comparing network traffic to a list of signatures, or traffic patterns known to be indicative of undesirable or malicious activity.

In the modern signature-based IDS used on IP-based networks, these signatures are defined parametrically. Parameters for an IP packet could include, for example, source and destination IP addresses, source and destination ports, various flags implemented in the protocol, and the actual payload content. An example of such an implementation is the ruleset of the popular Snort IDS, where users define parameters in an ASCII text file, along with the action to be taken if a detection is made [26].

In order to apply signature detection to MIL-STD-1553B data bus traffic, a similar taxonomy of parameters must be defined. Fortunately, as discussed in Section 2.1.2, the format of MIL-STD-1553B words is codified by the standard. While command, status and data words have different layouts, the order and size of these parameters is consistent within each word type.

From these defined field contents, it is possible to define parameters, similar to those used for IP-based IDS, that can then be used to define signatures.

3.2.1.1 Command Words

The parametric breakdown of command words is shown in Table 3.1. The remarks column highlights any special values used to indicate alternate functions.

Table 3.1: MIL-STD-1553B Command Word Fields

Name	Bits	Remarks
RT Address	5	`0b11111` is the broadcast address.
T/R Bit	1	`0b1` indicates transmit, `0b0` indicates receive.
Subaddress	5	A value of `0b00000` or `0b11111` indicates that the command word is a mode message.
Word Count	5	If the mode function is used in the Subaddress field, the Word Count field will contain a mode code.

3.2.1.2 Data Words

As outlined in Section 2.1.2, the structure of data words is not codified in the standard and is left to the discretion of the equipment manufacturer. While it may be possible to parametrize the contents of a given data word implementation in order to create signatures, this must be done on a case-by-case basis.

Without a defined parametric structure, signatures can instead be created at the level of individual bits.

3.2.1.3 Status Words

Table 3.2 lists the fields contained in a MIL-STD-1553B status word, along with their lengths. All one-bit flags are set to `0b0` by default, or `0b1` if the associated condition is met. Any special considerations are noted in the remarks column.

Table 3.2: MIL-STD-1553B Status Word Fields

Name	Bits	Remarks
RT Address	5	
Message Error	1	
Instrumentation	1	
Service Request	1	
Reserved	3	Always set to `0b000`.
Broadcast Command Received	1	
Busy	1	
Subsystem Flag	1	
Dynamic Bus Control Acceptance	1	
Terminal Flag	1	

3.2.1.4 Metadata

In addition to the information contained in the words, there are a number of other parameters that can be used to define signatures. This is typically information about the word itself, which we will refer to as metadata.

Metadata can be collected by quantifying various aspects of the message other than the message content itself. For example, gathering data on message length will give an approximation of the amount of data a given RT normally is normally asked to produce or consume. Collecting information about whether any RTs are using the backup B bus to communicate can also shed light on the system's operation.

There are also multiple parameters relating to timing that can provide useful information. Intermessage gap gives an approximation of bus usage, with long gaps suggesting unused capacity. This can be used to determine whether periods of high or low bus activity are cyclical or spurious. Because the bus's bitrate is a known constant, transmission length can be used to approximate the number of words sent in a message without the need to monitor the bus clock or to count bits[1].

Some commercial data bus monitors will track and report on a number of metadata parameters, enabling signatures based on metadata. Metadata can also be generated through post-processing of recorded data, potentially

[1]Disregarding collisions, where two words attempt to occupy the same time slot.

leading to richer data analysis than may be possible in near-real time aboard an aircraft.

The use of metadata-based signature parameters will be further discussed in Chapter 4.

3.2.2 Word Repetition Analysis

In typical bus operations, it would be rare to see two identical command or status words appear sequentially. Other words may not be expected more than a certain number of times within a defined time span. However, some of the scenarios given in Section 3.1 depend on an attacker impersonating a device by sending status words close in time to the legitimate ones. By monitoring the bus for such occurrences, it may be possible to reliably detect an attack.

By maintaining a running list of all command and status words seen on the bus, each subsequent command or status word observed can be compared to the list to determine whether it has been seen previously.

This technique may be susceptible to false alarms, primarily from bus operations depending on scheduled words. In such cases, it would not be unusual to observe recurring data transfers. This risk can be reduced by selecting a reasonable threshold for how recently a word has been seen: a word last seen 50 words ago may be normal while one last seen only five words ago may be suspicious. The selection of this threshold will depend on the configuration data bus being observed.

Limiting the length of the word list also has the benefit of eliminating performance issues in analyzing traffic over long periods of time. As more data is observed and the list grows longer, it becomes increasingly computationally expensive to compare new incoming words to the entire list. By setting a threshold, the oldest word can be dropped when the most recent one is added to the list, keeping the list at a set length.

Another parameter that can be adjusted to reduce the false alarm rate is the number of matches required in order to raise an alert. For example, if two identical words are indeed expected to be seen within the defined threshold, the number of required matches that will raise an alarm can be set to three.

By adjusting both the length of the word list and number of matches required to generate an alert, the analyst is able to fine-tune this detection system to match the bus under observation. The selection of matches required and list limit can be informed by the analyst's knowledge of the bus, or simply by trial and error against a data set known not to contain attack traffic.

While this technique is expected to be effective against command and status words, data words have been excluded from this discussion. Because the structure of data words is at the sole discretion of each device's designer, the frequency of appearance of a given data word does not necessarily imply anomalous activity. However, with knowledge of the data communications scheme of each device of the bus, this analysis technique may be useful in some cases.

3.2.3 RT Address Frequency Analysis

A number of the attack scenarios presented in Section 3.1 are expected to be detectable by analyzing the frequency with which certain RT addresses appear in traffic samples. Because RT addresses are contained in both command and status words, their aggregation can provide insight into the bus's operation.

By creating a list of every RT address observed in all command words sent over the bus, and noting their frequency of use, the analyst can build a picture of which devices the BC calls upon, and how frequently each is queried relative to the others. The same is true for status words, providing insight into which RTs are most responsive. Tallies for command and status words can also be compared to determine how many command words for each RT go unanswered by a status word.

This frequency data could also potentially be compared across multiple traffic samples. In the case of a bus where data transfers occur on a periodic basis according to a defined schedule, the traffic proportions for each RT would be expected to be relatively stable.

A deviation from the established baseline could imply a change to the system. If this change is unexpected, it could be indicative of a defective device or of an attack. In either case, further investigation would certainly be warranted.

A potential disadvantage of this technique is the requirement for configuration stability. If the bus's configuration has been altered in any way between traffic captures, the traffic ratios for each RT may differ, invalidating comparisons between the two samples. Aperiodic data transfers may also introduce some amount of variation if some legitimate bus actions appear in one data set but not the other. A common source of these are crew actions, such as lowering the landing gear or manually querying a sensor.

Careful consideration of external factors is also paramount, as both traffic samples must reflect a comparable aircraft state. For example, a histogram of words per device sampled during an active electronic warfare engagement would be expected to show increased traffic to the RTs governing the sensors than would be seen during routine cruise.

3.3 Conclusion

This section identified bus attack scenarios and proposed methods for detecting these.

Section 3.1 identified six scenarios where vulnerabilities in the MIL-STD-1553B protocol could have adverse effects. Four of these were direct reflections of the work of Stan et al., while the remaining two were devised by the author to reflect general bus malfunctions.

Building on these scenarios, Section 3.2 proposed three detection methods adapted from the broader NSM context. While the focus of this work is on signature-based detection, two anomaly-based methods were proposed for scenarios where signature-based detection may not be useful.

Chapter 4 will discuss how the detection methods discussed in Section 3.2 were implemented in software. The scenarios proposed in Section 3.1 will be revisited in Chapter 5, where the software implementation will be validated against each one.

4 Automated Detection System Design, Implementation and Testing

Starting from the scenarios and detection methods outlined in Chapter 3, this chapter will discuss the design and implementation of a prototype automated detection system, to be called Otto.

Recall that Section 3.1 introduced six scenarios where the operation of the MIL-STD-1553B data bus could be impaired, either accidentally or by deliberate means. From these scenarios, three detection methods were identified and presented in Section 3.2. This chapter will document the implementation of these three detection methods into an semi-autonomous detection system, capable of analyzing bus data and alerting an analyst to the presence of traffic that may be indicative of undesirable activity.

In the field of IP-based networking, such automated detection systems are typically referred to as Intrusion Detection Systems, or IDS, as discussed in Section 2.2. While the creation of a fully featured IDS is beyond the scope of this work, this chapter will describe a proof of concept to demonstrate the feasibility of the detection methods proposed in Chapter 3.

A number of initial design considerations will be presented, including methods for collecting bus data for analysis. These will give way to a high-level functional overview of Otto, followed by detailed discussions of each of the program's key components. The chapter will close with a discussion of pathways for future development to add features to the prototype and increase its robustness.

4.1 Design Considerations

4.1.1 Real-Time Detection vs Post-Mission Analysis

This work will consider a post-mission analysis construct, where recorded data bus traffic is replayed and searched for indicators of compromise. This decision enabled the independent development of the proof of concept implementation, without reliance on physical access to an aircraft or a high-fidelity data bus simulator.

Post-mission analysis is also a more likely use case. Data is routinely captured on missions for later processing. There is little advantage to in-flight, as the operator's immediate action options are limited, and such actions may be to the detriment of the overall mission.

4.1.2 Data Source

The first consideration in designing the detection system is the source of the data to be examined. There are a number of data bus monitors on the market, each with features and capabilities beyond the basic recording of raw data.

The ENET2-1553, manufactured by Alta Data Technologies, is a commercially available MIL-STD-1553 data bus monitor. The feature that sets the ENET2-1553 apart is its ability to translate MIL-STD-1553B traffic to Ethernet packets. While the primary intended use of this device is to add MIL-STD-1553B communication capability to modern computing devices using Ethernet, it also includes a fairly robust bus monitor mode [27].

In brief, the ENET2-1553 in bus monitor mode observes the bus and listens for the types of bus interactions described in Section 2.1.3, such as data transfers or mode messages. It then generates a representation of each interaction in its entirety, adds other potentially relevant metadata about the interaction, wraps this payload in a UDP/IP packet and sends the packet over Ethernet to a user-specified IP address and port. The end user can then process this UDP stream to recover the words from the bus as well as the metadata [28, 29].

4.1.2.1 UDP Packet Construction

As mentioned in the previous section, the ENET2-1553 constructs a UDP packet from the data it gathers from each bus interaction. The payload of this packet is formatted using the Alta Passive Monitor Protocol (APMP), whose structure is illustrated in Figure 4.1 [28].

The APMP adds a header containing a sequence number, a server status word, a field containing the ASCII representation of the word "ALTA" to

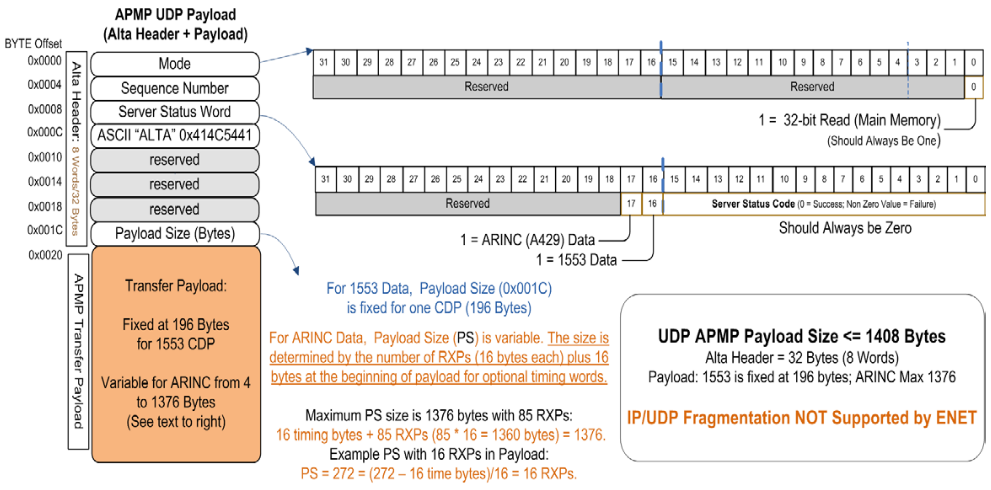

APMP UDP Payload
(Alta Header + Payload)
BYTE Offset
0x0000 Mode
0x0004 Sequence Number
0x0008 Server Status Word
0x000C ASCII "ALTA" 0x414C5441
0x0010 reserved
0x0014 reserved
0x0018 reserved
0x001C Payload Size (Bytes)
0x0020
Alta Header: 8 Words/32 Bytes
APMP Transfer Payload
Transfer Payload:
Fixed at 196 Bytes for 1553 CDP
Variable for ARINC from 4 to 1376 Bytes (See text to right)
Reserved
1 = 32-bit Read (Main Memory) (Should Always Be One)
Server Status Code (0 = Success; Non Zero Value = Failure)
Should Always be Zero
1 = ARINC (A429) Data
1 = 1553 Data
For 1553 Data, Payload Size (0x001C) is fixed for one CDP (196 Bytes)
For ARINC Data, Payload Size (PS) is variable. The size is determined by the number of RXPs (16 bytes each) plus 16 bytes at the beginning of payload for optional timing words.
Maximum PS size is 1376 bytes with 85 RXPs:
16 timing bytes + 85 RXPs (85 * 16 = 1360 bytes) = 1376.
Example PS with 16 RXPs in Payload:
PS = 272 = (272 – 16 time bytes)/16 = 16 RXPs.
UDP APMP Payload Size <= 1408 Bytes
Alta Header = 32 Bytes (8 Words)
Payload: 1553 is fixed at 196 bytes; ARINC Max 1376
IP/UDP Fragmentation NOT Supported by ENET

Figure 4.1: Alta Passive Monitor Protocol Packet Format. Reproduced from [28].

be used as a data integrity check, and payload size (fixed at 196 bytes for MIL-STD-1553 data) [28]. While the remainder of the APMP header largely consists of reserved fields, it also provides confirmation that the payload contains MIL-STD-1553 data, as this protocol is also used to packetize ARINC 429 data.

At the end of the APMP packet lies a 196 byte payload structured according to a second protocol known as the "APMP 1553 Common Data Packet", or CDP. The format of CDP is shown in Figure 4.2 [29].

The CDP is segmented into 32-bit fields containing different information, but the ones of interest in this case are the CDP 1553 Words. Each CDP contains up to 36 CDP 1553 words: two command words, two status words and 32 data words. The requirement for multiple command and status words is driven by RT-RT data transfers. Recall from Section 2.1.3 that such transfers require two command words: one to order the receiving RT to listen, and one to order the sending RT to send. Each RT will also generate a status word, requiring the CDP to contain two status words. Unused CDP 1553 words are assigned a value of `0xFFFFFFFF`.

The first 16 bits of the CDP 1553 word contain the 16-bit representation of the word observed on the MIL-STD-1553B data bus. The following 16 bits contain assorted metadata: gap time value, flags for errors observed by the ENET2-1553, and an A/B bus flag [29]. This latter flag is useful for

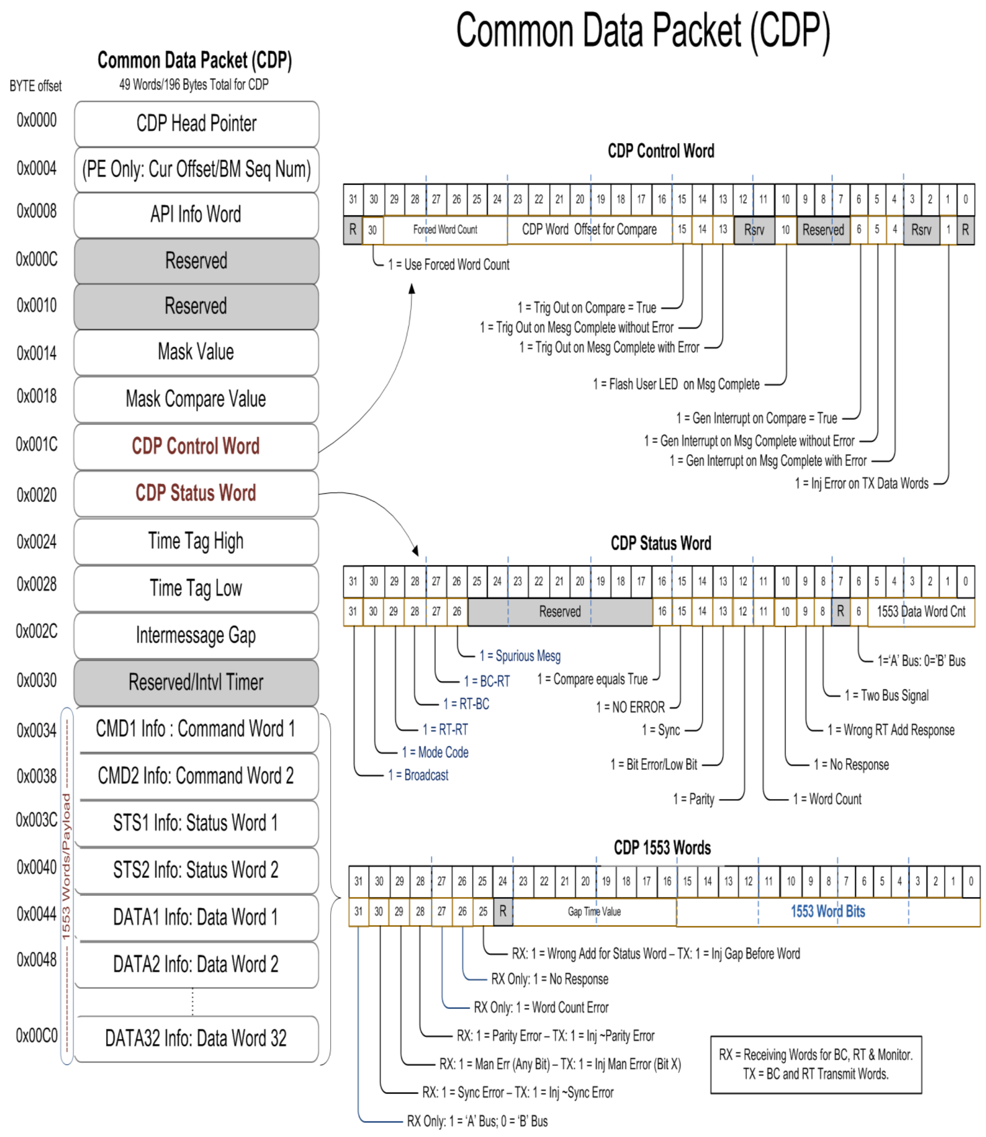

Common Data Packet (CDP)
Common Data Packet (CDP)
49 Words/196 Bytes Total for CDP
BYTE offset
0x0000 CDP Head Pointer
0x0004 (PE Only: Cur Offset/BM Seq Num)
0x0008 API Info Word
0x000C Reserved
0x0010 Reserved
0x0014 Mask Value
0x0018 Mask Compare Value
0x001C CDP Control Word
0x0020 CDP Status Word
0x0024 Time Tag High
0x0028 Time Tag Low
0x002C Intermessage Gap
0x0030 Reserved/Intvl Timer
0x0034 CMD1 Info : Command Word 1
0x0038 CMD2 Info: Command Word 2
0x003C STS1 Info: Status Word 1
0x0040 STS2 Info: Status Word 2
0x0044 DATA1 Info: Data Word 1
0x0048 DATA2 Info: Data Word 2
0x00C0 DATA32 Info: Data Word 32
1553 Words/Payload
CDP Control Word
R
Forced Word Count
CDP Word Offset for Compare
Rsrv
Reserved
Rsrv
R
1 = Use Forced Word Count
1 = Trig Out on Compare = True
1 = Trig Out on Mesg Complete without Error
1 = Trig Out on Mesg Complete with Error
1 = Flash User LED on Msg Complete
1 = Gen Interrupt on Compare = True
1 = Gen Interrupt on Msg Complete without Error
1 = Gen Interrupt on Msg Complete with Error
1 = Inj Error on TX Data Words
CDP Status Word
Reserved
R
1553 Data Word Cnt
1 = Spurious Mesg
1 = BC-RT
1 = RT-BC
1 = RT-RT
1 = Mode Code
1 = Broadcast
1 = Compare equals True
1 = NO ERROR
1 = Sync
1 = Bit Error/Low Bit
1 = Parity
1='A' Bus: 0='B' Bus
1 = Two Bus Signal
1 = Wrong RT Add Response
1 = No Response
1 = Word Count
CDP 1553 Words
R
Gap Time Value
1553 Word Bits
RX: 1 = Wrong Add for Status Word – TX: 1 = Inj Gap Before Word
RX Only: 1 = No Response
RX Only: 1 = Word Count Error
RX: 1 = Parity Error – TX: 1 = Inj ~Parity Error
RX: 1 = Man Err (Any Bit) – TX: 1 = Inj Man Error (Bit X)
RX: 1 = Sync Error – TX: 1 = Inj ~Sync Error
RX Only: 1 = 'A' Bus; 0 = 'B' Bus
RX = Receiving Words for BC, RT & Monitor.
TX = BC and RT Transmit Words.

