Practical Security Solutions for Serial Data Bus Networks



Matthew Rogers

New College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2022

Acknowledgements

Institutional

To the Rhodes Trust for funding the entirety of the DPhil along with a stipend for my living expenses.

To the CDT in Cyber Security for the interdisciplinary viewpoint into cyber security.

Personal

To the Twimpers for keeping joy and creativity alive.

To Kristiana Yao, Jonah Stewart, Nicolette D'Angelo, Jenny Huang, and Kate Reed for being lifelines throughout the Covid pandemic.

To Phillip Weigand, Olivia Puelo, Jeramy Lochner, Lee Vanrell, Austin Hancock, James Correnti, Ellie Daw, Hillary Hester and Dwight DeGroff for your remarkable talent and kindness which kept me motivated through this DPhil.

To Inda Imanda and Klaudia Krawiecka for keeping positivity up in the office.

To Claudine Tinsman, Sebastian Kohler, Anjuli Shere, and Yash Sharma for your friendship throughout the CDT.

To Jassim Happa for going above and beyond with providing feedback even well after he stopped being my on-paper supervisor.

Abstract

For decades, manufacturers incorporated computers into vehicles, weapon systems, and numerous pieces of modern infrastructure, networking them together via serial data bus networks. These networks took direct control away from the operator in the name of efficiency and safety, but we now know this bargain came at the cost of security. By having no authentication in the design, anyone communicating on the network can create harmful effects. The natural solution to this problem is to redesign these networks to support authentication, but abandoning decades of vital infrastructure would be unfathomably expensive. The only option is to continue using our existing equipment, but to develop a way to add security in an efficient and cost effective manner. However, there are an innumerable number of ways to design and implement a serial data bus network. This thesis lays out a methodology for taking a serial data bus network, understanding it, and designing a security patch for it. To illustrate the utility of our methodology, we implement two proof-of-concept intrusion detection systems. The first improves upon existing CAN detection techniques by using a standardized data field to abstract the state of the system. The second takes advantage of the architecture of MIL-STD-1553 to more efficiently detect new messages on the network. Our protocol-specific approach is emphasized by our development of new attacks on the CAN protocol. Finally, this thesis produces an evaluation framework for the prevention and detection systems increasingly being deployed to these vehicle and weapon system networks. Our framework generates a score for defenders, such that they know exactly what attacks and attack vectors their system can detect and prevent. By looking at multiple protocols, from ground vehicles to fighter aircraft, we highlight how a protocol targeted security approach creates more accurate, more efficient, and more practical technology, to detect even the most advanced attackers.

Contents

1 Introduction **1**

1.1 Existing Problems 4
1.2 Proposal 5
1.3 Ethical Considerations 9
1.4 Contributions 9
1.5 Structure of the Thesis 10

2 Background **12**

2.1 CAN Background 12
2.2 J1939 Background 14
2.3 MIL-STD-1553 Background 16
2.4 ARINC-429 and ASCB Background 19
2.5 Attacking Serial Data Bus Networks 20
2.6 Threat Models 21

3 Abusing the CAN Error Process with Bit-Flip Attacks **22**

3.1 The CAN Error Process 24
3.2 System and Adversary Model 27
3.3 Stealthy Bus-Off Attack 28
3.4 A Probabilistic Attacker 30
3.5 Blind Sync Methodology 33
3.6 Detecting Bus-Off Attacks Without Bit-Flips 35
3.7 Bit-Flip Attack Detection 38
3.8 Stealthy Attacker Security Analysis 42
3.9 Discussion 43
3.10 Conclusion 44

4 A Methodology for Designing Protocol-Specific Detection Systems 45
4.1 Protocol-Centric IDS Methodology 45
4.2 Generic IDS Development 46
4.3 Using the Protocol for Detection 50
4.4 Enforcing the Protocol 54
4.5 Threat Models 56
4.6 Building the IDS 58
4.7 Conclusion 65

5 Detecting CAN Attacks on J1939 and NMEA 2000 Networks 67
5.1 System and Adversary Model 69
5.2 CANDID's Design 71
5.3 Manipulation Attacks 81
5.4 Security Analysis 85
5.5 Implementation and Experimental Results 90
5.6 Discussion 96
5.7 Conclusion 99

6 Efficient Attack Detection for MIL-STD-1553 Networks 100
6.1 System and Adversary Model 102
6.2 Intrusion Detection for MIL-STD-1553 104
6.3 Security Analysis 110
6.4 Experimental Results 111
6.5 Preventing Illegitimate Command Words 114
6.6 Discussion 115
6.7 Conclusion 118

7 An Evaluation Framework for Intrusion Prevention Systems on Serial Data Bus Networks 119
7.1 Existing Research 121
7.2 Adversary Model 125
7.3 Types of Prevention Systems 125
7.4 Attack Types 128
7.5 Detection for the Purpose of Prevention 131
7.6 Evaluation Framework 133
7.7 Attack Testing 141
7.8 Case Studies 143

7.9 Discussion 151
7.10 Conclusion 154

8 Related Work **155**
8.1 Attacking Serial Data Bus Networks 155
8.2 Defending Serial Data Bus Networks 158

9 Conclusion **165**
9.1 Overall Conclusions 165
9.2 Chapter Contributions 167
9.3 Final Thoughts 170

Bibliography **172**

List of Figures

2.1 J1939 Packet Structure 14
2.2 J1939 SPN Example 15
2.3 MIL-STD-1553 Words 17
2.4 MIL-STD-1553 Traffic 19

3.1 Extended CAN Packet with Errors Highlighted 24
3.2 Bit-Flip Attack 28
3.3 Probabilistic Attacker for the Stealthy Bus-Off Attack 31
3.4 Blind Can Sync 34
3.5 Source Identification Tree for CAN 38
3.6 Cascading CAN Errors Against EREFs 40

4.1 IDS Deployment Options 48
4.2 Detection for Prevention Flowchart 59

5.1 CAN System Model 70
5.2 CAN Interval Data 70
5.3 CANDID Training Phase 73
5.4 State-Based Interval Design 74
5.5 CANDID Evaluation Phase 80
5.6 Manipulation Attack 83
5.7 Manipulation Attack Detection 84
5.8 CAN Testbench 89
5.9 Masquerade Attack Evaluation 94

6.1 MIL-STD-1553 System and Adversary Model 103
6.2 MIL-STD-1553 Transformer Coupling Stub Connection Plus IDS . . . 107
6.3 Bus Controller Fingerprinting Experimental Results 113

7.1 Options When Interacting With a Serial Data Bus 130
7.2 Evaluation Framework for IPS Validity 134

7.3 Evaluation Framework for IDS Efficacy and Suitability 135

List of Algorithms

1 Error Resistant Error Frame Error Counter Logic 39
2 CANDID Data-Dependency Training 78
3 MIL-STD-1553 Alerting 109
4 IPS Suitability Score Calculation 140
5 IDS Efficacy Score Calculation 141

List of Abbreviations

ACK	Acknowledgement
ARINC	Aeronautical Radio INC
ASCB	Avionics Standard Communications Bus
BC	Bus Controller
CAN	Controller Area Network
CANDID	CAN Data Intrusion Detection
CRC	Cyclic Redundancy Check
CUSUM	Cumulative Sum
DBC	CAN Database
DDC	Data Device Corporation
DoD	Department of Defense
ECM	Engine Control Module
ECU	Electronic Control Unit
EDL	Electronic Data Logger
EREF	Error Resistant Error Frame
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
LIPS	LRU Intrusion Prevention System
LRU	Line Replaceable Unit
MIL-STD-1553	Military Standard 1553
MIPS	Message Intrusion Prevention System
NMEA	National Marine Electronics Association
OBD	On-Board Diagnostic

PGN	Parameter Group Number
RT	Remote Terminal
SAE	Society of Automotive Engineers
SPN	Suspect Parameter Number

Chapter 1

Introduction

For the past several decades vehicles have become controlled not by physical mechanisms, but by computers, communicating on a shared serial data bus network. Manufacturers face increasing efficiency incentives to connect these networks to the external world. These networks and their external connections are crucial for much of the functionality and efficiency we see in modern vehicles, yet they have a fundamental problem: vehicle serial data bus networks have no mechanism for authentication or for validating that a message is really from who it says it is from. Someone with access to a serial data bus network can transmit a message as anyone and control the vehicle to possibly catastrophic outcomes. But this is not a new problem. Popular journalism around consumer car hacking was prevalent in the 2010s, and these networks were introduced as early as the 1970s. Five decades of vehicles use unauthenticated networks, and the world continues to use the same unauthenticated protocols invented in the 1970s and 1980s. This thesis looks to these legacy systems across numerous domains and seeks to add cyber security to them in a cost effective and efficient manner. In particular we focus on industrial and military systems, from trains and trucks which use the Controller Area Network (CAN) protocol to fighter jets which use the MIL-STD-1553 protocol.