Figure 4.2: Common Data Packet Format. Reproduced from [29].

determining whether the word was observed on the primary (A) or backup (B) bus in a dual-redundant bus configuration.

While not used in this prototype implementation, the CDP status word – not to be confused with the MIL-STD-1553 status word – contains a number of flags used to indicate various errors as well as the observed bus communication regime, e.g. BC-RT, RT-RT, spurious message. This information may be useful for future work.

4.1.2.2 Data Collection

While the stated intent of the APMP is to provide a UDP packet stream to an end user for processing by their preferred means, the transmission of packetized bus data enables the use of network packet capture and replay tools. Capturing the transmission of APMP packets allows the bus data to be preserved for processing at a later time.

By parsing the recorded packet captures, it is possible to extract the original bus data and the metadata. This information is sufficient to implement each of the three detection methods proposed in Section 3.2. Packet captures also have the advantage of abstracting away the actual MIL-STD-1553B data bus: once recorded, the traffic capture can be replayed and analyzed as required, eliminating the need for a running bus. This is an important consideration as recorded traffic can be used in offline analysis, including forensic investigation.

For these reasons, Otto will source its data from packet capture files collected by recording the output of the ENET2-1553. In the future, it should be possible to adapt the system to process incoming UDP data in realtime rather than from a packet capture but this is beyond the scope of the current work.

4.1.3 Hardware

As this work focuses entirely on constructing a proof of concept implementation, the initial prototyping of Otto was done using standard desktop computers.

It is anticipated that a detection system monitoring an active MIL-STD-1553B bus in real-time aboard an aircraft will most likely need to run in a resource-limited environment. Because space, weight and power are typically at a premium aboard aircraft, the system could be expected to run on a low-power, lightweight computing platform such as a Raspberry Pi, or to share existing hardware. While these considerations are beyond the scope of the development of a prototype, they may influence future design decisions.

4.1.4 Software

As Otto will be designed as a software solution, the choice of language and operating system will influence the architecture and implementation of the system.

4.1.4.1 Development Language

The decision to use packet capture files as a data source led to the selection of Python for Otto's development. Python's `scapy` library is commonly used to automate packet analysis functions. It is well documented and there are a multitude of examples available for study.

4.1.4.2 Operating System

The choice of operating system for the development of Otto was also guided by the potential for migration to an embedded system. As many embedded systems commonly run a variant of Linux, Otto was developed to run on this. Specifically, a Kali Linux 2016.1 virtual machine provided the primary development and test environment.

An artifact of the selection of Python for development of Otto is that Python interpreters are available for a wide variety of operating systems. OS interoperability was tested in an informal manner toward the end of the development process. Otto ran with no discernible errors within Microsoft Windows Subsystem for Linux using Ubuntu 14.04.5 LTS on Windows 10. No attempt was made to run Otto in a native Microsoft Windows environment.

4.2 Execution Flow

This section will discuss the details of Otto's operations from a dynamic perspective, broken down by functional block. Section 4.3 will present each of these blocks and how data is passed between them.

The flowchart shown in Figure 4.3 illustrates Otto's execution flow.

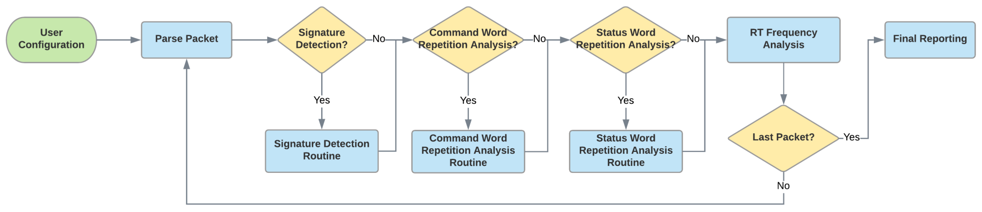

User Configuration
Parse Packet
Signature Detection?
No
Yes
Signature Detection Routine
Command Word Repetition Analysis?
No
Yes
Command Word Repetition Analysis Routine
Status Word Repetition Analysis?
No
Yes
Status Word Repetition Analysis Routine
RT Frequency Analysis
Last Packet?
Yes
No
Final Reporting

Figure 4.3: Otto High-Level Execution Flow

4.2.1 User Configuration

Figure 4.4: User Configuration

Otto is run from the command line. A path to a packet capture (pcap) file containing APMP-formatted packets is required as an argument.

Upon running Otto, the user is presented with options for the configuration of the signature detection and word repetition analysis functions.

For signature analysis, the user is asked to provide a path to files containing signatures for each of command, status and data words. Each signature file is parsed and loaded into Otto, and the parsed contents are displayed to the user as confirmation of successful loading. While signature parsing occurs at this time, a detailed explanation of the process will be given alongside the description of the signature detection routine in Section 4.2.5. Specifying an invalid file name raises a warning message and disables that portion of the signature detector.

The user is then prompted on whether or not to enable word repetition analysis. If so, the user is queried for the size of the window and number of matches to trigger detection. The introduction of the word repetition analysis concept in Section 3.2.2 provides context for the selection of these options.

4.2.2 Packet Parsing

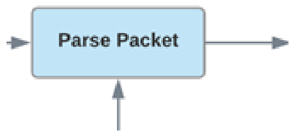


Figure 4.5: Packet Parsing

Otto's packet parsing function takes place automatically without user interaction. The intent of this function is to read the APMP UDP packets recorded in the pcap file specified and extract the data for analysis by the other software functions.

Otto is written such that this parsing function can be easily replaced by another, should APMP-formatted data not be available. Other possible options include a real-time data bus parser, or the ingestion of data captured by other bus monitoring solutions.

The packet capture is assumed to be formatted according to the Alta APMP. Otto does provide an elementary measure of filtering by discarding any non-UDP packets contained in the packet list imported from the pcap file. While it may be possible to integrate more advanced filtering into Otto, this is beyond the scope of this prototype.

MIL-STD-1553 words are extracted by converting the UDP payload into a string, slicing it down, converting the ASCII-encoded data into a binary string, and adding leading zeroes as required. The end result of this process is a 32-character string of ones and zeroes: the 16 bit word and 16 bits of metadata, including the A/B bus flag. The words are further sliced to extract the values of each word's constituent fields (refer to Figures 2.1, 2.3 and 2.2).

The first packet is now parsed and ready for analysis.

4.2.3 Signature-Based Detection

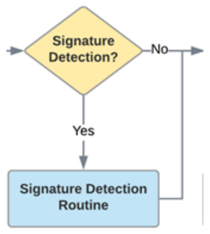


Figure 4.6: Signature Detection Routine

Otto's signature detection functionality is implemented across two separate functions. The first function reads the signature file prepared by the user and loads the contents. The second compares values between the word observed in the pcap and the signature and reports any matches.

4.2.3.1 Signature Definition

Recall from Section 3.2.1 that signatures were defined as parametrically-defined MIL-STD-1553B words and metadata that the operator wishes to be alerted to.

Otto ingests signatures in the form of comma-separated value (CSV) files. CSV files were selected as a medium as they are commonly used in configuration files, are easy to work with, and can be managed using a number of common software tools.

The CSV files are organized with each row representing a signature, while each column represents a parameter. The signature files provided with Otto include a header row specifying each parameter and a range of valid values. Table 4.1 is a representation of a command word signature CSV file, loaded with sample data.

Table 4.1: Sample Command Word Signature File

Address (0-31)	TR Bit (T or R)	Subaddress / Mode (0 to 31)	Word Count / Mode Code (0 to 31)	Bus (A or B)
5	T	12	3	A
18				
	R			B
	T	0	19	
	T	31	19	

In this example, the operator has defined five signatures.

The first signature will flag any command words seen on the primary (A) bus directing subaddress #12 on RT #5 to transmit 3 data words. If any one of these parameters does not match, no alert is to be generated.

The second signature is much more general. This one calls for the reporting of any command words sent to RT #18, regardless of the rest of the content, or what bus it was seen on. Because blank values are treated as wildcards, an RT address match will result in a detection.

The third signature also employs wildcards, but defines two parameters that must be matched: Receive commands sent over the backup (B) bus. Any combination of defined values and wildcards can be used in to define a signature.

The fourth and fifth signatures demonstrate an example of the care that must be taken when the operator is creating signatures. Both of these signatures are intended to search for mode messages instructing any RT to transmit a BIT word. However, as previously discussed in Section 2.1.3, there are two values used in the command word Subaddress/Mode field to indicate a mode message: `0b00000` (0) or `0b11111` (31). In order to cover both possible cases, two signatures are required. It is important that the operator consider whether multiple methods are possible to achieve an effect, and create signatures accordingly.

4.2.3.2 Signature Loading

Signatures from the CSV files are loaded into Otto using functions from the `CSV` library for Python. With the file open, each row is read into a Python list, skipping the first containing the headers.

The user-readable values entered in the CSV file are converted into their binary representation and stored. In addition to the ones and zeroes, `x` characters are used to represent wildcard values. For values normally five bits in length such as RT address and Word Count, a wildcard would be represented as `xxxxx`, while simple flags take only `x`.

Some error-checking has been added to the binary conversion logic to reject invalid values for the single-bit flags. In the future, this could be expanded to provide more rigorous error-checking, including prompting the user for corrected parameters.

As each signature is parsed, it is added to a global list variable. Once all signatures are loaded, Otto is ready to begin comparing signatures to the observed bus traffic.

4.2.3.3 Traffic Comparison

The objective of the traffic comparison functions is simple: to compare the MIL-STD-1553B word to the list of signatures. If a match is detected, the details are passed off to a detection handling function for verification and reporting. This function will be discussed in Section 4.2.7.

A first test is done to verify if all fields of command word and the bus flag contain all ones. Recall from Section 4.1.2.1 that `0xFFFFFFFF` is the null-value for a CDP 1553 word [29]. While this check effectively skips null-value CDP words, it will also skip over a malformed MIL-STD-1553B command word containing all ones broadcast on the A bus.

The first comparison is made between the RT address fields in the traffic and the signature. If they match, a secondary verification is done on the remaining signature fields. If all remaining signature fields either match the command word under evaluation, or are set to the wildcard value of `x`, a detection is confirmed and the detection handling function is called. Otherwise, there is no match and the traffic comparison function carries on.

The T/R bits are then compared in the same way. This process continues until each of the fields in the word has been compared, with secondary verification of the remaining fields in the case of a match.

4.2.4 Word Repetition Analysis

The word repetition analysis function's purpose is to make a list of words seen on the bus, and to consult that list for duplicate entries. While the command word repetition analysis function is discussed here, a nearly identical function carries out the same task for status word repetition analysis. Data

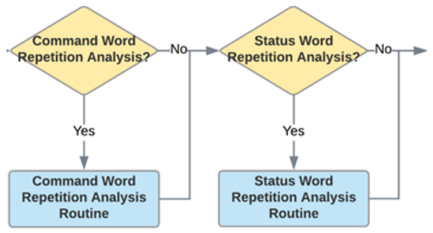

Command Word Repetition Analysis?
No
Yes
Command Word Repetition Analysis Routine
Status Word Repetition Analysis?
No
Yes
Status Word Repetition Analysis Routine

Figure 4.7: Word Repetition Analysis Routine

word repetition analysis is not done, as, due to their arbitrary makeup, the repetition of identical data words is not necessarily suspicious.

The command word repetition analysis function is called after each packet is parsed, with the two extracted command words and the sequential packet ID number as parameters.

Using the data contained in the command word parameters, a string of zero and one characters is built to create a single string representing the command word. This string is then tested for the null value. Should the data be null, the command word is not added to the list and the function moves on.

If the command word is non-null, the list is searched and instances of the same string are counted. If the number of matches equals or exceeds the user-defined limit, the detection handler is called to raise an alert. This same process is carried out for the second command word in the packet.

The final action taken by the function is to cull the command word list down to the maximum length specified by the user, i.e. the window size.

4.2.5 RT Address Frequency Analysis

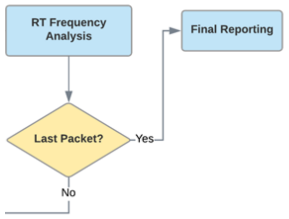


Figure 4.8: RT Frequency Analysis Routine and Final Reporting

The implementation of RT address frequency analysis is relatively simple but nonetheless powerful. After each packet is parsed, the binary representation of all RT addresses seen in both command and status words is appended to a global list.

After all packets have been parsed and analyzed, the number of occurrences for each unique address is tallied in preparation for the final reporting phase, discussed in the next section.

4.2.6 Final Reporting

After all packets have been parsed, the final reporting stage is used to carry out any analysis requiring an overview of the complete data set. In the current prototype implementation, this is limited to reporting the output of the RT address frequency analysis built in the previous section.

The global lists of RT addresses are used to count the instances of each RT address and order them by frequency, constructing histograms of the RT addresses observed in both command and status words. For ease of interpretation, the RT addresses are converted from their binary representation to a decimal value for reporting. Each address is displayed in order from most to least frequent, along with the count for each.

4.2.7 Detection Handling

The detection handling function exists to centralize Otto's alerting. It is called when a detection is made by the signature-matching and word repetition detection functions.

Otto alerts the user of detections by displaying text in the terminal. Writing these to a log would only require a simple modification.

The detection handling function takes a large number of parameters. The first of these is is used to indicate the type of detection, which in turn governs the output produced. Five possible cases are implemented: command, status and data word signature matching, command and status word repetition.

The rest of the parameters are the fields that compose MIL-STD-1553B command and status words, and the bus discriminator flag. The packet ID number is passed, as well as the signature that triggered the detection, if applicable.

Signature match reports begin with the Packet ID number and the Signature ID number, allowing the user to identify which packet contains a match to what signature. The content of the MIL-STD-1553B word parameters matching the fields in the signature are then displayed.

Word repetition detections are more straightforward, reporting the full content of the word causing the detection, and the packet ID of the word that caused the threshold to be exceeded. This information can be used by an analyst to begin further investigation.

4.3 Structural Design

While Section 4.2 discussed Otto's functioning at a conceptual level, this section will provide a brief overview of the software's structure. A more detailed discussion, including imported modules, variables, classes, and the arguments passed in function calls, is available in Appendix A.

4.3.1 Overview

As discussed in Section 4.1, Otto is written in Python. Figure 4.9 illustrates each of the functions in the program and the call relationships.

4.3.2 `main`

The `main` function is both the starting point for the program and where it eventually terminates. It has three main purposes: user configuration, calling the packet parsing function, and final reporting.

The first few lines of the function display details such as the program name, version number and authorship statements. Next the user is prompted to enter paths to command, status and data word signature files, which are passed to the signature loading functions: `loadcommandsig`, `loaddatasig`, and `loaddatasig`. The user is also prompted for the parameters that govern the command and status word repetition detection routines.

The `main` function then calls the `parsePCAP` function, which governs packet parsing. This function is explained in Section 4.3.4.

Once packet parsing is complete, the function's final action before the program terminates is to generate and display the command and status word histograms generated through RT frequency analysis.

4.3.3 `loadcommandsig`, `loadstatussig`, `loaddatasig`

The `loadcommandsig`, `loadstatussig`, and `loaddatasig` functions are similar, with each designed to interpret command, status and data signature files respectively.

In each, the user-specified CSV file is opened and the data is extracted row by row. The human-readable information in each cell row is converted to

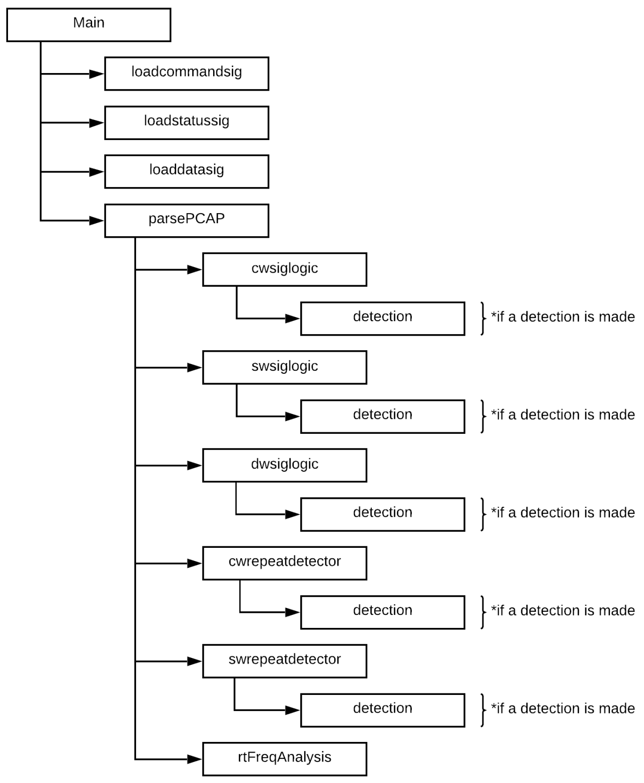


Figure 4.9: Diagram of Function Calls Within Otto

a representative string of binary ones and zeroes according to the MIL-STD-1553 protocol.

Once the final cell in the row is parsed, the resulting strings are concatenated and the new string is written as an element in a list variable. These elements are what is ultimately compared to the data bus traffic.

4.3.4 `parsePCAP`

The `parsePCAP` function is where MIL-STD-1553 data is extracted from the APMP-formatted CDP packets. These packets are parsed one at a time.

Using imports from the `scapy` library, the UDP payload of each packet is extracted and sliced to extract the CDP fields: command word 1, command word 2, status word 1, status word 2 and data words 1 to 32. Each of these are further sliced to extract each of the applicable MIL-STD-1553 fields as well as the A/B bus flag. These are all converted into binary strings, as with the command word signatures.

With the words now extracted from the packet, the signature detection and word repetition detection functions are called. A final call is made to a function that logs all command and status words for RT frequency analysis.

After all function calls have returned, the `parsePCAP` function carries out the same operation on the next packet. After the last packet has been parsed, the function returns to `main`.

4.3.5 `cwsiglogic`, `swsiglogic`, `dwsiglogic`

As the names imply, these functions are responsible for carrying out comparisons between the data extracted from the current packet and the signatures previously loaded. These are called in order by the /codeparsePCAP function.