It is worth noting why we believe now is the time to work on industrial and military serial data bus protocols. After all, these protocols have existed for nearly five decades and a decade of car hacking research has largely avoided these groups. We are currently in a unique time frame, where the US government is focusing on critical infrastructure cyber security, particularly avionic system security [64]. In 2018 the US Department of Defense (DoD) published the MIL-STD-1553C standard [61] which encourages cyber security by design, and the Government Accountability Office wrote a scathing report of weapon system security in the DoD [81]. In 2020, 2021, and 2022 the US Air Force allowed researchers access to satellite equipment in Hack-a-Sat [71]. Additionally, the

US Navy allowed hackers to access naval test beds in HackTheMachine [50], with a focus on intrusion detection for a naval variant of the CAN protocol. In the industrial space, legislation which forced rail companies to implement Positive Train Control (PTC) by the end of 2020 [62] opened these companies up to further technology discussions, including cyber security. These developments make it clear security has not been taken seriously in the past and that security could be implemented in the future. The problems of serial data bus cyber security worsen with the increasing integration between systems, be it tanks [79], new generations of fighter aircraft [82], or commercial rail [28]. This integration means that any security system we propose needs to account for remote attacks; at the same time, the critical nature of these systems means that a security system must account for highly motivated attackers implanting new devices onto the bus and targeting vehicle supply chains. By expanding beyond consumer-level car hacking, we are producing highly impactful cyber security research just as these industrial and military organizations are realizing the looming threat facing their decades of assets.

Our starting question for adding cyber security to these networks is what can feasibly be adopted at scale. The classical solution to authentication is adding a cryptographic security mechanism. However, this requires modifying existing computers to add a key exchange mechanism to them. While modern versions of this solution are proving to be effective and efficient, cryptography falls short for a few reasons. Upgrading each computer on the network is expensive at best and implausible at worst. In many cases vehicle operators will not own the source code of the tens of computers on their system, meaning they would have to request an upgrade from every one of their manufacturers. The drive and incentives for adding vehicle security are simply not there. More importantly, even with working authentication, the impetus for this work is the increased risk of remote attacks from the increasing connectivity between serial data bus networks and the outside world. Authentication is circumvented by any attacker which can take over a victim and gain access to any secret keys. These problems mean that cryptographic authentication is both impractical and ineffective against the attack vector we are most concerned about. We further examine the problems of cryptographic authentication for serial data bus networks in Chapter 8. To practically detect our remote attacker, as well as any attacker who corrupts a device, we focus on patching legacy systems with Intrusion Detection Systems (IDSes).

The idea behind a patch system to secure legacy networks is simple. Using a fairly cheap computer, the defender connects to the network and attempts to detect anomalies in network traffic. Adding a small device to the network is generally

unobtrusive, and by having a cheap computer a maintainer could afford to install this IDS across a fleet of vehicles. A decade of existing work proposes intrusion detection systems, yet they are almost all focused on consumer CAN systems. This begs the question: how does that detection system change for other protocols?

This thesis touches on four industrial and military protocols: J1939, MIL-STD-1553, ARINC-429, and ASCB, with a primary focus on J1939 and MIL-STD-1553. We choose these protocols as they are common amongst vehicles and they represent a diverse set of architectures and transmission patterns. This diversity is important for elevating this work from a series of useful but limited intrusion detection systems to a methodology for adding security to new and old serial data buses. Each architecture we include is another example for where the attacker can be designed out of the system and where we can provide better tools for the defender. From J1939, an industrial standard built on top of CAN, we observe the benefits of having an accessible standard for the data field. The standardized data field enables easier integration and makes it easier for a third party to develop a comprehensive security solution, but it comes at the cost of attackers knowing what messages do what without the extensive reverse engineering of a proprietary system. MIL-STD-1553 is designed for fighter jets and demonstrates the benefit of a bus controller architecture, where one computer controls all of the traffic. The bus controller provides a dangerous target for an attacker to impersonate, but it allows for simpler detection mechanisms for the majority of attacks. ARINC-429 is designed for commercial aircraft and uses a point to point architecture where each transmitter is wired directly to its receivers with one way connections. The point to point architecture makes patching the system difficult but achieves greater security through the segmentation and isolation of systems. ASCB is a proprietary aircraft protocol from Honeywell with a fixed-frame transmission structure. This makes detecting an attacker imitating a legitimate device trivial for defenders but comes at the cost of making it incredibly difficult to modify the system.

We elaborate more on each of these protocols throughout this thesis, with full chapters dedicated to the J1939 and MIL-STD-1553 protocols. Of course, a decade of CAN intrusion detection research guides how we design our detection systems. The fundamental approach for how to solve the anomaly detection problem is always relevant. However, we find several problems with translating existing approaches to this new context.

1.1 Existing Problems

The first and foremost problem with existing work is accuracy. Vehicles exist in a context where attacks and problems are rare, and traffic throughput is so high that any level of false positives can quickly become problematic. For example, from a 0.055% false positive rate on an attacker-free CAN bus, we would expect approximately 6 false positives a minute, because most of the traffic we have seen indicates that approximately one million messages an hour is normal. This level of false positives could be acceptable in a modern TCP/IP network, but in the context of a vehicle, alerts are a dangerous distraction if they do not provide some context to the operator. At the very least, these alerts need to provide valuable intelligence for future operational issues. The solution to this problem is a more accurate IDS. For accuracy, we focus on the protocols we have outlined above. By having a better understanding of how the protocol works, particularly how it processes errors and faults, we can often develop more accurate detection mechanisms, even with limited computational capacity. The primary contribution of Chapter 5, CANDID, does this by using the standardized data field to more accurately capture the state of a vehicle.

The second problem is covering all possible attack messages. From prior work it is clear that many attacks only require a single message. This means that any detection system needs to be capable of registering an attack on every single message. Otherwise, an attacker with knowledge of the IDS will simply use the subset of messages the security analysis does not cover. There are two cases for this gap in coverage: model-based gaps and protocol-specific attacks.

Model-based gaps are situations where an IDS accounts for the majority of messages, but finds some messages have characteristics which are difficult for a security system to analyze. Thus, to avoid false positives, those messages are ignored. In CAN systems we often see this from the perspective of timing. In brief, most messages on the CAN bus are transmitted periodically, but a few are transmitted with changing periodicity, or on the basis of some change in state. These gaps, which are prominently featured to attackers in academic papers [44], [12], [86], are simple to exploit. Model-based gaps must be addressed, otherwise the security of the system is reliant on the obscurity of the system.

The protocol-specific attack issue is best demonstrated with a denial of service attack in J1939 [55]. The protocol adds a networking message to CAN to better define what devices do and to allow for more targeted messages. The consequence is that an attacker can send the protocol's address claim message to a legitimate device, win the

address competition every time, and then that legitimate device will stop transmitting. There are a number of ways to go about detecting this attack, but a generic approach will often fail to account for it. The address claim message is common enough that alerting on every occurrence would result in false positives, but not so common that an algorithm can easily predict when the message should occur. A more targeted approach is often better, but it requires an understanding of the protocol.

This segues into our third problem of accounting for the different threat models that spring from different protocols. While it can be tempting to seek a silver bullet solution which is applicable to every protocol, to every vehicle, regardless of how the attacker gets onto the system, the reality of what is possible is different. Let us use the CAN bus as an example for how the protocol can completely change what is possible on a system. In Chapter 3 we outline a technique to synchronize the bus regardless of the attacker's ability to read any of the data going over the bus. This enables a remote attacker to transmit full messages without concerns for transmitting while the bus is active, where another protocol like MIL-STD-1553 has no synchronization mechanism. Each remote transmission against a MIL-STD-1553 bus is blind and could override an existing message, or transmit a new message. This distinction affects how realistic different attack vectors are, which drastically changes the needs of the security model. While creating a security system which accounts for all possible attacks is useful for the sake of academic argument, it risks an over-engineered system which is worse at detecting the most realistic attacks. The same is true for discussions of protocol architecture. A bus controller system like MIL-STD-1553 or ASCB, where one device initiates every message, has different timeframes for when an attacker can inject a message versus an arbitration based protocol like CAN. This becomes important for considering how much work the IDS has to do. If an attacker can always send a message, then the IDS must consider every message an attack. A more structured transmission pattern allows a defender to restrict their IDS to specific time windows, rather than processing every message.