Starting with the first signature in the list, the MIL-STD-1553 fields are compared to the same fields in the packet. There is logic in place to reject the CDP null value and to account for wildcards.

If a match is observed, the detection handling function is called. This function will be described in Section 4.3.8.

After all signatures have been compared to the packet data, these functions return to the `parsePCAP` function.

4.3.6 `cwrepeatdetector`, `swrepeatdetector`

After all three signature detection functions have run, the `parsePCAP` function will call the two word repetition detection functions.

The command or status word will first be compared to a list of words previously observed on the bus. If the number of matches exceeds the threshold specified by the user, an alert is generated using the `detection` function.

The word is then appended to the aforementioned list, the length of which is then checked. If the length exceeds the maximum number of words specified, i.e. the “window size”, the oldest word on the list is culled.

As with the signature detection functions, once the repetition analysis is complete, these functions return to the `parsePCAP` function.

4.3.7 `rtFreqAnalysis`

This function exists to tally all command and words seen on the bus. It simply generates a string of binary characters representing the word and appends it to a global list variable.

These list variables are ultimately used to generate the histogram at the end of the `main` function, as discussed in Section 4.3.2.

4.3.8 `detection`

The `detection` function's purpose is to generate alerts when detections are raised by the signature or word repetition detection logic.

If the detection is a signature match, the packet number is displayed, along with which word caused the detection, e.g. "Packet #: 36, Command Word 1". The value of each matched MIL-STD-1553 fields is also displayed.

For word repetition detections, the packet and word numbers are displayed, along with the values in each MIL-STD-1553 word field.

While this function was designed to print alerts to the console, it was intended to be adaptable to perform other functions such as writing to a log.

Once the alert has been displayed, the function returns and the detection logic carries on where it left off.

4.4 Testing

Before applying Otto against the scenarios discussed in Chapter 3, a brief testing phase is required to ensure that the detection functions operate as designed. To confirm this, each of the three main detection functions will be exercised using two packet capture files containing MIL-STD-1553B data bus traffic: one from a simple bus to test basic functionality, and one from a more complex bus for testing in a more realistic environment.

This section will discuss the setup used to generate and capture data for analysis, and the experiments used to test the signature-based detection, word repetition analysis and RT address frequency analysis functionalities. The experiment setup used in this chapter will also be applied to the validation phase discussed in Chapter 5.

4.4.1 Experiment Setup

Before Otto can be used, establishing a reliable source of MIL-STD-1553B data bus traffic is essential. The quality of the data analyzed is a fundamental requirement before detection work can begin.

While traffic captured from a live data bus is the ideal source, and indeed a reasonable end state for Otto, a simulated data bus is used to produce traffic for analysis within the context of this work. The experimental setup described in this section will also be used in the validation phase, as will be discussed in Chapter 5.

The experiment setup phase covers of two key topics: the generation and capture of MIL-STD-1553B traffic, and conversion of the captured information to a format usable by Otto.

4.4.1.1 Data Bus Traffic Generation

A commercial data bus simulator is used to generate MIL-STD-1553B data bus traffic suitable for testing. BusTools by Abaco Systems (formerly General Electric Intelligent Platforms) is normally used for prototyping buses and testing new configurations. While the software itself is intuitive and user-friendly, BusTools can be used to accurately represent even the most complex bus arrangements.

The bus is designed using a drag-and-drop interface to place and connect BCs, RTs and bus monitors. The behaviour of each device can then be individually configured. For instance, RTs can be made to return pre-scripted data when polled, including different streams depending on the subaddress requested.

The bus controller's interactions are also configurable: mode messages and data transfers are scripted, including polling frequency and the amount of data to be transferred for each RT. Aperiodic commands are also supported, relying on conditions rather than strict timing.

On the hardware side, BusTools interfaces with a number of devices to provide a breakout to standard data bus connectors. These can be used to incorporate actual MIL-STD-1553B devices for hardware-in-the-loop testing.[2] They also provide a convenient connection point for an external data bus monitor. Figure 4.10, adapted from [31] shows the Abaco R15-USB, a USB-connected breakout device used in this experiment.

BusTools is useful for prototyping data buses or reproducing existing systems in a laboratory environment, the software also ships with a number of pre-configured bus scenarios. These are intended to be used for demonstrations and functional testing. Two of these pre-configured scenarios will be used in the testing phase, as they are conveniently available data sets, and produced independently of the experiment.

The selected scenarios are:

[2]Hardware interfaces for other bus types, including ARINC 429, are available from the manufacturer.

Figure 4.10: Abaco (formerly GE) R15-USB.

ONE_RT The simplest possible bus configuration: a bus controller connected to a single RT. The bus controller periodically requests a data transfer from the RT. Due to its simplicity, this scenario is ideal for initial tests.

FOUR_RT This scenario is slightly more complex: the bus now consists of a bus controller requesting data from four RTs. RT-BC, and RT-RT data transfers are present, both periodic and aperiodic, as well as mode messages. This scenario is designed to represent aircraft instruments reporting data to a mission computer, as well as a representation of the housekeeping mode messages typically seen on a bus.

4.4.1.2 Traffic Capture

As previously discussed, the ENET2-1553 is used to capture MIL-STD-1553B traffic, packetize it into a UDP/IP format and send it out over Ethernet. A detailed overview of how this is done was presented in Section 4.1.2. This section will focus on the setup of the ENET2-1553, and the process for capturing the data from the network and saving it to a packet capture (pcap) file.

The ENET2-1553 is configured using AltaView. This software provides a number of features, but the most relevant to this work is the ability to select bus monitor mode, used to generate APMP packets [32]. The AltaView computer is connected to the ENET2-1553 over Ethernet, as will be shown in Section 4.4.1.3.

The traffic generated by the ENET2-1553 is passively captured using `tcpdump`, using the following arguments:

```
tcpdump -ns 0 -i eth0 ‘src host 192.168.0.128’ -w filename.pcap
```

For ease of physical configuration, the packet capture is executed on the AltaView computer. While both AltaView and the traffic capture are running on the same machine, the two functions are entirely independent of one another.

Once the packet capture is complete, a verification of the resulting pcap file is done using Wireshark. The APMP plugin is used to confirm that all traffic in the capture is actually APMP traffic. This was indeed the case for all packet captures made in the course of this work. If the ENET2-1553 was to have generated any non-APMP traffic, further filtering would be required. This could be done manually within Wireshark, or by modifying the `tcpdump` filter attributes.

Using this setup, 36 packets were captured for the ONE_RT scenario, and 341 for FOUR_RT.

4.4.1.3 Physical Layout

Figure 4.11 illustrates the physical layout of the experimental setup. A USB connection is shown from the ENET2-1553 to the AltaView computer. This connection is simply to provide 5V power, and no data is exchanged over it.

The physical connection to the data bus is made using the Abaco R15-USB. It provides a standard physical breakout of the MIL-STD-1553B data bus simulated by BusTools. The ENET2-1553's bus leads are connected to this, taking care to properly connect the primary and backup bus. Swapping these will result in the observed traffic being tallied against the wrong bus.

Ethernet network connectivity is also required for the ENET2-1553 to output the packetized data. The built-in Ethernet cable was connected to a standard Ethernet switch.

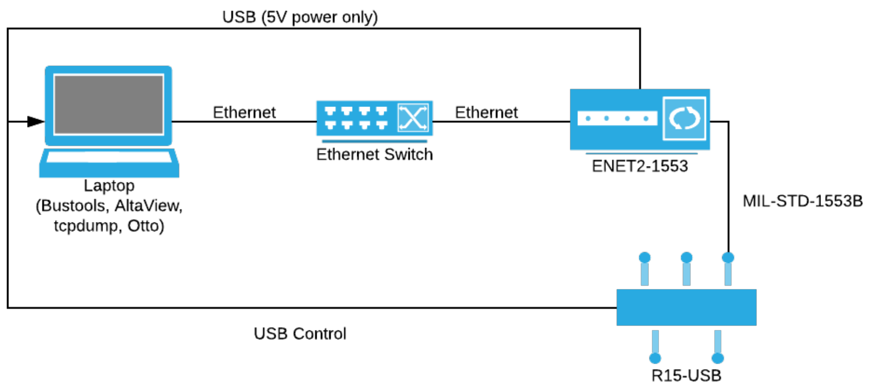


Figure 4.11: Experimental Setup Layout

The final item in the test setup is the AltaView computer, which is connected to the same network switch via Ethernet.

4.4.2 Signature-Based Detection

Signatures are entered into the command or status word signature file and Otto is run to process the pcap. Alerts are expected from signatures where the entered characteristics are known to be present in the data, while signatures not present should never raise alerts. Additionally, no false alarms should be observed.

4.4.2.1 ONE_RT

Because the ONE_RT scenario is fairly basic, there are a limited number of signatures that can be expected to generate alerts. However, this basic test is a good starting point for ensuring functionality.

Studying the ONE_RT pcap file in Wireshark reveals that the command word sent is identical in every case: a transmit request to RT #1, subaddress #2 for four words of data, sent over the A bus. Similarly, the status word is always RT #1 replying with no flags prior to sending the data, again over the A bus. Because every packet is similar, a signature crafted to detect any of these parameters is expected to return 36 detections, one for each packet. Conversely, signatures containing none of these parameters will generate no detections.

Command Word Signature-Based Detection Table 4.2 shows the command word signatures chosen to test the ONE_RT scenario. Recall the signature definition described in Section 4.2.3.1: each line contains a signature, and each column represents a different parameter. While signature numbers have been added to the table for ease of reading, these are not defined in the CSV file. Rather, they are assigned automatically by Otto during the signature loading phase described in Section 4.2.3.2

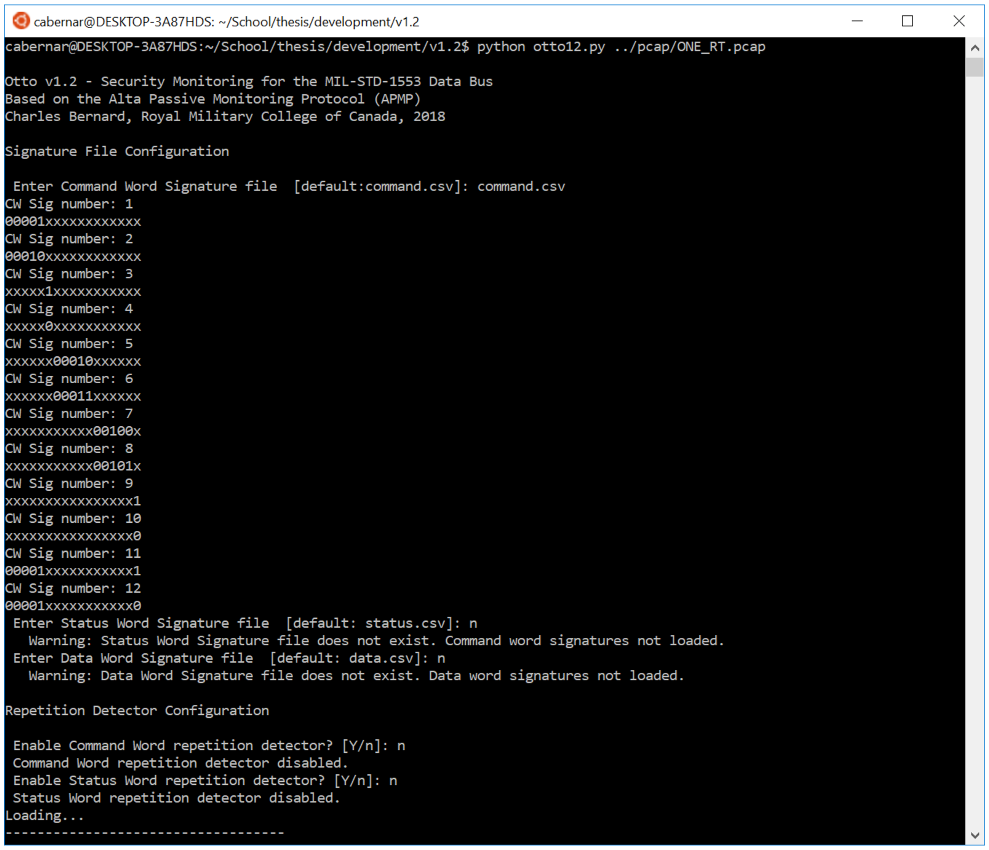
Table 4.2: ONE_RT Command Word Signature File

Number	Address (0-31)	T/R Bit (T or R)	Subaddress / Mode (0 to 31)	Word Count / Mode Code (0 to 31)	Bus (A or B)
1	1				
2	2				
3		T			
4		R			
5			2		
6			3		
7				4	
8				5	
9					A
10					B
11	1				A
12	1				B

The command word signature file was crafted to test each individual parameter with both a matching and non-matching value. It is expected that odd numbered lines will raise alerts, while even lines will not. The final two lines test the ability to match multiple parameters in single signature. Again, the first of these should result in matches, while the second should never appear.

Figure 4.12 illustrates a typical Otto session. Once invoked from the command line, some boilerplate information is displayed, including title, version number, a brief description and an ownership statement.

The user is then prompted to enter a command word signature file name, with a default option of command.csv. Upon loading the file, Otto prints each signature to console, starting with a sequentially assigned signature number, followed by the 17-bit string representation of each signature. In this string,



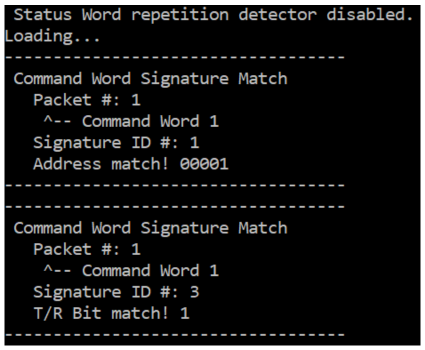
```
cabernar@DESKTOP-3A87HDS: ~/School/thesis/development/v1.2
cabernar@DESKTOP-3A87HDS:~/School/thesis/development/v1.2$ python otto12.py ../pcap/ONE_RT.pcap

Otto v1.2 - Security Monitoring for the MIL-STD-1553 Data Bus
Based on the Alta Passive Monitoring Protocol (APMP)
Charles Bernard, Royal Military College of Canada, 2018

Signature File Configuration

 Enter Command Word Signature file  [default:command.csv]: command.csv
CW Sig number: 1
00001xxxxxxxxxxx
CW Sig number: 2
00010xxxxxxxxxxx
CW Sig number: 3
xxxxx1xxxxxxxxxx
CW Sig number: 4
xxxxx0xxxxxxxxxx
CW Sig number: 5
xxxxxx00010xxxxx
CW Sig number: 6
xxxxxx00011xxxxx
CW Sig number: 7
xxxxxxxxxx00100x
CW Sig number: 8
xxxxxxxxxx00101x
CW Sig number: 9
xxxxxxxxxxxxxxx1
CW Sig number: 10
xxxxxxxxxxxxxxx0
CW Sig number: 11
00001xxxxxxxxxx1
CW Sig number: 12
00001xxxxxxxxxx0
 Enter Status Word Signature file  [default: status.csv]: n
   Warning: Status Word Signature file does not exist. Command word signatures not loaded.
 Enter Data Word Signature file  [default: data.csv]: n
   Warning: Data Word Signature file does not exist. Data word signatures not loaded.

Repetition Detector Configuration

 Enable Command Word repetition detector? [Y/n]: n
 Command Word repetition detector disabled.
 Enable Status Word repetition detector? [Y/n]: n
 Status Word repetition detector disabled.
Loading...
-----------------------------------
```

Figure 4.12: Command word signature loading for ONE_RT

the x character denotes an unspecified parameter, which is treated as a wildcard. This string is what is ultimately compared to the data lifted from the pcap file. By grouping the bits according to the word layout and converting the binary representation of the numerical fields to decimal, these strings can be compared to the signatures defined in Table 4.2. In this case, these strings are confirmed to have been correctly generated.

Status and data word signatures are loaded in a similar fashion. Providing an invalid signature file path opts out of doing signature detection. The subsequent inputs toggle command and status word repetition detection.

Figure 4.13 shows a sample of the output produced when a signature match is detected. The user is shown the packet where the detection is made, which of the two command words within the packet triggered the detection, which signature was detected, and a short summary of the detection.



```
 Status Word repetition detector disabled.
Loading...
-----------------------------------
 Command Word Signature Match
   Packet #: 1
    ^-- Command Word 1
   Signature ID #: 1
   Address match! 00001
-----------------------------------
-----------------------------------
 Command Word Signature Match
   Packet #: 1
    ^-- Command Word 1
   Signature ID #: 3
   T/R Bit match! 1
-----------------------------------
```

Figure 4.13: Command word signature detection results for ONE_RT

Combing through the output, the results matched the initial expectation, with only odd-numbered signatures raising alerts and no even-numbered signature alerts observed.

In order to check for false alarms, a random sample of alerts were manually analyzed, comparing each to the raw packet dissected in Wireshark using the Alta CDP plugin. For all alerts investigated, the signature and packet data were a match, confirming the validity of the alert. Another random sampling of packets for manual analysis revealed no false negatives. While we realize that this is not exhaustive testing, we are able to confirm that there are no programmatic reasons why alerts would not be raised.

Status Word Signature-Based Detection Table 4.3 shows the contents of the status word signature file, with signature numbers added for clarity.

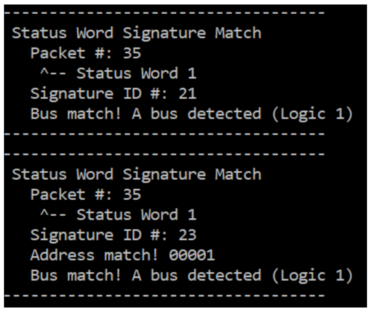
Table 4.3: ONE_RT Status Word Signature File

Number	Address	E	I	SR	Reserved	BC	B	SF	DBCA	T	Bus
1	1										
2	3										
3		N									
4		Y									
5			N								
6			Y								
7				N							
8				Y							
9					0						
10					1						
11						N					
12						Y					
13							N				
14							Y				
15								N			
16								Y			
17									N		
18									Y		
19										N	
20										Y	
21											A
22											B
23	1										A
24	2										B

The status word signature file was designed similarly to the command word signature file, with odd lines generating alerts and even lines generating none.

This was indeed the case: Otto produced a long list of detections against odd-numbered signatures, and no even-numbered signatures to be seen. This test can therefore be deemed successful.

The partial output shown in Figure 4.14 illustrates alerting for single-parameter signatures (#21) and multiple-parameter signatures (#23).



```
---------------------------------
Status Word Signature Match
  Packet #: 35
   ^-- Status Word 1
  Signature ID #: 21
  Bus match! A bus detected (Logic 1)
---------------------------------
---------------------------------
Status Word Signature Match
  Packet #: 35
   ^-- Status Word 1
  Signature ID #: 23
  Address match! 00001
  Bus match! A bus detected (Logic 1)
---------------------------------
```

Figure 4.14: Status word signature loading for ONE_RT

4.4.2.2 FOUR_RT

As mentioned, the traffic contained in the FOUR_RT scenario is more diverse than the ONE_RT capture, containing a number of data transfers to and from different RTs, as well as mode messages and messages sent over the B bus. With the ONE_RT testing confirming basic detection functionality, the FOUR_RT tests focused on exercising the detection of these more complex events.

Command Word Signature-Based Detection The command word signature file used in this test is replicated in Table 4.4.

Table 4.4: FOUR_RT Command Word Signature File

Number	Address (0-31)	T/R Bit (T or R)	Subaddress / Mode (0 to 31)	Word Count / Mode Code (0 to 31)	Bus (A or B)
1	5				
2					B
3			0		
4			31		

In this file, the first rule is never expected to generate an alert, as only RT addresses 1 to 4 are present on the bus. The second rule should generate occasional alerts, as the traffic sample is known to contain some B-bus activity. The third and fourth rules work together to detect mode messages. Two rules are required, as both 0 and 31 in the subaddress/mode field indicate a mode message.

A sample of the alerts yielded by this ruleset against the FOUR_RT traffic sample is shown in Figure 4.15.

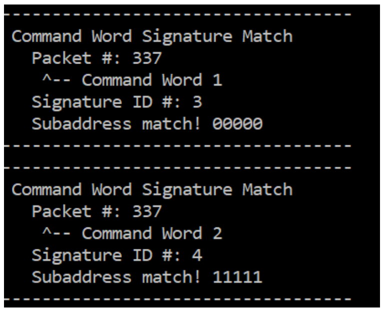

```
----------------------------------
Command Word Signature Match
  Packet #: 337
   ^-- Command Word 1
  Signature ID #: 3
  Subaddress match! 00000
----------------------------------
----------------------------------
Command Word Signature Match
  Packet #: 337
   ^-- Command Word 2
  Signature ID #: 4
  Subaddress match! 11111
----------------------------------
```

Figure 4.15: Command word signature detection results for FOUR_RT

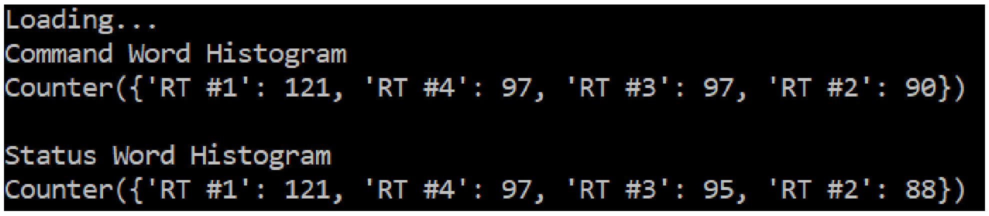
Manually reviewing each alert confirmed the expected behaviour described above. No alerts generated by the first rule were seen. A number of B-bus traffic mode message alerts were generated, which were confirmed using Wireshark. A random sampling of packets were dissected in Wireshark to search for false negatives, none of which were noted.

Status-Word Signature-Based Detection The status word signature file used is similar to the one used against the ONE_RT scenario, pared down to avoid repeating tests done against the previous data file. The contents of the file are shown in Table 4.5.

Table 4.5: FOUR_RT Status Word Signature File

Number	Address	E	I	SR	Reserved	BC	B	SF	DBCA	T	Bus
1	7										
2		Y									
3			Y								
4					7						
5						Y					
6							Y				
7								Y			
8									Y		
9										Y	
10											B
11	7										B

The output generated is shown in Figure 4.16. As expected, no detections were made: Otto proceeds directly from the "Loading..." indicator to the RT frequency analysis output, which will be further discussed in Section 4.4.4



```
Loading...
Command Word Histogram
Counter({'RT #1': 121, 'RT #4': 97, 'RT #3': 97, 'RT #2': 90})

Status Word Histogram
Counter({'RT #1': 121, 'RT #4': 97, 'RT #3': 95, 'RT #2': 88})
```

Figure 4.16: Status word signature detection results for FOUR_RT

With both the ONE_RT and FOUR_RT scenarios producing the expected results, the status word signature detection mechanism was deemed to be correctly implemented.

4.4.3 Word Repetition Analysis

The ONE_RT and FOUR_RT pcap files lend themselves well to testing of the word repetition analysis function. Both scenarios cycle through a scripted sequence of bus interactions, and this cycle causes words to be repeated. This assures repetition in the files which can be detected.

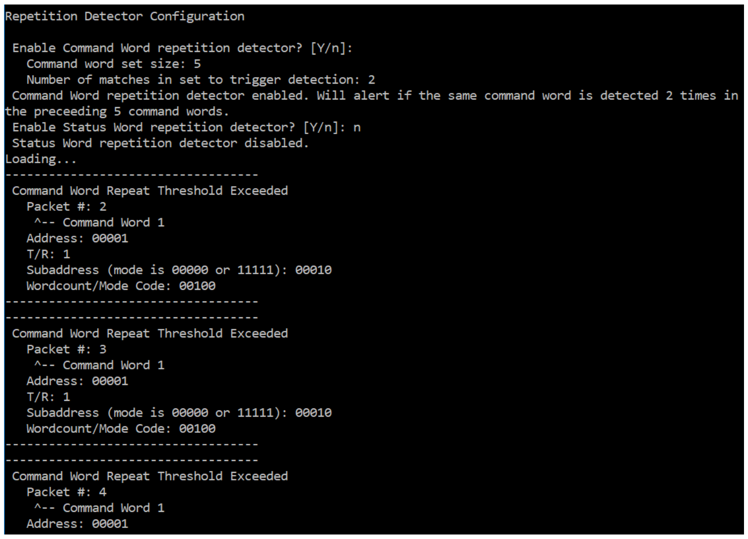
4.4.3.1 ONE_RT

As the ONE_RT scenario consists entirely of one RT-BC data transfer on a loop, it is an ideal cast for a basic functionality test.

Command Word The only command word expected to be seen in this capture is the RT-BC data transfer command. Therefore, any threshold set to 2 or greater should yield a detection, regardless of the selected command word set size.

Figure 4.17 shows the result when the threshold is set to 2. As expected, alerts were generated for every command word in the capture from the second packet onward.

This test was repeated with thresholds of 3, 4 and 7 with arbitrary command word set sizes in order to verify this behaviour. As expected, detections



```
Repetition Detector Configuration

 Enable Command Word repetition detector? [Y/n]:
   Command word set size: 5
   Number of matches in set to trigger detection: 2
 Command Word repetition detector enabled. Will alert if the same command word is detected 2 times in
the preceeding 5 command words.
 Enable Status Word repetition detector? [Y/n]: n
 Status Word repetition detector disabled.
Loading...
-----------------------------------
 Command Word Repeat Threshold Exceeded
   Packet #: 2
    ^-- Command Word 1
   Address: 00001
   T/R: 1
   Subaddress (mode is 00000 or 11111): 00010
   Wordcount/Mode Code: 00100
-----------------------------------
-----------------------------------
 Command Word Repeat Threshold Exceeded
   Packet #: 3
    ^-- Command Word 1
   Address: 00001
   T/R: 1
   Subaddress (mode is 00000 or 11111): 00010
   Wordcount/Mode Code: 00100
-----------------------------------
-----------------------------------
 Command Word Repeat Threshold Exceeded
   Packet #: 4
    ^-- Command Word 1
   Address: 00001
```

Figure 4.17: Command Word Repetition Analysis for ONE_RT

were made on every packet, starting from the third, fourth and seventh respectively.

This test successfully demonstrates the basic functioning of command word repetition analysis.

Status Word As in the previous test case, every interaction captured contains the same status word. The expectation is therefore the same: a detection on every packet beginning from the user-selected threshold.

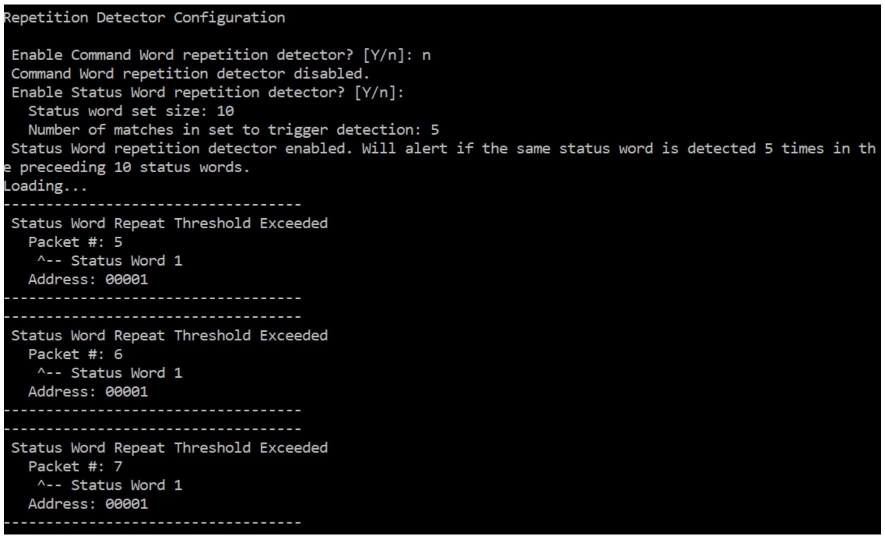
Figure 4.18 shows the partial output of the test with the threshold set to 5. As expected, an alert is generated for each packet from the fifth onward.[3]

This test was again repeated with arbitrary threshold values, and in each case, the packet number of the first alert coincided with the selected threshold. This test is also deemed successful.

4.4.3.2 FOUR_RT

Rather than containing a single repeated bus interaction, the FOUR_RT capture contains a number of data transfers between multiple RTs and the BC, as well as mode messages. The scenario is cyclical, however, repeating the same

[3]For clarity, reporting of the full packet content in Figure 4.18 has been suppressed, with only the RT address shown in the output.



```
Repetition Detector Configuration

 Enable Command Word repetition detector? [Y/n]: n
 Command Word repetition detector disabled.
 Enable Status Word repetition detector? [Y/n]:
   Status word set size: 10
   Number of matches in set to trigger detection: 5
 Status Word repetition detector enabled. Will alert if the same status word is detected 5 times in th
e preceeding 10 status words.
Loading...
-----------------------------------
 Status Word Repeat Threshold Exceeded
   Packet #: 5
    ^-- Status Word 1
   Address: 00001
-----------------------------------
-----------------------------------
 Status Word Repeat Threshold Exceeded
   Packet #: 6
    ^-- Status Word 1
   Address: 00001
-----------------------------------
-----------------------------------
 Status Word Repeat Threshold Exceeded
   Packet #: 7
    ^-- Status Word 1
   Address: 00001
-----------------------------------
```

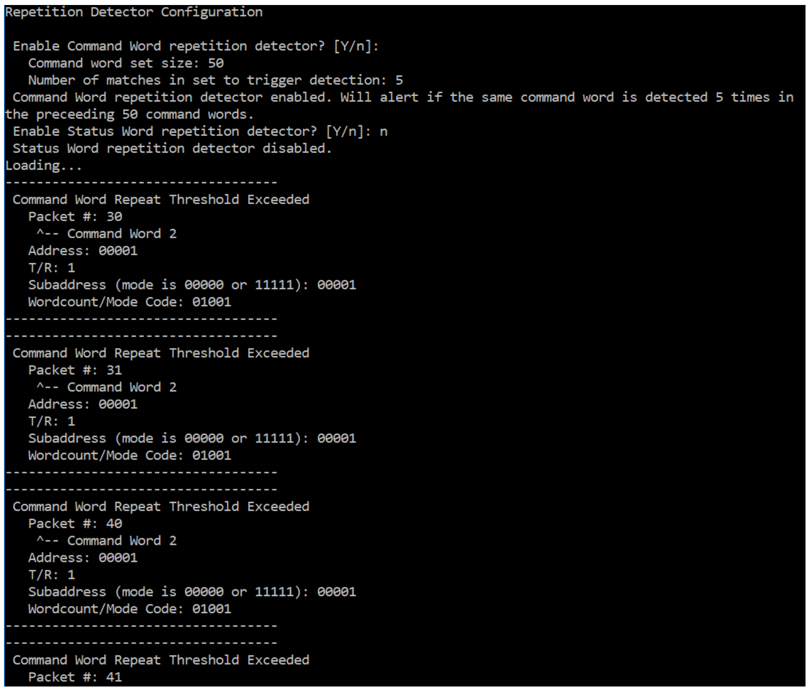
Figure 4.18: Status Word Repetition Analysis for ONE_RT

script of operations on a loop. Given the length of the capture, 341 packets containing up to two command words each, some repetition is expected.

Command Word The starting point for the test was a set size of 50 words and a threshold of 5. With these settings, Otto will generate an alert if the same word is seen five times in the previous fifty words. Figure 4.19 shows the output generated.

The first alert is generated on the second command word of packet #30, for a command word requesting the transmission of nine words of data from RT #1, subaddress #1. This same word generates an alert on the second command word of packets #31, #40 and #41. If packet #30 is the fifth time this word was seen, it would be reasonable to hypothesize that the four previous times were on the second command words of packets #10, #11, #20 and #21.

Verification of this hypothesis using Wireshark shows that this is indeed the case for packets #20 and #21 and #10, but packet #11 does not contain the word. Instead, packet #9 does. This implies that either the word does not appear at strict intervals, the words are cyclical but irregularly, or the end of the BusTools script cycle was captured somewhere between packet #10 and #20, throwing off the stagger interval. Further analysis would be required to verify this, but is beyond the scope of this functional test.



```
Repetition Detector Configuration

 Enable Command Word repetition detector? [Y/n]:
   Command word set size: 50
   Number of matches in set to trigger detection: 5
 Command Word repetition detector enabled. Will alert if the same command word is detected 5 times in
the preceeding 50 command words.
 Enable Status Word repetition detector? [Y/n]: n
 Status Word repetition detector disabled.
Loading...
----------------------------------
 Command Word Repeat Threshold Exceeded
   Packet #: 30
    ^-- Command Word 2
   Address: 00001
   T/R: 1
   Subaddress (mode is 00000 or 11111): 00001
   Wordcount/Mode Code: 01001
----------------------------------
----------------------------------
 Command Word Repeat Threshold Exceeded
   Packet #: 31
    ^-- Command Word 2
   Address: 00001
   T/R: 1
   Subaddress (mode is 00000 or 11111): 00001
   Wordcount/Mode Code: 01001
----------------------------------
----------------------------------
 Command Word Repeat Threshold Exceeded
   Packet #: 40
    ^-- Command Word 2
   Address: 00001
   T/R: 1
   Subaddress (mode is 00000 or 11111): 00001
   Wordcount/Mode Code: 01001
----------------------------------
----------------------------------
 Command Word Repeat Threshold Exceeded
   Packet #: 41
```

Figure 4.19: Command Word Repetition Analysis for FOUR_RT

Another notable item is that the command word repeated appears in the second command word of the packet. This would most likely mean that the first command word is used to order another RT to receive traffic, which would make these RT-RT data transfers. Because the first command word doesn't trip the detector, we can surmise that not all the associated receive commands are the same. Therefore, RT #1 is likely feeding data to multiple RTs.

Again turning to Wireshark to analyze the capture file, this supposition is confirmed. The first command words in packets #9, #20, #30 and #40 instruct RT #3 to receive nine words of data, while #10, #21, #31 and #41 give the receive instruction to RT #2.

Continuing to plot the gaps between these pairs of packets, it could be possible to work out the periodicity of the capture and narrow down the range where the script loops back to the start. While this could be developed into an additional analysis technique, this investigation is beyond the scope of this test.

Elsewhere in the Otto output, similar patterns can be found. Packets #42, #43, #44 and #45 all contain the synchronize mode message. However, the

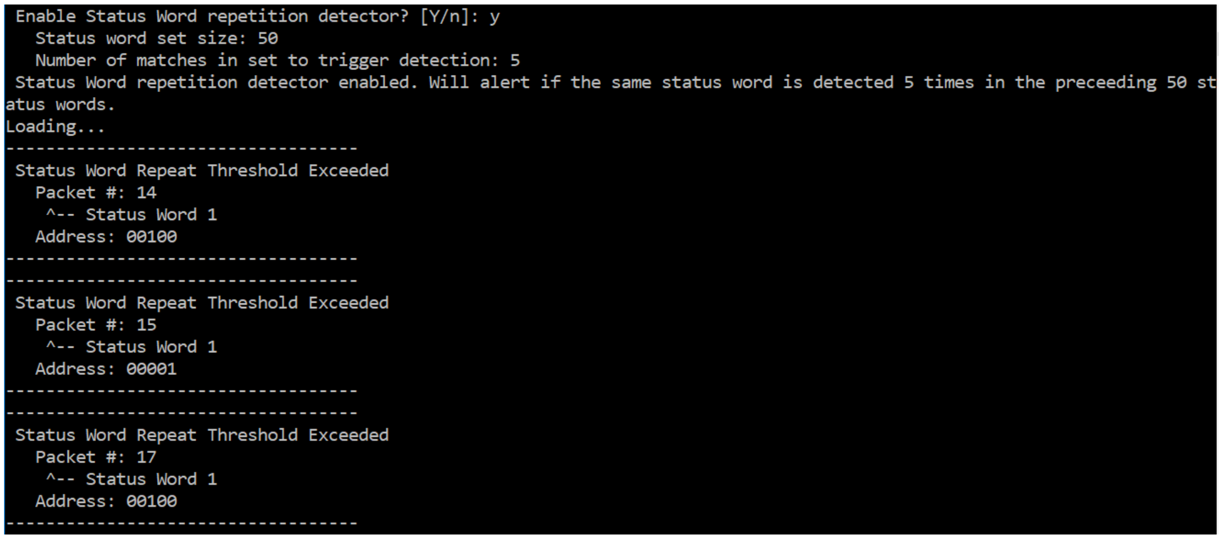
RT addresses for each are different: RT #1, #2, #3 and #4 respectively. The same string of messages is seen again in packets #53-56, #64-67, #74-77, and so on. Working backwards through Wireshark, this string is repeated on packets #32-35, #22-25, #11-14 and #1-4.

It would appear that this sequence appears every 10th or 11th packet. Again, more analysis would likely reveal clues about the periodicity of these transfers and insight into the bus's function. These clues may also inform attempts to determine each RT's function.

While much of this analysis goes beyond the scope of the testing in this chapter, it does demonstrate how word repetition analysis can be used to characterize a data bus. The data obtained through repetition analysis can also be used to craft signatures to obtain even more focused information.

Returning to the functional testing, further analysis of other alerts generated reveals similar repeating behaviours, all of which is confirmed by comparing the alerts to the packet content using Wireshark. While this analysis was not exhaustive, the accuracy with which patterns are detected demonstrates the expected function of the command word repetition analysis module.

Status Word Taking the same parameters used in the command word example, namely a 5 repetition threshold in a set size of 50 words, generates another long string of alerts. A truncated representation of the output is shown in Figure 4.20.



```
 Enable Status Word repetition detector? [Y/n]: y
   Status word set size: 50
   Number of matches in set to trigger detection: 5
 Status Word repetition detector enabled. Will alert if the same status word is detected 5 times in the preceeding 50 st
atus words.
Loading...
-----------------------------------
 Status Word Repeat Threshold Exceeded
   Packet #: 14
    ^-- Status Word 1
   Address: 00100
-----------------------------------
-----------------------------------
 Status Word Repeat Threshold Exceeded
   Packet #: 15
    ^-- Status Word 1
   Address: 00001
-----------------------------------
-----------------------------------
 Status Word Repeat Threshold Exceeded
   Packet #: 17
    ^-- Status Word 1
   Address: 00100
-----------------------------------
```

Figure 4.20: Status Word Repetition Analysis for FOUR_RT

Upon investigating content of the first status words flagged, it becomes clear that these are basic acknowledgment status words: the ones sent by the RT in response to a command, containing only the RT address and no flags. With a set size as large as 50 and a relatively low threshold of 5, these detections are to be expected as these acknowledgment messages are somewhat common. Combing through the previous packets in Wireshark, it is confirmed that the status words flagged are indeed the fifth occurrence of the word, as well as any subsequent occurrences.

Interestingly, the second status words in packets #20 and #21 have also generated detections. These packets correspond to the RT-RT data transfer discussed in the previous section. This stands to reason, since the command words that caused a detection would appear alongside the corresponding status words, and the same detection parameters were specified.