By addressing the problems outlined above, we can answer the growing calls for security in industrial and military systems. This work provides a map for analyzing and securing old and new serial data bus protocols.

1.2 Proposal

This thesis seeks to answer the primary question of how to design an intrusion detection system for a serial data bus network in a practical and efficient manner.

Practical means integratable for legacy systems, accurate, adaptable to new attacks, and actionable. Efficient means making the most of the information the bus provides and being as close to real time as possible where it matters. Each chapter of this thesis breaks down this question into a series of secondary questions.

First we explore how the protocol affects what actions an attacker can perform. Existing work shows that protocol-specific attacks exist and that some abuse of the protocol can lead to new attacks [55]. But there is a difference between a very specific attack for which it is trivial to design static rules for, and attack techniques which enable a breadth of attacks by disguising them or making them more reliable. For CAN we look to the bit-flip attack, where an attacker modifies a message as it is transmitted on the line. The attack itself flips bits without changing the timing properties of the victim message, making it a good technique for avoiding many detection systems. The error process built into the CAN protocol increases the reliability of the bit-flip attack. Simultaneously CAN errors enable entire swathes of limited blind attackers, from remote attackers incapable of queuing their messages, to electromagnetic induction attackers remotely flipping bits on the wire without any feedback. As we see how an attacker can abuse the protocol for offensive gains, the next question is asking if a defender can do the same for security gains.

From existing work we have seen protocol abusing attack papers, but when we look to defensive papers we find that techniques are often quite similar from protocol to protocol [41], [88]. This leads to bespoke solutions for protocol-specific attack papers rather than a comprehensive solution. More broadly, it results in overcomplicated solutions which do not necessarily factor in what an attacker can actually do on a given system. To this end, this thesis asks what information, particularly protocol-specific information, is pertinent for building a serial data bus intrusion detection system. We analyze the common design patterns across architecture, message traffic, error conditions, and hardware. This exploratory analysis informs a methodology for designing an IDS to suit the protocol and for discovering protocol-specific attacks. We apply this methodology to two systems: a J1939 bus and a MIL-STD-1553 bus. Each protocol asks and answers another secondary question.

For J1939 we ask what happens if we incorporate vehicle specific data into an IDS and how that can be integrated into a full CAN system. J1939 is almost the same as CAN with the exception of some basic networking and the standardized data field. Answering this question determines if it is worthwhile to translate proprietary CAN data fields and allows us to improve the accuracy and coverage problems of existing work. Our J1939 solution, CANDID, highlights what can be done by incorporating

the standardized data field of J1939 into a timing-interval-based detection system for CAN. This data analysis enables us to capture the state of the vehicle, detect messages with irregular timing intervals, and take advantage of the dependencies between different vehicle computers to restrict even the most advanced attacker. Our advances in spoofing detection and data analysis mean that an attacker must take over or disable a victim device to transmit that victim's messages, and even in that scenario the attacker's transmitted data cannot deviate from the rest of the system unless the victim is independently in control of that data. Previously the standardized data field of J1939 was seen as a boon for attackers [8], but CANDID shows it can be even more of a boon for defenders. On top of our data and timing analysis CANDID includes a protocol-specific voltage detection system, designed to elegantly and simply detect the bit flips which enabled so many attacks in our analysis of CAN. This removes the need for a generic voltage fingerprinting solution which requires the overhead of a complicated training system and which risks producing false positives.

For MIL-STD-1553 we ask what impact the architecture has on an IDS. MIL-STD-1553 is completely different from CAN systems, yet existing work treats it largely the same [88], [29]. Accounting for the architecture reveals whether a generic solution is best and what benefits a defender can derive from the architecture. Our MIL-STD-1553 solution takes the bus controller architecture and places trust in that controller. This reveals how having a centralized device that initiates all messages can simplify detection systems while making them more accurate against most attacks. Our MIL-STD-1553 IDS also provides insight into what level of accuracy an intrusion detection system requires. A single message in MIL-STD-1553 can result in disabling every avionics computer; thus, a detection system must detect every individual attack message. We achieve this level of detection by fingerprinting only the bus controller. MIL-STD-1553 also needs its own solution to bit flip attackers. In this case we use the error flags built into MIL-STD-1553 to detect bit-flip attacks. Our MIL-STD-1553 IDS replaces a voltage fingerprinting system and achieves better detection coverage than previous work with error flags and simpler, protocol-specific fingerprinting. Additionally, our MIL-STD-1553 chapter considers the ramifications of an intrusion prevention system and what detection systems are suited to powering prevention systems.