Overall, the word repetition analysis function appears to work as designed. However, the FOUR_RT status word test revealed that a large number of detections were made on the acknowledgment status words generated by RTs. Since these are not uncommon, the alerts are not unexpected, but they do create clutter in the console. Having Otto outright ignore matches for these simple status words may not be desirable, as they may be important to an analyst, particularly when trying to detect status word flooding. Careful adjustment of the set size and threshold may minimize these benign alerts. A toggle feature for disabling repetition alerts for these types of status words (either entirely or selectively by RT address) could be valuable for future implementation.

4.4.4 RT Address Frequency Analysis

Because the RT address frequency analysis generates a histogram of the command and status words seen, the functionality can be confirmed simply by studying the histogram generated after processing each traffic sample.

4.4.4.1 ONE_RT

Because the ONE_RT scenario contains only a single RT, the histograms generated by Otto should contain only a single RT address: #1. Furthermore, because the bus traffic consists of RT-BC data transfers, the number of command words should equal the number of status words. Finally, because each packet is known to contain only one command word and one status word, the number of packets should correspond to the number of command and status words seen.

Upon inspecting the ONE_RT.pcap file in Wireshark, it is shown to contain 36 packets, recorded over a duration of 8.75 seconds. Therefore, we would expect to see 36 command words and 36 status words.

The data compiled by Otto is reproduced in Table 4.6. Also indicated is the difference in the number of command and status words observed, calculated as command words - status words. While the number of command and status words will not always necessarily be equal, this can be a useful metric in characterizing the bus.

Table 4.6: Distribution of Command and Status Words for RT Frequency Analysis (ONE_RT)

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	36	100.00%	36	100.00%	0
Total	36	100.00%	36	100.00%	0

As expected, 36 command words and 36 status words were observed, all containing RT address #1. This corresponds to both the size of the pcap file and simulated bus configuration.

4.4.4.2 FOUR_RT

While the FOUR_RT capture is more complex than the previous example, the results are still predictable.

Only four RT addresses are expected to be present: RT #1 to RT #4. Because the bus traffic in the FOUR_RT sample contains a wider variety of traffic, e.g. RT-RT and RT-BC data transfers as well as mode messages, the traffic will not necessarily be equally distributed between all RTs.

Table 4.7 shows Otto's results from the FOUR_RT sample.

Table 4.7: Distribution of Command and Status Words for RT Frequency Analysis (FOUR_RT))

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	121	29.88%	121	30.17%	0
2	90	22.22%	88	21.95%	2
3	97	23.95%	95	23.69%	2
4	97	23.95%	97	24.19%	0
Total	405	100.00%	401	100.00%	0

The most striking result from the table is the difference in the number of command and status words for RT #2 and #4. In order to verify this, the FOUR_RT pcap file was examined in Wireshark using the Alta-provided APMP plugin to determine whether or not the discrepancy is present in the actual data.

Upon inspection, it was noted that four packets (#73, #153, #229 and #312) contained parity errors in the 9th data word sent from RT #1 to RTs #2 and #4. While RT #1 sends the expected status word prior to transmitting the data, the receiving RTs appear to fail silently, and generate no status word at all. This error may be attributable to the R15-USB. Consultation with colleagues revealed that similar errors have been observed in experiments using a comparable equipment configuration, but the cause has not been conclusively identified. Nevertheless, as the observed discrepancy is present in the data, this is not a fault with the RT frequency analysis function.

Overall, the RT address frequency analysis function appears to behave as designed: the resulting data contains only assigned RT addresses, the number of observed words aligns with the 341 packets contained in the pcap file, and the percentage of total traffic for each RT appears reasonable, with no one device dominating the traffic.

In addition to passing the basic functional check, RT address frequency analysis also enabled the detection of errors in the FOUR_RT traffic sample, illustrating the usefulness of this feature.

4.5 Conclusion

This chapter's objective was to lay out the design and implementation of the detection methods discussed in Chapter 3, and to test Otto's functionality in preparation for validation.

Each detection method addressed in Section 3.2 was implemented in Python, along with the supporting infrastructure required to configure Otto, load signatures, parse data bus traffic, and raise alerts.

The method for capturing data bus traffic using the ENET2-1553 described in Section 4.4.1.2 was found to be suitable. All pcap files recorded were readily detected and interpreted by Wireshark's Alta CDP plugin, with no data corruption noted.

The signature-detection analysis function operated correctly after correction of some minor bugs. Alerts were generated only where expected based on prior knowledge of the traffic sample, with no false negatives observed in random sampling.

Word repetition analysis functionality also performed as expected, with alerts confirmed by manually processing the pcap files in Wireshark. The test also demonstrated how this form of analysis can be used to gain insight into the overall bus design and function.

Otto's RT address frequency analysis component also performed as designed, and uncovered a number of parity errors in the traffic generated from the FOUR_RT scenario.

Testing of each module did rely on varying degrees of random sampling rather than exhaustive test coverage. Nonetheless, the testing conducted confirmed that Otto's performance is sufficient to proceed. Within these random samples of alerts, no discrepancies were observed, nor were false negatives noted.

Overall, the implementation of the traffic analysis functionalities in Otto were found to be implemented correctly, and the results of the testing phase provide sufficient confidence to proceed with validation experiments.

5 Validation

In order to validate the effectiveness of the proposed design, the six scenarios discussed in Chapter 3 were recreated for testing in Otto. It should be noted that the intent of these experiments is to detect the presence of the bus activity associated with each scenario, and not to test the effectiveness of the attacks against the MIL-STD-1553B bus.

Four aspects of each scenario will be discussed, as follows:

- a brief summary of the scenario;
- the mechanism expected to make a required detection;
- experimentation description; and
- a discussion of the effectiveness of the detection method, as well as any other observations.

For each experiment proposed by Stan et al. in [3], the scenario was reproduced using a high-fidelity commercial MIL-STD-1553B data bus simulator as discussed in [25]. Whereas the previous chapter relied on the BusTools bus prototyping software to provide basic simulations suitable for testing, the simulator used in this chapter is of the calibre used for development of avionic components and software. This provides a more realistic traffic capture, containing large amounts of traffic generated by a multitude of bus-connected devices.

In order to characterize the bus and to provide a baseline for comparison, a traffic capture was made with the simulator in a known good configuration, i.e. unaffected by any sort of attack. Table 5.1 shows a breakdown of the traffic observed, as well as the CW - SW metric introduced in Section 4.4.4.

Table 5.1: Baseline Traffic Capture for Experimental Scenarios

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	3,753	35.61%	3,745	35.62%	8
2	2,739	25.99%	2,735	26.01%	4
3	1,537	14.58%	1,525	14.50%	12
4	1,208	11.46%	1,208	11.49%	0
5	1,144	10.85%	1,144	10.88%	0
6	158	1.50%	158	1.50%	0
Total	10,539	100.00%	10,515	100.00%	24

Based on this capture, the bus has six RTs, addressed #1 through #6. These addresses also correspond to how much traffic each generates, with RT #1 responsible for over 35% of command and status words, while RT #6 handles a mere 1.50% of each.

In all cases throughout this chapter, recording of the bus traffic was done using following the method described in Section 4.4.1: connecting the ENET2-1553 to the bus simulator's breakout ports and capturing the resulting network traffic using `tcpdump`.

5.1 Transmission Timing DoS (Behaviour Manipulation)

5.1.1 Scenario

This scenario involves flooding the data bus with traffic in order to consume available time slots on the bus and impede the transmission of legitimate traffic.

As Stan et al. make no distinction between the word types used to flood the bus, two separate experiments were conducted within this scenario: one focusing on command word flooding, and the other on status word flooding [3].

5.1.2 Expected Detection Mechanism

Detection of a transmission timings DoS is complicated by the fact it can be carried out using any word type, at a rate selectable by the attacker, and does not necessarily require the use of legal words, i.e. words in compliance with the standard. However, the detection methods described in Chapter 3 have the flexibility to cope with this [25].

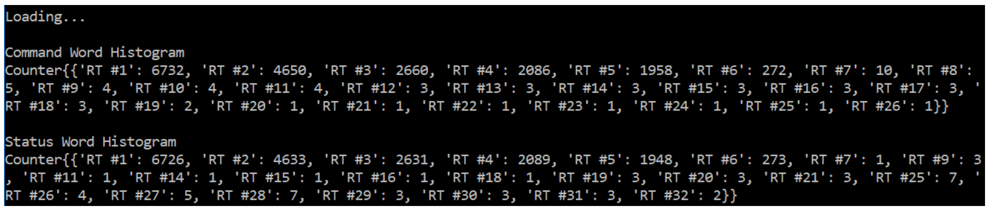
If the bus is flooded using a single word repeated, it can be detected using word repetition detection. Additionally, if the composition of the flooding word is previously known to the analyst or discovered in the course of analysis, a signature can also be written to detect it [25].

If the flooding is caused by multiple different words, either randomly generated or following a prescribed pattern, the flooding can be detected using the RT histogram, as irregular traffic patterns and unassigned RT addresses are likely to appear [25].

5.1.3 Experimentation

5.1.3.1 Command Word Flooding

As illustrated in Figure 5.1, more RT addresses were observed in this capture than were present in the baseline data (Table 5.1).



```
Loading...

Command Word Histogram
Counter({'RT #1': 6732, 'RT #2': 4650, 'RT #3': 2660, 'RT #4': 2086, 'RT #5': 1958, 'RT #6': 272, 'RT #7': 10, 'RT #8':
5, 'RT #9': 4, 'RT #10': 4, 'RT #11': 4, 'RT #12': 3, 'RT #13': 3, 'RT #14': 3, 'RT #15': 3, 'RT #16': 3, 'RT #17': 3, '
RT #18': 3, 'RT #19': 2, 'RT #20': 1, 'RT #21': 1, 'RT #22': 1, 'RT #23': 1, 'RT #24': 1, 'RT #25': 1, 'RT #26': 1})

Status Word Histogram
Counter({'RT #1': 6726, 'RT #2': 4633, 'RT #3': 2631, 'RT #4': 2089, 'RT #5': 1948, 'RT #6': 273, 'RT #7': 1, 'RT #9': 3
, 'RT #11': 1, 'RT #14': 1, 'RT #15': 1, 'RT #16': 1, 'RT #18': 1, 'RT #19': 3, 'RT #20': 3, 'RT #21': 3, 'RT #25': 7, '
RT #26': 4, 'RT #27': 5, 'RT #28': 7, 'RT #29': 3, 'RT #30': 3, 'RT #31': 3, 'RT #32': 2})
```

Figure 5.1: RT Histogram for Command Word Flooding

The data from the histogram is reproduced in Table 5.2, along with relative percentages of overall traffic, and the discrepancy between command and status words. This is immediately indicative of questionable traffic on the bus, as most of these addresses have not been assigned.

Table 5.2: Distribution of Command and Status Words for Command Word Flooding

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	6,732	36.56%	6,726	36.66%	6
2	4,650	25.25%	4,633	25.25%	17
3	2,660	14.44%	2,631	14.34%	29
4	2,086	11.33%	2,089	11.38%	-3
5	1,958	10.63%	1,948	10.62%	10
6	272	1.48%	273	1.49%	-1
7	10	0.05%	1	0.01%	9
8	5	0.03%	0	0.00%	5
9	4	0.02%	3	0.02%	1
10	4	0.02%	0	0.00%	4
11	4	0.02%	1	0.01%	3
12	3	0.02%	0	0.00%	3
13	3	0.02%	0	0.00%	3
14	3	0.02%	1	0.01%	2
15	3	0.02%	1	0.01%	2
16	3	0.02%	1	0.01%	2
17	3	0.02%	0	0.00%	3
18	3	0.02%	1	0.01%	2
19	2	0.01%	3	0.02%	-1
20	1	0.01%	3	0.02%	-2
21	1	0.01%	3	0.02%	-2
22	1	0.01%	0	0.00%	1
23	1	0.01%	0	0.00%	1
24	1	0.01%	0	0.00%	1
25	1	0.01%	7	0.04%	-6
26	1	0.01%	4	0.02%	-3
27	0	0.00%	5	0.03%	-5
28	0	0.00%	4	0.02%	-4
29	0	0.00%	3	0.02%	-3
30	0	0.00%	3	0.02%	-3
31	0	0.00%	3	0.02%	-3
32	0	0.00%	2	0.01%	-2
Total	18,415	100.00%	18,349	100.00%	66

Although the attack involves command word flooding, a number of status words bearing suspicious RT addresses were also observed. These may have been generated by the BC to indicate an error, i.e. the RT does not exist.

The number of command words for each suspicious RT address is also notable, ranging between 1 and 10. If the addresses were randomly generated, a more uniform distribution would be expected. The flooding routine most likely follows a set RT distribution rather than randomly generating an RT address each time. While this may provide some insight into the attacker's tactics and could be used as a fingerprint for attribution, detection is not impacted by the degree of randomness in RT address selection for flood traffic.

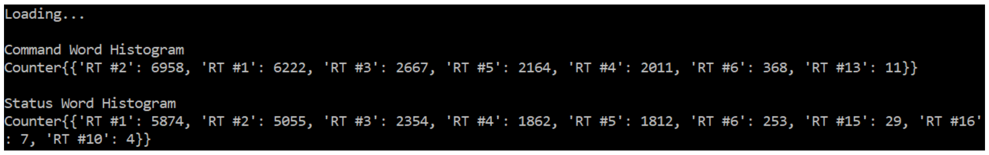
While RT frequency analysis identified the flooding, signature detection can be used to gain further insight into the attack. The RT addresses observed can be used to craft signatures to detect the unexpected command word traffic.

From the signature detection alerts raised, there appear to be two distinct waves of flooding involving the suspect RT addresses. The first begins at packet 18,414 and ends at 19,855, with a suspicious packet appearing roughly every 50th to 100th packet observed. The second wave begins at packet 47,407 and ends at packet 47,489, with a detection occurring on nearly every packet. This may represent a second wave of flooding, shorter but more aggressive.

Opening the pcap file in Wireshark and searching for these packet numbers, allowed us to determine that the first wave begins 22.63 seconds into the capture and ends 0.49 seconds later at the 23.12 second mark. The second wave begins at 32.30 seconds and ends with a detection on the last packet of the file, at 32.36 seconds. It is possible that the second wave of flooding continued past the end of the capture.

5.1.3.2 Status Word Flooding

Similarly to command word flooding, the RT histogram for the capture containing status word flooding in Figure 5.2 shows status word activity from unassigned RT addresses 15, 16 and 10. These figures are also presented in Table 5.3.



```
Loading...

Command Word Histogram
Counter{{'RT #2': 6958, 'RT #1': 6222, 'RT #3': 2667, 'RT #5': 2164, 'RT #4': 2011, 'RT #6': 368, 'RT #13': 11}}

Status Word Histogram
Counter{{'RT #1': 5874, 'RT #2': 5055, 'RT #3': 2354, 'RT #4': 1862, 'RT #5': 1812, 'RT #6': 253, 'RT #15': 29, 'RT #16'
: 7, 'RT #10': 4}}
```

Figure 5.2: RT Histogram for Status Word Flooding

Table 5.3: Distribution of Command and Status Words for Status Word Flooding

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
2	6,958	34.12%	5,055	29.30%	1,903
1	6,222	30.51%	5,874	34.05%	348
3	2,667	13.08%	2,354	13.65%	313
5	2,164	10.61%	1,812	10.50%	352
4	2,011	9.86%	1,862	10.79%	149
6	368	1.80%	253	1.47%	115
13	1	0.00%	0	0.00%	1
15	0	0.00%	29	0.17%	-29
16	0	0.00%	7	0.04%	-7
10	0	0.00%	4	0.02%	-4
Total	20,391	100.00%	17,250	100.00%	3,141

The status word flooding attack appears to use a smaller number of unassigned RT addresses than the command flooding attack. This could be by design, with the attacker favouring the use of assigned RT addresses to blend into the background.

Table 5.3 also shows the effectiveness of the flooding, with a large deficit of status words sent in reply to command words for legitimate RT addresses, as compared to the baseline traffic sample.

Once again using the suspicious RT addresses to craft signatures, only a single wave of flooding was observed. The flooding occurred between packets 12,361 and 17,931, with anywhere from 10 to 300 legitimate packets between each detection. As with command word flooding, some of the flood traffic may contain legitimate RT addresses and can be difficult to separate from the legitimate traffic.

Cross-referencing the packet ID numbers to the packet capture, the first detection was made 15.18 seconds into the packet capture, and the last occurred at the 23.92 second mark, for a total of 8.74 seconds of disruption. The total duration of the capture is 29.27 seconds.

5.1.4 Discussion

5.1.4.1 Command Word Flooding

It should be noted that the maliciously generated command words may contain legitimate RT addresses. Through statistical analysis of multiple traffic captures, it may be possible to estimate the number of spoofed command words containing legitimate RT addresses and to specifically identify them. This analysis has not been done, as it falls outside the scope of the current work.

Should the attacker have previous knowledge of the data bus's construction, it may be possible to carry out command word flooding using only RT addresses that would ordinarily appear in the traffic. In this instance, the attack might not be detectable using the RT histogram method. This could also aid the attacker in flooding the bus more effectively, as each RT receiving a command word typically generates some amount of bus traffic in response. A more detailed statistical analysis of the variations in traffic generation rates of each RT would be required in order to detect this attack variant.

There is also a marked difference in the number of command words for the valid RT address with the lowest rate of incidence (RT #6 with 252 command words) and the malicious RT with the highest rate of incidence (RT #7 with 10). Through the observation of multiple traffic captures containing various lengths of flooding attacks, it may be possible to correlate this gap to the duration of the flooding attack relative to the total length of the capture. Further work would be required to study this.

5.1.4.2 Status Word Flooding

In contrast with the command word flooding technique, no corresponding command words are generated in response to the status words used to flood the bus. Interestingly, this capture does contain a single command word to RT #13. Further investigation is required to determine whether this is part of the attack, a misconfiguration issue, or simply an infrequently transmitting RT present on the bus whose traffic does not appear in the baseline traffic sample.

As with command word flooding, the requirement for statistical analysis in cases where the attacker only uses assigned RT addresses holds, as does the potential correlation between traffic volumes seen for both assigned and unassigned RT addresses and duration of attack.

Also notable is the fact that the first detection has a value other than `000` in the reserved bit field. This is not permitted by the standard. This could be another signature usable to detect flooding traffic, or to help differentiate between legitimate and non-legitimate traffic containing the same RT address where flooding is suspected. Stan et al. also posit that these reserved bits could be used as a covert data transmission channel [3]. This concept will be further explored in the Status Word Data Manipulation scenario in Section 5.4.

5.2 Status Word Data Integrity (Message Manipulation)

5.2.1 Scenario

The Status Word Data Integrity (Message Manipulation) scenario presents a case where an RT sends a status word in response to a command word addressed to a different RT.

Recall from Section 2.1.2.4 that status words can be used to signal an error. Sending a false status word could, for instance, cause the BC to potentially discard valid data.

5.2.2 Expected Detection Mechanism

The hallmark of this scenario is the appearance of two status words sent in response to a command: one from the RT, and a second one sent by the attacker as a fake, after corrupting the legitimate status word [3].

This double status word, the real one and the falsely generated one, should be detectable through Otto's word repetition analysis function. For the attack proposed by Stan et al. to be effective, the false status word would be expected to appear relatively soon after the legitimate status word. However, this method might be prone to false alarms if the RT targeted typically generates a large number of status words in a short period of time.

A secondary possible detection method is a statistical analysis of the RT addresses appearing in the RT histogram. The difference between the number of command and status words containing the targeted RT address would be expected to deviate from the pattern observed in the known-good baseline traffic.

5.2.3 Experimentation

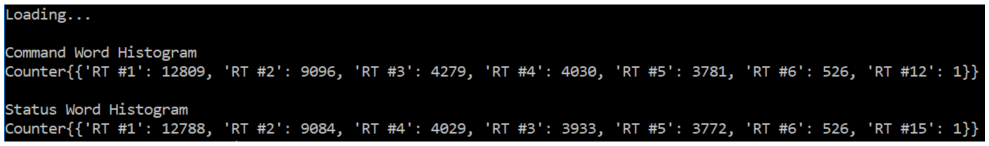
Upon conducting the experiment, it was realized that Stan et. al.'s description of this scenario was misunderstood. As noted in Section 3.1.2, Stan et. al. remark that the duplicate status word corrupts the legitimate status word [3].

This corruption is most likely achieved by attempting to transmit within the same time slot as the legitimate status word. The result would be unintelligible and would not be parsed as a status word by the ENET2-1553. Had Otto been designed to read raw bus data and perform its own interpretation and parsing, it may have been possible to detect this scenario.

Because of this, the proposed detection mechanism, capturing the duplicate status word, cannot work, as there are in fact zero intelligible status words observed as a result of the attack. For the same reason, signature detection is not an option in this scenario as there are no words to compare against the signature list.

RT frequency analysis, however, might be useful in this scenario. Since the status words are quashed, we might see a higher number of command words without reply than was observed in the baseline.

Figure 5.3 shows the histogram generated by Otto, while the compiled distribution of command and status words by RT is displayed in Table 5.4.



```
Loading...

Command Word Histogram
Counter({'RT #1': 12809, 'RT #2': 9096, 'RT #3': 4279, 'RT #4': 4030, 'RT #5': 3781, 'RT #6': 526, 'RT #12': 1})

Status Word Histogram
Counter({'RT #1': 12788, 'RT #2': 9084, 'RT #4': 4029, 'RT #3': 3933, 'RT #5': 3772, 'RT #6': 526, 'RT #15': 1})
```

Figure 5.3: Histogram of Suspicious Bus Traffic (sw_data_integrity.pcap)

Table 5.4: Distribution of Command and Status Words for Status Word Data Integrity

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	12,809	37.10%	12,788	37.47%	21
2	9,096	26.35%	9,084	26.61%	12
3	4,279	12.39%	3,933	11.52%	346
4	4,030	11.67%	4,029	11.80%	1
5	3,781	10.95%	3,772	11.05%	9
6	526	1.52%	526	1.54%	0
12	1	0.00%	0	0.00%	1
15	0	0.00%	1	0.00%	-1
Total	34,522	100.00%	34,133	100.00%	3,895

This is indeed the case. The baseline traffic characterized in Table 5.1 shows 12 of 1,537 RT #3 command words not having a corresponding status word. In other words, 0.78% of all RT #3 command words go unanswered.

In the capture containing the malicious traffic, however, RT #3 has 346 command words out of 4,279 without a status word, or 8.08%. This represents a more than tenfold increase in the relative rate of non-responsiveness for RT #3. Meanwhile, with the CW-SW count for other RTs appears to scale linearly with the overall traffic volumes. These observations suggest something is amiss with RT #3 and further investigation is merited.

To confirm this observation, a second packet capture containing the attack traffic was made. The data from the RT histogram along with the distribution of command and status words is shown in Table 5.5.

Table 5.5: Distribution of Command and Status Words for Status Word Data Integrity (Second Capture)

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	9,320	37.61%	9,305	37.98%	15
2	6,449	26.03%	6,437	26.28%	12
3	2,965	11.97%	2,704	11.04%	261
4	2,908	11.74%	2,908	11.87%	0
5	2,756	11.12%	2,753	11.24%	3
6	380	1.53%	380	1.55%	0
12	1	0.00%	0	0.00%	1
13	1	0.00%	0	0.00%	1
15	0	0.00%	10	0.04%	-10
Total	24,780	100.00%	24,497	100.00%	283

In the second capture, the CW-SW metric for RT #3 again sticks out dramatically, with 261 of 2,965 command words without a corresponding status word, or 8.80%. This is consistent with the observations in the first capture. The CW-SW count for all other RTs again remain consistent with the baseline, scaling linearly with the overall traffic volume.

5.2.4 Discussion

The initial approach of detecting this scenario by flagging the duplicate status word associated with the attack proved not to be possible. It was incorrectly assumed that these status words would appear on the bus, when the scenario in fact relies on corrupting the status word specifically so that it cannot be interpreted.

Because of this word corruption, signature detection is also not a valid approach. Anomaly detection proved to be useful in this scenario, detecting the increase in the delta of command and status words observed.

5.3 Command Word DoS (Behaviour Manipulation)

5.3.1 Scenario

Command word DoS through behaviour manipulation is a straightforward attack. By using "fake commands [...] that are not part of the system's normal operation", an attacker can have an effect on the bus [3].

Stan et al. mention that these commands can be ones defined by the standard, or meaningless, and that the anticipated results can include word collisions, clock synchronization errors and targeted transmitter shutdowns [3].

Of these cases, the scenario involving the use of a transmitter shutdown command was selected for further development. This method is both deliberate and targeted at a specific RT, while the other proposed mechanisms rely more on fortuitous timings to produce effects such as collisions. Further, the concept of using fake commands to cause collisions and affect bus timings was discussed in the Transmission Timings DoS attack discussed in Section 5.1.

5.3.2 Expected Detection Mechanism

Stan et al. suggest that most readily recognizable feature of this command word DoS scenario is the transmitter shutdown command targeting a particular RT [3]. The shutdown command consists of a command word formatted as a mode message, with the corresponding mode code. The attacker may also choose to send a second mode message to re enable the targeted RT's transmitter at the conclusion of the DoS. [25].

Transmitter shutdown and start-up commands are not typically seen on the data bus in normal operations. The most common use of these commands is to shut down an RT that is malfunctioning or damaged and producing unintelligible data. This would require the bus designer to build logic into the BC to verify the output of the RT and take the shutdown action if required.

With this knowledge of how the DoS is performed, it is possible to write signatures to detect occurrences of transmitter shutdown and start-up commands on the bus. By detecting both, an analyst can identify a specific period of time during which communications from the targeted RT have been impaired. As these occurrences are expected to be rare, they can be reliably used as indicators of this form of attack.

5.3.3 Experimentation

The transmitter shutdown and start-up commands utilize the command word's mode functionality. Recall from Section 2.1.3.4 that mode messages are indicated by using a reserved value in the subaddress field, and the mode is determined by the value in the word count field.

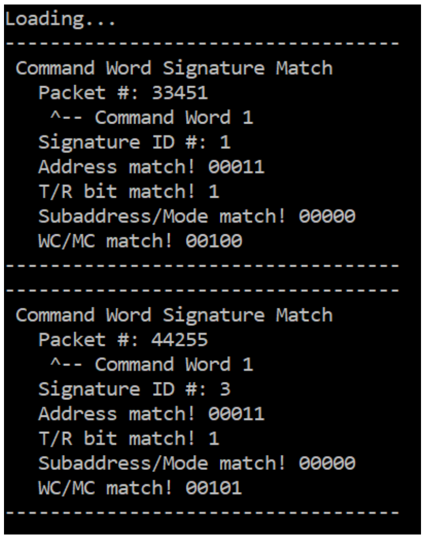
There are two command words that will initiate a transmitter shutdown: one for the entire RT, and one that selectively targets a particular subaddress on the specified RT. Similar general and selective mode messages exist for the transmitter shutdown override commands. These command words are shown parametrically, in order, in Table 5.6.

Table 5.6: Transmitter Shutdown Command Word Signatures

Address (0-31)	TR Bit (T or R)	Subaddress / Mode (0 to 31)	Word Count / Mode Code (0 to 31)	Bus (A or B)
	1	0/31[4]	4	
	1	0/31	20	
	1	0/31	5	
	1	0/31	21	

Running Otto against the baseline traffic sample using these command word signatures raises no detections. However, when the same signatures are run against the traffic during an active command word DoS session, two detections are made, shown in Figure 5.4.

[4]Both mode indicators are shown here for brevity. Two separate signature lines are required to cover both mode indicators when crafting signatures for Otto.



```
Loading...
-----------------------------------
 Command Word Signature Match
   Packet #: 33451
    ^-- Command Word 1
   Signature ID #: 1
   Address match! 00011
   T/R bit match! 1
   Subaddress/Mode match! 00000
   WC/MC match! 00100
-----------------------------------
-----------------------------------
 Command Word Signature Match
   Packet #: 44255
    ^-- Command Word 1
   Signature ID #: 3
   Address match! 00011
   T/R bit match! 1
   Subaddress/Mode match! 00000
   WC/MC match! 00101
-----------------------------------
```

Figure 5.4: Detection of Transmitter Shutdown and Override Transmitter Shutdown Mode Commands

The first detection is a transmitter shutdown command in packet #33,451 of the capture, addressed to RT #3. The second is an override transmitter shutdown command in packet #51,225 again sent to RT #3.

The detection can be confirmed by loading the same packet capture file into Wireshark, filtering on the identified packet and using the Alta-provided Wireshark plugin to interpret the Alta CDPs.

The timing data from Wireshark can also be used to identify the duration of the disruption. The shutdown command occurred 41.41 seconds into the capture, while the override command followed at the 63.41 second mark, for a total disruption lasting 22.27 seconds.

5.3.4 Discussion

Otto's signature detection functionality was successful in identifying the hallmarks of the command word-initiated DoS. The packet number data reported by Otto was also useful in pinpointing the precise times the targeted RT's communications were impaired.

While this experiment focused on the transmitter shutdown and shutdown override commands, Stan et al. also suggest that other commands can be used to affect the system's proper operation. The examples provided include using legal or illegal commands to cause collisions and impair the bus functionality, improperly synchronizing clocks to create incorrect timings [3], or issuing other commands. If the makeup of the command word used (legal or not) is known, a signature can be written to detect it, as was done in this experiment. However, consideration must be given to the potential for false alarms. While transmitter shutdown and shutdown override commands are rarely seen on a normally functioning bus, some of these other commands suggested by Stan et al. may be more commonplace. Separating attack events from regular traffic may require an additional level of triage, done either manually by an operator or automatically by a new Otto function. Detection of the use of command words to flood the bus was covered in the transmission timings scenario discussed in Section 5.1.

Actions related to timing, including skewed clock synchronization are not currently supported in Otto, but it may be possible to expand Otto's functionality to cover these (see Section 5.7).

5.4 Status Word Data Manipulation (Message Manipulation)

5.4.1 Scenario

MIL-STD-1553B explicitly states that the reserved bits in a status word are to be set to zero for a word to be valid [1]. However, it is not compulsory for this to be checked. Stan et al. suggest that these three bits are ideal for surreptitiously passing data over the bus. This would most likely be done by an RT that has been somehow compromised by an attacker.

5.4.2 Expected Detection Mechanism

As discussed earlier, the MIL-STD-1553B standard requires the reserved bits in every status word to be set to zero. As status words with any other value in the reserved bits are non-compliant with the standard and suspicious, signature detection can be used to flag such cases. This check effectively eliminates a key enabler to this scenario, namely the "lack of status word monitoring" mentioned by Stan et al. [3, 25].

5.4.3 Experimentation

In order to test the proposed detection mechanism, bus data containing status words with non-zero values in the reserved field is required. Fortunately, the packet capture used to study Status Word Denial of Service in Section 5.1 was found to contain such traffic.

In order to detect these values, status word signatures are required. As this is a three-bit field with only one acceptable value, seven signatures are required to cover the illegal cases we seek to detect. These are shown in Table 5.7.

Table 5.7: Non-Zero Reserved Bit Status Word Signatures

Address	E	I	SR	Reserved	BC	B	SF	DBCA	T	Bus
				1						
				2						
				3						
				4						
				5						
				6						
				7						

Running these signatures against the capture file confirms that the file does indeed contain status words with a non-zero reserved bit. A total of 489 detections were made, with 488 having a value of 7 (`0b111`) and one with a value of 5 (`0b101`).

5.4.4 Discussion

Ordinarily, detecting the presence of non-zero bits in the reserved field would be the starting point of a deeper investigation to determine what data is being sent by which RT. However, because the data used in this scenario contains randomly generated status words, there is no further usable information to extract. However,the successful detection of the non-zero words is enough to detect signs of data exfiltration by this method.

This scenario also raises a potential future improvement to Otto, namely value whitelisting. While Otto's current signature detection logic reports matches between the signature file and the bus traffic, this example demonstrates a case where it would be simpler to raise a detection on any traffic that does not match a given value. In such a case, only one signature would be required to detect non-zero traffic, instead of seven to detect all possible scenarios for illegal use of the three-bit reserved field.

5.5 Malfunctioning RT

5.5.1 Scenario

This scenario aims to detect a case where an RT simply malfunctions without having been compromised by an attacker. Automated detection of malfunctioning RTs would be useful to aircraft maintenance organizations as a troubleshooting tool. It could also serve bus designers by confirming that all bus-connected devices are behaving as expected [25].

This scenario will demonstrate a detection based on a comparison to known good baseline traffic. Changes in the number of command or status words observed for a given RT address, or changes in the RT response (i.e. command words without a corresponding status word) can be used as indicators of a malfunctioning RT [25].

5.5.2 Expected Detection Mechanism

In this scenario, detection will be based on a comparison to a baseline traffic capture. The number of command and status words for each RT as a function of the total number of command and status words observed in the baseline capture will be calculated. The same will be done for the traffic capture under test. Variations in the ratios of command and status words to total words observed may be indicative of a malfunctioning RT. Changes in the RT response, i.e. command words without a corresponding status word, may also be considered indicative of anomalous behaviour [25].

5.5.3 Experimentation

The FOUR_RT scenario from Section 4.4 was used as a starting point for this experiment. In order to have a baseline for comparison, Otto was used to gather baseline data for the command and status words seen in the capture. This breakdown was previously presented in Table 4.7 and is reproduced below in Table 5.8.

Table 5.8: Distribution of Command and Status Words for the FOUR_RT Capture (Reproduction of Table 4.7)

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	121	29.88%	121	30.17%	0
2	90	22.22%	88	21.95%	2
3	97	23.95%	95	23.69%	2
4	97	23.95%	97	24.19%	0
Total	405	100.00%	401	100.00%	4

As the FOUR_RT capture made in Section 4.4 is short (8 seconds), a second capture of traffic generated by the FOUR_RT scenario was taken, lasting 497 seconds and called FOUR_RT_LONG. This longer traffic capture was similarly analyzed and the results are shown in Table 5.9 [25].

Table 5.9: Distribution of Command and Status Words for the FOUR_RT_LONG Capture

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	7,458	30.20%	7,458	30.53%	0
2	5,470	22.15%	5,337	21.85%	133
3	5,966	24.16%	5,834	23.88%	132
4	5,802	23.49%	5,802	23.75%	0
Total	24,696	100.00%	24,431	100.00%	265

In comparing the traffic volumes for each RT address to the total traffic observed for each word type, no large deviations were noted. While the percentages observed vary by up to approximately 0.5%, this can be explained by the random start and stop times of the data capture. Packet captures were launched while the BusTools scenario was running and stopped after an arbitrary amount of time, rather than launched simultaneously to the scenario start and stopped after a prescribed number of cycles. The partial cycles observed are the likely cause of this discrepancy [25].

Given its longer run time, the effects of partial cycles can be assumed to be less significant in the FOUR_RT_LONG capture. Therefore, FOUR_RT_LONG will be used as the baseline for comparison in the experimentation [25].

Simulation of a malfunctioning RT was done by modifying the FOUR_RT BusTools script. Two new scenarios were created to represent cases where an RT generates both more and less traffic than expected [25].

5.5.3.1 FOUR_RT_FASTER

The FOUR_RT_FASTER scenario sees an additional command word added to the scripted bus interactions: a mode message requesting a status word, sent to RT #3. This should result in one additional command word and one additional status word containing the RT #3 address for every iteration of the script [25].

Table 5.10 shows the breakdown of the traffic observed in this capture.

Table 5.10: Distribution of Command and Status Words for the FOUR_RT_FASTER Capture

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	1,553	29.58%	1,553	29.89%	0
2	1,141	21.73%	1,114	21.44%	27
3	1,347	25.66%	1,319	25.39%	28
4	1,209	23.03%	1,209	23.27%	0
Total	5,250	100.00%	5,195	100.00%	55

For both command and status words, the expected increase in traffic for RT #3 is observed. The percentage of total traffic observed for RT #3's command and status words increased by approximately 1.5% (recall proportions of 24.16% and 23.88% respectively from the original FOUR_RT_LONG capture). The corresponding decrease in traffic observed was shared equally by the remaining RTs, with each showing a drop of approximately 0.5% [25].

The relative increase in traffic for RT #3 suggests that this RT is functioning differently than in the baseline. Given the increase is observed for both command and status words, it can be assumed that the RT is being issued additional commands resulting in the generation of status words. This information can help the analyst better pinpoint the cause of the additional traffic.

An increase in command words only points to these command words not requiring a status word response, while an increase seen only for status words suggests additional status words are being generated without a corresponding command word [25].

5.5.3.2 FOUR_RT_SLOWER

To simulate an RT generating less traffic than expected, the FOUR_RT script was modified to delete a status word from RT #3 used to signal the results of a power-on built-in test (PBIT). The expected effect is fewer status words seen containing RT address #3 [25].

The traffic observed in this capture is broken down in Table 5.11.

Table 5.11: Distribution of Command and Status Words for the FOUR_RT_SLOWER Capture

RT Address	Command Words (CW)	% of total	Status Words (SW)	% of total	CW - SW
1	1,835	30.19%	1,835	31.80%	0
2	1,346	22.14%	1,313	22.76%	33
3	1,346	22.14%	1,192	20.66%	154
4	1,420	23.52%	1,430	24.78%	0
0	122	2.01%	0	0.00%	122
Total	6,079	100.00%	5,770	100.00%	309

There is one other apparent deviation from the baseline: 122 command words destined for RT #0. This RT was not assigned in the scenario, and no traffic with this address was observed in the baseline traffic captures [25].

In baselining, the difference in the number of command words and the number of status words for RT #2 and RT #3 are typically similar. Meanwhile, in this scenario, RT #2 showed 33 command words without corresponding status words, while RT #3 showed 154. However, if the 122 command words for RT #0 are subtracted from RT #3's tally, the total is 32, which is nearly identical to RT #3, matching the baseline [25].

Further analysis is required to understand this and confirm that it is an artifact of the removal of the PBIT status word from the FOUR_RT_SLOWER script, but this hypothesis appears valid [25].

Moving on to the total number of status words seen, there is a drop from 23.88% to 20.66%. There is also a corresponding drop in the number of RT #3 command words seen, from 24.16% of total traffic to 22.14%, however if the hypothesis surrounding RT #0 is correct, the share would increase to 24.15%, which does match the baseline [25].

5.5.4 Discussion

In the malfunctioning RT scenario, two cases were presented. The first was of an RT generating more traffic than what was observed in the baseline (Section 5.5.3.1), and the second was for an RT generating less traffic (Section 5.5.3.2). In each case, the deviation was slight, with one command addition or removal from a script of approximately 35 actions [25].

The extra command added in the FOUR_RT_FASTER capture at Section 5.5.3.1 actually resulted in two additional words being generated: the actual command word, and a status word generated as a reply to the former. The impact was readily noticeable in the comparison to the baseline, with the affected RT showing an increase in both command and status word traffic of 1.5% over the baseline. The proportion of traffic observed for all other RTs had a corresponding drop totaling 1.5%, suggesting that they continued to perform as expected while RT #3 generated additional traffic. This data would provide a sufficient starting point for a data bus designer or aircraft maintainer to begin investigating the issue [25].