In aggregate these questions, when answered, form a series of more targeted intrusion detection systems which provide a defender more confidence in their ability to detect attacks. This confidence comes from an understanding of how messages are transmitted across the bus, and a deeper understanding of what features of the protocol are prone to vulnerabilities. Once a defender builds this more comprehensive

and efficient detection system, the obvious question is what should be done on an alert. We believe that prevention systems are a natural next step as detection systems improve.

Intrusion prevention is a common next step for intrusion detection research. If we can detect attacks, then surely we can, and should, prevent them. This is an increasingly popular, or at least more frequently suggested, concept in the field of vehicle and weapon system security [81], [94], [53]. However, there are a limited number of scenarios where IPSes are applicable, this is for two reasons. IPS technology is predicated on the idea that its prevention mechanism can stop an attack, but an attacker can circumvent the prevention mechanism itself. On serial data bus networks the prevention mechanism often needs to be communicated on the network, this means that IPSes often assume that an attacker has to read the bus, or follow the standard, or even accept firmware updates. These attacker assumptions inherently limit the applicability of the IPS. On top of this nuance with when IPSes work, we find that detection systems need certain features to be safe prevention systems. IDSes can be perfectly good detection systems while making for dangerous and foolhardy prevention systems. This is because an IPS cancelling a message on a flying aircraft could result in disastrous consequences. Whether an IDS is good enough is a subjective measure, dependent on how the defender perceives their threat model and their own risk tolerances. But there are certain criteria we can use to know how well an IDS performs and how suited it is to becoming an IPS. To this end, we develop an evaluation framework which scores an IDS and IPS solution based on three axes; how viable an IPS is against the defenders threat model; how suitable an IDS is to powering an IPS; and how effective the IDS is at detecting attacks. Our evaluation framework quantifies how practical and how efficient an IDS needs to be and allows a defender to knowledgeably select an IDS/IPS solution knowing full well what it can and cannot defend against. Simultaneously it allows researchers to design stronger detection systems while providing much needed clarity for what kind of attacks their IDS can actually detect, how resilient their detection strategy is, and which aspects of the detection system are compatible with becoming a prevention system.

All of our proposed research is based on either processing serial data bus data in real time by connecting to the vehicle network, or having it log this data and replaying it via software with the timestamps attached during collection. By testing our implementations on real systems we prove out the feasibility of them being used in the field. This is true not only from an integration perspective, but also in terms of our highlighted existing problems. The consequences of high false positives are easily

visible when a known good system is alerting while operating normally. If the IDS does not cover all known messages, clever attackers avoid detection. To help avoid this, some of our implementations were tested by third party red teams independently developing attacks and executing them with our system installed. These attack tests also demonstrate what vehicle maintainers believe is a realistic threat model and give us the threshold for our weakest possible attacker.

1.3 Ethical Considerations

All attacks done on real systems were done by the system owners with our device passively connected to their system with their explicit permission. The attacks themselves were developed entirely by the system owner with our device acting purely in an alerting capacity. The names of the organizations in charge of these systems and any datasets they shared are not included in this thesis.

1.4 Contributions

This thesis provides the following novel contributions to the field of serial data bus and cyber physical systems security.

- An improvement of the state of the art for attacking serial data bus protocols through the exploitation of error processes and the thorough the study of bit flip attacks. We demonstrate that protocol information simplifies complicated attacks and significantly lowers the bar for attacking serial data bus networks. (Chapter 3)

- Two novel intrusion detection systems which improve the state of the art for CAN and MIL-STD-1553 security. Our IDSes expand security coverage, improve accuracy, and simplify implementations. Our IDS development methodology enables the development of more protocol-centric detection systems. All of this results in more practical and efficient security systems which are safe to apply to existing systems. (Chapters 4 - 6)