The FOUR_RT_SLOWER capture at (Section 5.5.3.2) had an unexpected result from the outset, with 122 command words generated containing the RT #0 address. While further investigation is required, it appears that this was an unintended consequence of removing the status word used to report the result of the PBIT. Putting this observation aside, the analysis shows a readily noticeable decrease in the relative number of RT #3 status words to 20.66% against the 23.88% observed in the baseline. This deviation should again provide an indication to analysts that RT #3's characteristics no longer match the baseline observations [25].

All data sets used in this experiment were simulations of fairly small data buses following a repeating script. Actual aircraft data buses are expected to be more saturated and less predictable in nature. The concept demonstrated in this experiment should scale to these larger buses, but additional care may be required to compare data from similar conditions, e.g. similar phases of flight, aircraft attitude, and equipment configuration [25]. This notion will be further expanded upon in Section 5.7.

The analysis conducted in this scenario was conducted manually using the histogram data reported by Otto in order to prove the concept. In future versions of Otto, this baseline comparison could be automated in order to generate clear alerts to the user when the traffic under study deviates significantly from the baseline. This could also present an opportunity to implement more advanced statistical analysis and detailed reporting [25].

5.6 Unassigned RT Address

5.6.1 Scenario

The unassigned RT address scenario posits a case where an RT is added to the bus without the knowledge of the bus operator. Because the MIL-STD-1553B standard has no provisions for device authentication, such a device would be able to freely send and receive data over the bus.

This scenario presumes that the rogue RT actively communicates over the bus without attempts to conceal its identity or spoof other devices: it does not send command words, only responds to commands addressed to it, and does not provide a false RT address in the status words it sends out. Attacks carried out by an RT using deception techniques to blend into the normal bus traffic would fall into other scenarios, such as the Status Word Data Integrity (Message Manipulation) attack discussed in Section 5.2.

While the first four scenarios presented in this chapter deal in activity that could be malicious in nature, this scenario is more likely to be a configuration error. Potential causes could include an RT being replaced with the wrong component, or an RT configured with the wrong address.

5.6.2 Expected Detection Mechanism

The aim of this scenario is to detect MIL-STD-1553B data bus traffic containing an RT address that is not known to exist on the bus. Such traffic could be a result of a misconfiguration, or of an RT added to the bus without the designer's knowledge [25].

The detection of the unassigned RT will be made using the RT histogram method. As the histogram tallies all command and status word traffic on the bus for each RT, it can be used as a record of all RT addresses appearing in the traffic capture [25].

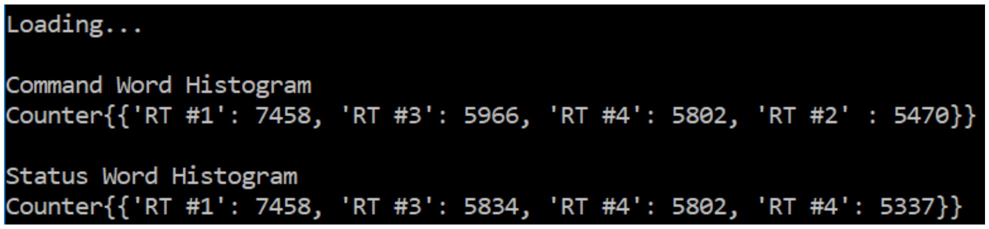
5.6.3 Experimentation

As with the malfunctioning RT scenario, this scenario was developed on a simulated data bus, using the FOUR_RT example from GE BusTools as a starting point. A capture of data bus traffic was taken to use as a baseline: the same FOUR_RT_LONG.pcap capture used in the previous section.

The FOUR_RT script was copied as UNASSIGNED_RT and modified in order to add an extra RT, given the address #12. This represents the scenario where a device is unexpectedly added to the bus [25].

The interaction script for the scenario was also modified to cause the RT to issue a command word to RT #12. This command is a mode message, requesting a status word as a reply. This scenario was then played and traffic was captured to `unassigned_rt.pcap` [25].

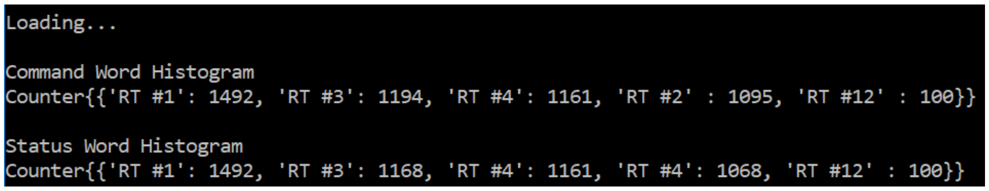
The FOUR_RT scenario is known to only contain RT #1, #2, #3, and #4. This is shown in Figure 5.5, where the output of Otto's histogram contains only the expected addresses [25].



```
Loading...

Command Word Histogram
Counter{{'RT #1': 7458, 'RT #3': 5966, 'RT #4': 5802, 'RT #2' : 5470}}

Status Word Histogram
Counter{{'RT #1': 7458, 'RT #3': 5834, 'RT #4': 5802, 'RT #4': 5337}}
```

Figure 5.5: Histogram of Known Good Traffic Capture (FOUR_RT_LONG.pcap)

Using Otto to generate the RT histogram for the UNASSIGNED_RT scenario (Figure 5.6), it is readily apparent that both command and status words were observed containing the RT #12 address [25].



```
Loading...

Command Word Histogram
Counter{{'RT #1': 1492, 'RT #3': 1194, 'RT #4': 1161, 'RT #2' : 1095, 'RT #12' : 100}}

Status Word Histogram
Counter{{'RT #1': 1492, 'RT #3': 1168, 'RT #4': 1161, 'RT #4': 1068, 'RT #12' : 100}}
```

Figure 5.6: Histogram of Suspicious Bus Traffic (unassigned_rt.pcap)

This raw information should be sufficient for a technician to conclude that something is amiss. More detailed analysis could include correlating the number of command words to status words, or searching through the capture file to understand what is happening. However, this more detailed analysis is beyond the scope of this proof of concept focusing on detection [25].

5.6.4 Discussion

In the unassigned RT scenario, a detection was made based on knowledge of the addresses of all RT addresses present on the bus. As the bus can contain a maximum of 32 bus connected devices, a bus designer could reasonably be expected to include an RT address list in the system documentation [25].

In the case of a black box system where no design information is known prior to analysis, a baseline analysis can be done. By observing the bus traffic with the system in a known-good state, a picture of the bus is formed. This includes a list of present RTs, the ratio of command words to status words, and the relative frequency of command and status words for each RT [25].

It may be possible for baselining to miss a given RT, such as one that receives very infrequent traffic, or is only queried in a particular phase of flight. Such an RT could cause a false positive if it appears in the sample under analysis but not the baseline. Care must be taken to either gather comprehensive baseline data across all flight regimes and a sufficiently long time period, or to compare traffic samples only from similar phases of flight [25].

With knowledge of the system's design, another possible detection method would be the use of Otto's signature matching functionality to generate alerts when command and status words containing unassigned RT addresses appear on the bus. The baseline comparison method was selected for this experiment as it is a more flexible option, not requiring prior knowledge of the bus's composition [25].

Finally, a baseline analysis function to flag new RTs could be used to automate some of the comparisons, either as a standalone function or as part of an existing mechanism.

5.7 Discussion of Validation Results

While previous sections of this chapter contained detailed discussions of each of the demonstrated scenarios, this section will present a more general discussion of points applicable to the application of network security monitoring techniques to the MIL-STD-1553B data bus.

The work presented thus far in this chapter demonstrates that while signature-based detection is valuable, it is not a perfect solution. Some scenarios were only detectable using anomaly-based techniques. Though tools such as MAIDENS use more advanced techniques than those implemented in this work, there is still a valuable role for signature-based detection techniques in this space.

5.7.1 Effectiveness of Signature-Based Detection

Signature-based detection was the primary detection method used in two scenarios: Command Word DoS (Behaviour Manipulation) and Status Word Data Leakage (Message Manipulation).

In the former case, anomaly-based methods such as RT frequency analysis could have picked up on the fact the targeted RT was producing comparatively less data than expected. However, basic RT frequency analysis would not have yielded an exact time the bus was disrupted. Even though more advanced anomaly detection tools such as MAIDENS are able to estimate the disruption time, detecting the transmitter shutdown and override commands the moment they are sent gives a more accurate picture.

Detecting the mode commands would also produce clear detections in cases where the RT is only disabled for very short periods of time. Very short disruptions may not cause a significant enough deviation from the baseline for an anomaly-based solution to raise a detection, particularly if these blanking periods are spread out over time.

In the case of Status Word Data Leakage (Message Manipulation), signature detection can provide a clear and unambiguous alert whenever the reserved bits are set to non-zero values.

The first case, Transmission Timings DoS (Behaviour Manipulation), gives an example of how signature detection can be used in support of other detection methods as an investigative tool. RT frequency analysis was the primary means of detection for this scenario, uncovering traffic to RT addresses not present on the bus. These RT addresses were then used to produce signatures, allowing the analyst to determine when the attacks began and ceased, and how long they lasted.

This is also true of the Unassigned RT Address scenario, where RT frequency analysis and comparison to the baseline was the primary detection method. However, signature detection can also be applied by creating signatures to alert on addresses known not to be in use. Such alerts would be clear and unambiguous, but require advance knowledge of the bus's configuration and signature updates if the bus layout changes.

In the Malfunctioning RT scenario, there is nothing for signature detection to offer. The scenario proposed an RT producing the expected traffic, but at increased or decreased time intervals. In this case, anomaly detection is the only means of making a conclusive definition.

Similarly, signature detection is ineffective against the Status Word Data Integrity (Message Manipulation) technique. Even if the duplicate status word followed a known pattern that could be used to create a signature, it would not be detected as the result of a successful attack is an unintelligible word that cannot be interpreted.

5.7.2 Effectiveness of Word Repetition Analysis as a Detector

Word repetition analysis was proposed as a method for making a detection in the Status Word Data Integrity (Message Manipulation) scenario, where a duplicate status word was expected. Since there is in fact no status word interpreted by the ENET2-1553, this approach was not feasible.

Although attempting to apply this method yielded no success, it was noted that the false alarm rate was far higher than was observed in testing. While the bus configurations used in testing the detector were lightly loaded with only a handful of RTs transmitting in slow time, the test data was captured on a far busier bus, with much more traffic and therefore many more routine acknowledgment status words.

It was incorrectly assumed that the expected double status word was detectable, but it is lost in the sea of routine acknowledgment status words. Even with the knowledge that the double status word does not appear, the false alarm rate observed was unexpectedly high.

The word repetition analysis routine added all status words to a single list, regardless of source address. Trying to measure repetition in this list is not viable, as some RTs are polled more frequently than others. To catch word repetition from a relatively quiet RT, many more repetitions would be observed from RTs broadcasting more frequently.

To solve this, a number of lists would need to be maintained: one for each RT. This would result in a more reasonable comparison, looking only for word repetition in a single RT's traffic, and discounting other RTs that may be repeating frequently by design.

5.7.3 Traffic Capture

This work identified three main considerations for capturing bus traffic: capture length, system conditions and format.

5.7.3.1 Capture Length

When capturing traffic either as a baseline or for analysis, every effort should be made to ensure the capture is comprehensive enough to accurately represent normal bus activities. The best way to maximize the chance of this is to take a long capture. The threshold for a long capture may be situationally dependent, but several minutes of data is more likely to be representative than a few seconds.

Taking longer captures also reduces the prominence of the start and stop points. For instance, starting a capture at the precise moment after a command is sent but before the RT has replied would produce an imbalance in the number of command and status words recorded, with one more status word seen. If only ten command/reply pairs are observed, this extra status word is significant. But if hundreds or thousands of iterations are recorded, a single extra word becomes much more negligible.

5.7.3.2 System Conditions

Another important aspect to consider when taking a traffic capture is ensuring the captures were taken during representative phases of the system's operation.

Over the course of a typical flight, the type of traffic that appears on the bus will vary depending on conditions. For instance, takeoffs and climb outs require changing throttle settings and control surface actuation, while cruise typically requires less frequent control input. What is "normal" and "expected" therefore varies depending on the regime of flight. An analyst (or automated tool) may need to account for this in making a determination.

Similarly, when an analyst compares a traffic sample to a baseline, knowing the phase of flight in which each was captured may be relevant. For instance, a command to the RT and subaddress responsible for landing gear deployment will most likely only appear on samples captured during the climb or approach phases. If the command appears in the sample under analysis but not the baseline, is it suspicious, or was the baseline recorded during cruise?

As an example, recall that in the Transmission Timing DoS (Behaviour Manipulation) scenario discussed in Section 5.1, a command word for RT #13 was observed in what was otherwise status word flooding. While this might be a result of the flooding, it is also possible that it was a legitimate command word appearing so infrequently that it was only observed once. While this is unlikely, it cannot be discounted without knowing the exact layout of bus.

In the traffic samples used to test Otto in Section 4.4 and some of the cases used to validate the tool in Chapter 5, the traffic samples generated using BusTools was scripted, repeating the same known sequence of actions. Such cases lend themselves well to baselining, as the conditions are the same for both the baseline and the suspect capture. The Malfunctioning RT and Unassigned RT Address scenarios demonstrate the utility of the technique when this is properly done.

5.7.3.3 Capture Format

A more fundamental issue related to bus traffic recording is the format of the capture. While the simplest method is to directly monitor the bus's electrical activity and record the bits as they are observed, this work exploited the Alta ENET2-1553's bus monitoring feature, which is able to translate MIL-STD-1553B traffic into a packetized format. Leveraging this existing technology provided advantages such as metadata generation and compatibility with existing IP protocol analysis tools. However, this convenience came at the cost of added implementation complexity, requiring Otto to interpret UDP/IP packets and extract data from a payload structured according to an obscure protocol. With the concept now proven, future projects in this domain may opt to capture traffic differently. Capturing raw data from the bus has the advantage of being more direct, but may require additional development work to create tools required to support further analysis.

5.7.4 Timing Attacks

When selecting a subset of the attacks proposed by Stan et al. to test Otto's effectiveness, attacks involving timing were not considered. Examples of these include asynchronous messages, non-respect of the intermessage gap and improper clock syncing [3].

While these attacks may be as insidious as any other discussed in Chapter 5, an emphasis was placed on attacks requiring the words transmitted over the bus to be analyzed and interpreted. This investigation of command and status words at the bit level was judged to be more directly linked to the network security monitoring techniques proposed. However, observing, tracking and correlating event timings on a busy bus, while complex, can yield valuable investigative information and presents a good opportunity for future study.

5.7.5 Attack Feasibility and Effectiveness

Early in this work, an underlying assumption was made that an attacker would be able to gain any prerequisite foothold to carry out the attacks studied. While it is possible to hypothesize vectors for initial compromise, the feasibility of these was not considered. This was a deliberate decision, as there can be many ways for an adversary to inject data onto the bus. The absence of device authentication in the MIL-STD-1553B standard makes it as simple as connecting a rogue device to a data bus port. As prevention of attacks is strongly preferable to their detection, the study of initial compromise methods is ripe for further development.

Similarly, this work presents no evaluation of the effectiveness of the attacks proposed by Stan et al., or the likelihood of such attack occurring [3]. While some work has been done in this realm, the attacks proposed were assumed to be effective and all equally likely to be used by an attacker. As attack tactics, techniques and procedures (TTPs) are further developed, detection techniques will need to be similarly refined in order to increase detection rates, reduce false alarms, and further automate analysis. This underscores the importance of studying the "offensive" side of security in order to better prepare the defensive side.

5.7.6 Prevention vs Detection

Finally, it bears noting that this work focuses on the detection of an attack or malfunction in progress. In the case of recorded traffic, the methods discussed provide a forensic capability for determining what occurred.

With a focus squarely on detection, this work does not provide any preventative measures to avoid undesired effects on the bus, nor does it take action to stop any actions it detects. While detection is certainly important, the importance of prevention should not be understated. It is suggested that methods for protecting the MIL-STD-1553B data bus from unauthorized access be considered as an important avenue for future work in this domain.

6 Conclusion

Throughout the development of this work, numerous opportunities for future work have presented themselves. While these are beyond the scope of a proof of concept, they can be used to guide the way forward in this space.

This chapter will summarize these potential areas for advancement, and will end with concluding remarks recapitulating the work done.

6.1 Future Work

While Otto is a sufficiently functional proof of concept, a number of avenues for future work have been identified over the course of the development. While they are discussed within the context of the proof of concept tool developed in the course of this work, many of these enhancements have applicability to more general MIL-STD-1553 network security monitoring concepts.

Firstly, while some cursory testing has been done on different operating systems, Otto was designed to run on its development station, namely the default Python installation from Kali Linux 2018.1 running on x64 architecture. Some incompatibility with other platforms was noted in the testing phase, likely due to unmet dependencies. While such issues are acceptable for a proof of concept, these should be addressed if Otto is to be developed further.

Notwithstanding the discussion on traffic capture format presented in Section 5.7.3.3, it is possible to better leverage the existing APMP CDP-formatted data. Specifically, there is a considerable amount of metadata contained within each packet. While Otto uses the CDP's A/B bus flag as a signature parameter, it may be possible to use other fields such as CDP error detection flags or intermessage gap times to create richer signatures, or to devise new detection methods.

While the prototype does some basic checking to reject null-value CDPs, these tests could stand to be made more robust. Firstly, the current checking only tests the fields extracted from the CDP 1553 word, and not the entire CDP word. While this is not expected to impact any of the test cases studied in Chapter 5, it could result in falsely rejected command words that, though invalidly formatted, might be generated by an attacker. Re-implementing the check to compare the full 32-bit CDP 1553 word to the `0xFFFFFFFF` null value rather than a selection of fields would eliminate this possibility. This checking could also more efficiently be performed at the packet parsing stage, rather than within both the signature detection and word repetition analysis phases.

On a related note, Otto's current structure assumes that the pcap files provided contain only APMP CDP-formatted data, and parses all packets accordingly. Pre-filtering is done manually using `tcpdump` to include only UDP traffic from the ENET2-1553's IP address and specified port. Building this filtering functionality into Otto would eliminate the need to manipulate pcap files, and would be required to ingest a stream in near-real time to ensure only APMP-formatted data is parsed.

Near-real time processing of MIL-STD-1553 data is also a highly desirable option. While it should be feasible to do this using the packet stream produced by the ENET2-1553, successful implementation may require improved efficiency. This is especially important for analysis of a bus over a long period of time, as even small delays will agglomerate into noticeable lag. Alternate methods for capturing MIL-STD-1553 bus traffic may also lead to better performance.

Related to this, as mentioned in Section 4.2.3.3, the signature detection logic used in the prototype version of Otto was designed to be human-readable, and not for computational efficiency. Additional work to better streamline this algorithm, along with enhancements elsewhere in software, has the potential to improve Otto's performance.

Improvements to Otto's detection handling function could also improve computational efficiency. Currently, every available MIL-STD-1553 word field is passed as an individual argument to the function. Those fields not relevant to the detection are still passed, filled with throwaway values. Instead, passing the packet as a single variable of an appropriate class would make this more intuitive. Improvements to the user output are also possible, giving more data to aid in deeper analysis. However, the alerts currently generated are sufficient to prove the concept.

In the same vein, a mechanism for the operator to fine-tune the generation of alerts could improve detection rates by eliminating clutter. An example of this was discussed at the end of Section 4.4.3, where routine status words were frequent enough to raise word repetition alerts. Such an improvement may have also been beneficial in the Status Word Data Integrity (Message

Manipulation) scenario, where the failure to detect may have been due in part to such alerts.