- An evaluation framework for intrusion detection and prevention systems which provides future researchers the tools to develop more resilient detection systems capable of detecting advanced attackers, while safely preventing the attacks the IDS can detect swiftly and accurately. (Chapter 7)

1.5 Structure of the Thesis

This thesis is divided into a background section, five content chapters, related work, and a conclusion. The background provides information about our four protocols, focused on high level architecture and message formats, as well as a brief background on attacking serial data bus networks. Chapter 3 presents novel CAN attacks and methodologies which abuse the CAN error process, as well as a redesign of said process to disable CAN error attacks. Chapter 4 expands on each of our protocols from the perspective of a defender and attacker. In this chapter we propose a methodology for developing future intrusion detection systems for serial data bus protocols. The next two chapters are implementations of this methodology. Chapter 5 is our J1939 intrusion detection system, featuring the integration of timing and data analysis, in addition to targeted voltage analysis for a bit flipping attack we develop. Chapter 6 is our MIL-STD-1553 IDS, which avoids well trodden CAN IDS research to use the architecture of MIL-STD-1553 to our advantage, allowing for more targeted detection, better suited to the most dangerous attacks possible for the protocol. Chapter 7 proposes an evaluation framework which allows a defender to assess how well a given intrusion prevention system and intrusion detection system work. The framework assesses what attack vectors prevention is viable against, as well as how suitable the IDS is to powering the prevention system, and the efficacy of the IDS itself. Finally, we go over related work in Chapter 8 before concluding the thesis.

The chapters of this thesis are based on the following peer reviewed publications, submissions, and talks.

- Chapter 3: M. Rogers and K. Rasmussen 'Silently Disabling ECUs and Enabling Blind Attacks on the CAN Bus', ESCAR Europe 2022
- Chapter 4: M. Rogers, 'The Vulnerabilities and Security Benefits of Various Vehicular Network Protocols', Ground Vehicle Systems Engineering & Technology Symposium (GVSETS) 2021.
- Chapter 5: M. Rogers, P. Weigand, J. Happa, and K. Rasmussen, 'Detecting CAN attacks on J1939 and NMEA-2000 networks', Transactions on Dependable and Secure Computing
- Chapter 6: M. Rogers and K. Rasmussen, 'Targeted Intrusion Detection for the MIL-STD-1553 Protocol', Transactions on Aerospace and Electronics Systems (accepted with included revision)

- Chapter 7: M. Rogers and K. Rasmussen, 'An Evaluation Framework for Intrusion Prevention Systems on Serial Data Bus Networks', Asia CCS 2023. (accepted with included revision)

Chapter 2

Background

This chapter provides a high-level background on the protocols referenced in this thesis and how they function. Table 2.1 summarizes these protocols and what they are used for. We go into more detail on protocol information and attacks in the relevant content chapters, Chapters 3 and Chapter 5 for CAN, and Chapter 6 for MIL-STD-1553. Notably, the computers connected to serial data bus networks have multiple names depending on what protocol is being discussed. We use LRU (Line Replacable Unit) as a generic term for a computer on a vehicle, with the protocol specific terms being ECU (Electronic Control Unit) for CAN, and RT (Remote Terminal) for MIL-STD-1553. Additionally, we provide a high level overview of serial data bus network attacks.

2.1 CAN Background

CAN is a serial data bus protocol invented in 1986 by Bosch [7]. It is used in most modern ground vehicles. CAN was designed to reduce costs by increasing the efficiency of vehicle communication through a common bus. CAN uses the differential signal between two wires, CAN High (CAN-H) and CAN Low (CAN-L) to read either a 0 or 1 signal on the bus. A 0 signal is approximately 2 differential volts and a 1 signal is approximately 0 differential volts. A 0 signal is referred to as the dominant state. If

Protocol	Bitrate	Primary Customer	Primary Domain	Primary Differentiating Factor
CAN2.0	250-1000Kbits/s	Consumer	Ground	Arbitration Process
CAN-FD	1-8Mbits/s	Consumer	Ground	64-Byte Data Field
J1939	250Kbits/s	Industry/Military	Ground/Sea	Standardized Data Field
MIL-STD-1553	1Mbits/s	Military	Air	Bus Controller
ARINC-429	100Kbits/s	Industry	Air	Direct Connections
ASCB-C	2/3Mbits/s	Industry	Air	Structured Frames

Table 2.1: The key differences between the serial data bus protocols covered in this thesis.

any device is transmitting a 0 signal on the bus, the value of the bus is 0. A 1 signal is referred to the recessive state, and it is the passive state of the bus. Computers connected to the CAN bus are commonly referred to as Electronic Control Units (ECUs). The specific ECUs vary by vehicle, but some example systems controlled by ECUs include: transmission, engine, brakes, steering, and more. An ECU is usually connected to some physical component like an engine block which sends inputs and reads outputs to/from the physical component while reading data from the bus to see what the rest of the system is doing. To listen to these ECUs, any device with the appropriate connections only has to connect to the common bus.