In addition to the requirement for conducting the analysis on a per-RT address basis, the RT frequency analysis function could also be enhanced by automating baseline comparisons. Currently, the user must compare the results of two traffic captures manually. By adding the ability to input multiple packet capture files, it would be possible to conduct the required calculations and present a side-by-side comparison. Multiple file input could also be used to conduct signature detection on multiple files in a single operation, and generate a unified report.

The signature detection logic was designed to use blacklisting: if a value appears in the signature, it generates an alert. The implementation of whitelisting, i.e. raising an alert on any value except the one in the signature, would eliminate the need for the analyst to list every possible anomalous value when only a single value is valid. An example of this is the reserved field in the MIL-STD-1553 status word. This concept was illustrated in the Status Word Data Manipulation (Message Manipulation) scenario discussed in Section 5.4.

Looking away from Otto and to the broader state of the art, a number of other detection techniques could be adapted to MIL-STD-1553 data. For instance, the Markov modeling proposed Stan et al. has promise for automated anomaly detection [3]. State monitoring, as briefly discussed in Section 5.2 is another option, albeit one that may be computationally expensive.

Overall, while the present work does prove the concept of applying signature-based network security monitoring techniques to the MIL-STD-1553B data bus, there are a number of opportunities for future work. Development of new detection techniques and improved implementation of existing ones are two broad avenues for ongoing research and development.

6.2 Conclusions

As initially presented in Section 1.2, this work set out to demonstrate an application of signature-based detection to MIL-STD-1553B data bus traffic to identify signs of undesirable system activity. Detection of such traffic is critical to ensuring that data bus-connected devices continue to operate as designed, and are free from compromise or faults.

Starting with samples of bus traffic encoded using the Alta Common Data Packet protocol, a tool called Otto was written in Python to implement signature detection, as well as the required packet dissection functions and user interfaces. Two anomaly-based detection techniques, word repetition analysis and RT frequency analysis, were implemented for use in cases where signature-based detection may not be effective.

The testing phase revealed a number of coding and logic errors, which were corrected. The updated version of Otto performed as expected and was ultimately used to carry out testing against sample traffic demonstrating attacks postulated by Stan et al. [3].

As demonstrated in Chapter 5, signature-based analysis was ultimately successful at making detections in two of the selected scenarios. In two others, signature-based detection was used to support the investigation of an anomaly-based detection. In the two remaining scenarios, signature detection was not useful.

The success of this work further opens the door to further research in this space. Opportunities exist to continue the refinement and automation of the detection techniques demonstrated, or to develop and implement new techniques. There is also a need for continued research into implementing attacks against the bus, in order to better inform the development of detection techniques. Finally, there are also opportunities to continue research into incident handling and active bus defense.

Although MIL-STD-1553 is an aging standard, it continues to be in service in both legacy and new aeronautical platforms. The inherent design of the data bus can make it a potential target for malicious actors. While much of the regulatory activity surrounding aircraft systems has been prompted by the introduction of more modern computing and network technologies, continued work in this domain is important to help ensure safe flight.

References

[1] Aircraft internal time division command/response multiplex data bus. Technical Report MIL-STD-1553B, United States Department of Defense, September 1978.

[2] Jeremy Paquet. Uncovering MIL-STD-1553 vulnerabilities: Exploitability of military aircraft networks. Master's thesis, Royal Military College of Canada, 2014. [SECRET].

[3] Orly Stan, Yuval Elovici, Asaf Shabtai, Gaby Shugol, Raz Tijochinski, and Shachar Kur. Protecting military avionics platforms from attacks on MIL-STD-1553 communications bus. Technical report, Ben-Gurion University of the Negev and Astronautics C.A. ltd., July 2017.

[4] Cary R. Spitzer. *The Avionics Handbook*. CRC Press, Williamsburg, VA, 2001.

[5] Charles Bernard. Application of network monitoring concepts to the MIL-STD-1553B data bus. Technical report, Royal Military College of Canada, Kingston, ON, 2015.

[6] Fibre optics mechanization of an aircraft internal time division command/response multiplex data bus. Technical Report MIL-STD-1773, United States Department of Defense, 1988.

[7] Richard Bejtlich. *The Practice of Network Security Monitoring: Understanding Incident Detection and Response*. No Starch Press, San Francisco, CA, 2013.

[8] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, March 2000.

[9] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. In *IEEE Communications Surveys & Tutorials*, volume 16, pages 303–336, 2014.

[10] T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. In *IEEE Communications Surveys & Tutorials*, volume 10, pages 56–78, 2008.

[11] P. Garcia-Teodoroa, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. In *Computers & Security*, volume 28, pages 18–28. Elsevier, 2009.

[12] George Khalil. Open Source IDS High Performance Shootout. Whitepaper, SANS Institute InfoSec Reading Room, Bethesda, MD, February 2015.

[13] Nick Ierace, Cesar Urrutia, and Richard Bassett. Intrusion prevention systems. *ACM Ubiquity Magazine*, page 2, June 2005.

[14] Special conditions: Boeing model 787-8 airplane; systems and data networks security — isolation or protection from unauthorized passenger domain systems access. In *Federal Regsiter*, volume 73, pages 27–29, Renton, VA, January 2008. Federal Aviation Administration, United States Department of Transportation.

[15] United States Code of Federal Regulations. *14 CFR 21.16 — Special conditions*, September 1980.

[16] Cristina Chaplain. Weapon system cybersecurity. Technical Report GAO-19-128, Government Accountability Office, Washington, DC, October 2018.

[17] Blaine Losier, Ron Smith, and Vincent Roberge. Design of a time-based intrusion detection algorithm for the MIL-STD-1553. Technical report, Royal Military College of Canada, Kingston, ON, January 2019.

[18] Sebastien J.J. Genereux, Alvin K.H. Lai, Craig O. Fowles, Vincent R. Roberge, Guillaume P.M. Vigeant, and Jeremy R. Paquet. MAIDENS: MIL-STD-1553 anomaly-based intrusion detection system using time-based histogram comparison. *IEEE Transactions on Aerospace and Electronic Systems*, Early Access, April 2019.

[19] Thuy D. Nguyen. Towards MIL-STD-1553B covert channel analysis. Technical report, Naval Postgraduate School, Monteray, CA, January 2015.

[20] Nagaraja Thanthry and Ravi Pendse. Aviation data networks: security issues and network architecture. In *38th Annual 2004 International Carnahan Conference on Security Technology*, pages 77–81, October 2004.

[21] Andy Greenberg. Hackers remotely kill a Jeep on the highway — with me in it. *Wired Magazine*, July 2015.

[22] Andy Greenberg. The Jeep hackers are back to prove car hacking can get much worse. *Wired Magazine*, August 2016.

[23] Julien Savoie. Software Defined Murder. Halifax, NS, April 2017. Presented at Atlantic Security Conference.

[24] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Probing the limits of anomaly detectors for automobiles with a cyberattack framework. *IEEE Intelligent Systems*, 33(2):54–62, March 2018.

[25] Charles Bernard and Sylvain P. Leblanc. Application of network security monitoring to the MIL-STD-1553 data bus. Technical Report ECE-2017-02, Royal Military College of Canada Computer Security Laboratory, Kingston, ON, November 2017.

[26] Rapid7. Understanding and Configuring Snort Rules. Technical report, December 2016.

[27] Richard Wade. Avionics appliances for ethernet networks: Real-time MIL-STD-1553B and ARINC 429 appliances bridge to ethernet networks. Technical report, Alta Data Technologies, May 2011.

[28] Alta Data Technologies, Rancho Rio, NM. *AltaAPI Software User's Manual*, November 2014.

[29] Alta Data Technologies, Rancho Rio, NM. *AltaCore-1553 MIL-STD-1553 Protocol Engine Specifications/Users Manual*, September 2014.

[30] Suzanne J. Matthews and David R. Raymond. Packet sniffing in python (cs1). Technical report, United States Military Academy, 2015.

[31] GE Intelligent Platforms. R15-USB RoHS Dual port MIL-STD-1553 USB Adapter. Technical report, GE Intelligent Platforms, Charlottesville, VA, 2012.

[32] Alta Data Technologies, Rancho Rio, NM. *AltaView Software User's Manual*, November 2014.

Appendices

A Detailed Structural Design

This appendix provides a detailed breakdown of the code behind Otto, intended as an accompaniment to Section 4.3, which provides a high-level overview of the program's structural design.

As discussed in Section 4.1, Otto is implemented in Python. This appendix will first discuss the imported Python modules used, the selection and use of global variables, the variable classes implemented. Armed with this knowledge, this appendix will then describe the implementation of each function.

A.1 Modules

Otto takes advantage of a number of Python modules.

`scapy` `scapy` is used to ingest APMP-formatted pcap files

`csv` The `csv` module is used to process the comma-separated value files used to define signatures.

`os.path` This module enables file imports, namely pcap and CSV files.

`binascii` During the packet processing routine, the data extracted from each of the payload fields is stored in decimal format by default. `binascii` functions are used to convert these into an ASCII string of 0s and 1s to represent the bitfield.

`logging` Functions from `logging` are used in conjunction with `scapy` in order to log any errors in packet manipulation.

`time.sleep` Used to provide a brief pause between the completion of the user configuration and result reporting. While not strictly required, the pause does make the user experience more pleasant by not bombarding the user with information.

`collections` The results of the RT frequency analysis are presented as part of the final reporting stage. This is done by way of a histogram, which is powered by the `collections` module.

`sys` `sys` is used to allow the software to accept arguments from the command line, namely the packet capture file to be analyzed.

`math` Used to provide various mathematical functions.

A.2 Global Variables

Otto uses three global list variables to store the signatures provided in the user-defined CSV files and make them available to all functions. These are called `cwsiglist`, `swsiglist`, and `dwsiglist`, for command, status and data words respectively.

Three more global variables are also used to store parameters for both command and status word repetition analysis. These are set by the user in the main function and taken up in the relevant repetition analysis functions. Using command words as an example, the `cwrepeat` variable is a boolean value that stores whether or not the user has requested word repetition analysis. If so, `cwsetsize` is used to define the number of previous words used to form the analysis set and `cwthreshold` is the number of identical words within that set that would trigger a detection.

Finally, two global variables are used to store the binary representation of the RT addresses of every command and status word observed on the bus: `cwhistogram` and `swhistogram`. As their names imply, these are used to collect data to generate the RT frequency analysis histogram displayed to the user before the program terminates. This will be further discussed in the `rtFreqAnalysis` portion of Section A.4.

A.3 Classes

Three classes are defined in Otto, used to structure the various fields from the ingested signatures into single variables. These variables are `CWSig`, `SWSig`, and `DWSig`.

Listing A.1 shows how the `CWSig` class is structured. This is done similarly for command, status and data words respectively, with the appropriate field names for the type of word in question.

```
class CWSig:
instance = 1
def __init__(self,address,tr,subaddress,wordcount,bus):
self.address = address
self.tr = tr
self.subaddress = subaddress
self.wordcount = wordcount
self.bus = bus
self.id = CWSig.instance
CWSig.instance += 1
```

Listing A.1: CWSig Class

Each time a new variable is created using one of these classes, the "instance" parameter is used to assign a unique serial ID number, and is incremented to prepare a new number for the next variable.

There are also some occasions where the use of a variable class could have been useful. Classes were overlooked as an option in the early stages of this work, and this artifact carried on through the development cycle.

A.4 Functions

A.4.1 `main`

Parameters Packet capture file location passed as a command line argument
Returns None
User Inputs User settings, including signature file locations and word repetition thresholds and windows.
User Outputs Text prompting the user for inputs, errors for invalid inputs, command and status word histograms
Function Calls `loadcommandsig`, `loadstatussig`, `loaddatasig`, `parsePCAP`

Otto's `main` function is where the user interaction occurs. The user is prompted to provide paths to signature files and options to enable or disable command and status word repetition detectors. There is also a measure of input error detection that occurs, including validating integer inputs and verifying whether the specified signature file paths are valid.

The contents of the pcap file specified in the path is loaded into a list variable called `pcap`, with one packet per list item. This functionality is provided by the `scapy` module. Some simple packet filtering is done here, with only UDP packets added to the list.

The signature file paths are passed as arguments to the respective signature loading functions: `loadcommandsig`, `loadstatussig`, and `loaddatasig`.

After all required user input is gathered, the `parsePCAP` function is called, with the `pcap` variable as argument.

This function is also responsible for some of the final reporting, namely the generation of the RT frequency analysis histogram. This is done using the `collections.counter` function. A detailed explanation of the output is given in the `rtFreqAnalysis` function description, as the output is directly related to that function's actions.

A.4.2 `parsePCAP`

Parameters `pkts` variable containing the pre-filtered packets from the `main` function
Returns None
User Inputs None
User Outputs None
Function Calls `cwsiglogic`, `swsiglogic`, `dwsiglogic`, `cwrepeat`, `swrepeat`, `rtFreqAnalysis`

The `parsePCAP` function serves to extract the useful data from the APMP packets contained in the user-specified pcap file.

The crux of this function is a `for` loop, which iterates through each packet in the file. For each packet, two command words, two status words and 32 data words are extracted. This is accomplished by slicing the string containing packet data at the appropriate bit offsets for each word. As the words are stored as a string of 0s and 1s, some additional manipulation is done to convert hexadecimal values into binary and to ensure any leading 0s are not dropped.

A packet ID is then defined, incrementing one each time through the loop to create sequential packet numbers. This is ultimately used to identify which packet causes a detection.

Once a packet is fully broken down, the relevant words are passed to the signature analysis functions (`cwsiglogic`, `swsiglogic`, `dwsiglogic`). If the user has requested command word or status word repetition analysis, those functions are invoked next. Finally, the `rtFreqAnalysis` function is called to ensure the command and status words are counted in the RT frequency analysis.

With the analysis of the first packet complete, Otto then repeats the cycle with the next packet, until the end of the capture file.

A.4.3 `loadcommandsig`, `loadstatussig`, `loaddatasig`

Parameters Path to the applicable signature file
Returns None
User Inputs None
User Outputs Prints the ID number and content of each line in the signature file in a binary format, displays error messages if the signature file contains improperly formatted parameters.
Function Calls `open`

The signature loading functions are similar for command, status and data words and can be discussed as a group. Command word signature detection will be the example discussed.

These functions are used to parse the comma-separated value signature files, check that the defined parameters are correctly formatted, convert the human-readable parameters into binary strings for further processing, assign signature ID numbers and load the signature strings into the appropriate signature list variable, as discussed in Section A.2.

Using Python's regular file manipulation functions, the signature file path passed as a parameter is used to open the CSV file and extract the contents. Then, row by row, the human-readable parameters are converted into their binary string equivalents. Prior to each conversion, basic input validation is done to ensure the value given is correctly formatted, e.g. the T/R bit field should only contain a T or an R. Any input validation test failures are reported as errors and the program exits.

Each parameter is loaded into a temporary CWSig, SWSig or DWSig-class variable, along with a sequentially generated signature ID number. The contents of that temporary variable are then appended to the global cwsiglist, swsiglist or dwsiglist variable for later use.

A.4.4 `cwsiglogic`, `swsiglogic`, `dwsiglogic`

Parameters Relevant words extracted from the current packet, packet ID number
Returns None
User Inputs None
User Outputs None
Function Calls `detection`

As with the previous functions, the signature logic functions can be discussed as one.

The function is centered around a `for` loop, iterating for each command word signature held in the global variable list.

Nested inside is an `if` statement, which skips the logic if the word under analysis is a null-value (defined by the APMP as all `1`s).

If this test is passed, the comparison of the command word to the signature begins, testing each word parameter-by-parameter. If a match is detected in the RT address field for example, all other word parameters are checked to see if they match the signature, or the "don't care" value (all `x`s). This is done to avoid a detection if multiple matching parameters are required for a detection.

If a detection is made, the `detection` function is called to handle the reporting. It is passed the type of detection made ("command", "status", or “data”), along with the value of each word parameter, the signature that triggered the detection, and the packet ID number.

This check is done for both command word 1 and command word 2, and the loop repeats with the next signature on the list. Once all signatures are exhausted, the function returns to the calling `parsePCAP` function, which parses the next packet.

It should be noted that in the case of data words, signature checking is done at the word level since there are no standard parameters for this word type. Therefore, a check of the full word against the full signature is done. Since none of the scenarios presented are expected to require data word signature detection, this basic functionality is sufficient for this proof of concept.

A.4.5 `cwrepeatdetector`, `swrepeatdetector`

Parameters Command and status words, packet ID
Returns None
User Inputs None
User Outputs None
Function Calls `detection`

If requested by the user in the configuration phase, the word repetition detection functions are called. The command word repetition analysis function will be discussed as the example.

Firstly, the command word parameters are concatenated to form a 17 character string (16 bits plus one bit to indicate the bus). This string is tested for the null string discussed previously to ensure the null string is not saved to the set as this would result in a high number of false alarms.

The `.count` attribute of the cwrepeatlist list variable is then used to count the number of occurrences of the string in the set. If this count is equal to or greater than the user-defined threshold, the `detection` function is called.

Once the test is complete, the string is then added to the set by simply appending it to the cwrepeatlist variable.

This is done a second time using the string from command word 2.

Finally, the length of the set is checked using a while loop. While the length of the set exceeds the set size defined by the user, the oldest list element is dropped. This ensures that only the most recent values are retained, up to the maximum number defined by the user.

A.4.6 `detection`

Parameters The content of the word causing the detection and the bus on which it was observed, the type of detection (command, status or data word signature match, or command or status word repetition threshold exceeded), the packet ID number, and the signature matched for signature-based detections.
Returns None
User Inputs None
User Outputs Text containing details of the detection.
Function Calls None

The `detection` function is perhaps the most versatile one in the program, handling the generation of alerts for both the signature detection or word repetition analysis functions.

A single detection function ensures a consistent output for all detections. It also provides the analyst with a single location to modify the detection handling, including changing the outputs displayed to the user, implementing a logging feature, or passing specific alerts to other functions or programs.

An `if` statement is first used to check the detection type passed. Valid options are "command", "status" and "data" for signature detections, and "commandrepeat" and "statusrepeat" for detections made by word repetition analysis.

Command, status and data signature detections are handled nearly identically, by displaying the packet ID number where the detection was made, which word within that packet raised the detection, the ID number of the signature matched, and the parameters of that signature, e.g. T/R bit set to transmit, RT# 17 and subaddress 13. For data words, the parameters are replaced by the full word.

Signature detection also includes a check to verify that the parameters in the word do indeed match those in the signature. If no match is observed, an error message is displayed. Initially written as a debugging feature, this check was kept as part of the final code as a backstop against erroneous detection or incorrect variables being passed to the detection function. However, this error has never been observed during the development or experimentation phases of this work.

A.4.7 `rtFreqAnalysis`

Parameters cw1, cw2, sw1, sw2
Returns None
User Inputs None
User Outputs None, though the information compiled by this function in global variables is presented to the user by the `main` function.
Function Calls None

The `rtFreqAnalysis` function is used to keep track of the command and status words observed. It is called by the `parsePCAP` function every time an APMP packet is parsed.

The function is passed the contents of both command and status words in the packet as binary strings. It performs a first-order check to ensure the value is not the null value, i.e. all `1`s: if the check passes, the content is appended to either `cwhistogram` or `swhistogram`. After the last packet is parsed, these two variables will contain a list of all command and status words observed on the bus.

These global variables are used by the `main` function immediately before the program terminates. Histograms are constructed for command and status words observed, using the RT address as the key. The results are reported as lists of the RTs ordered by frequency, along with the number of total words containing that RT address.