ECUs send standardized CAN packets across the bus. A CAN packet is split into a few major components: the arbitration ID, control, data field, CRC, acknowledgement, and end of frame. The arbitration ID, so named because it is used to arbitrate which device gets to transmit, is either 11 or 29 bits. This ID is used to identify what device is sending the message and what kind of message it is. For arbitration, every ECU attempts to transmit either on a silent bus or after the 11 consecutive 1's at the end of the frame. The bus is synchronized by the bus transmitting an initial 0 to start a CAN packet. Once ECUs are transmitting, they monitor the bus line and compare it to their own transmission. If they are transmitting a 1 and they observe a 0 then they stop transmitting. This means the lower the value of the arbitration ID, the higher the priority of the message. Arbitration continues throughout the entire arbitration ID, with ECUs designed such that no two ECUs transmit the same ID. By the end of the arbitration process only one ECU is transmitting.

After arbitration the control field indicates the number of bytes of data. Data is maxed out at 8 bytes for CAN 2.0 systems. The data field is straightforward: it contains the actual vehicle information such as engine speed, gear, cruise control, brake pedal angle, etc... The CRC is used to check the integrity of the CAN packet. The acknowledgment is a 0 bit set by receiving ECUs and is used to test that receivers are properly synchronized and listening. Finally the end of frame is 11 consecutive recessive bits, after which ECUs are free to begin transmitting again. In terms of speed, the CAN 2.0 bus is relatively slow. It has a maximum bitrate of 1MBits/s, though slower bitrates are common. The CAN-FD bus is newer (2011) and has an 8 MBit/s bitrate as well as a 64 byte data field.

An important note for the CAN 2.0 data field is that it is proprietary. An automobile manufactured by Ford uses a completely different ID mapping than one made by Honda, and the data fields may change meaning for each make and model [18]. There is a standardized way of translating CAN data, called a DBC (CAN Database)

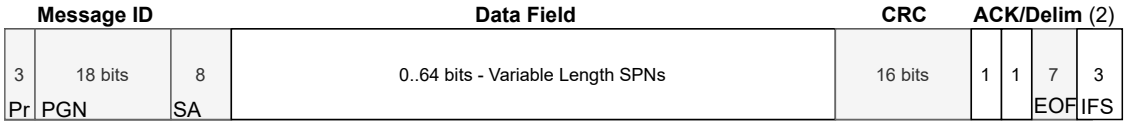


Figure 2.1: A J1939/NMEA 2000 Extended CAN Packet. Pr is priority, PGN indicates what kind of data is being sent, and SA is the source address. ACK indicates Acknowledgement (dominant then recessive), EOF indicates End of Frame (recessive), and IFS is the interframe space (recessive).

file. These are often difficult for researchers to gain access to, but are produced by vehicle manufacturers. Without access to a DBC file it is necessary to reverse engineer the vehicle, either by going through firmware, or rapidly injecting random messages while the vehicle is on stilts and documenting the results.

2.2 J1939 Background

This thesis focuses primarily on J1939, a software standard created by the Society of Automotive Engineers (SAE) which tells ECUs how to interpret the contents of an extended CAN 2.0 packet [84]. J1939 was invented to help industrial systems made by different manufacturers work together by having a standardized data field and to help diesel engines meet regulations by expanding on the ECU controlling the engine. Over time, J1939 became common in industrial ground vehicles, primarily in the trucking industry, though it is also used in some weapon systems, agricultural equipment, MRI machines, and an eclectic collection of other systems. Generally, any system that uses a diesel engine control module uses J1939. As for its transmission structure, J1939 uses an extended CAN 2.0 packet which can be seen in Figure 2.1. J1939 splits the ID field into a priority field, Parameter Group Number (PGN), and source address field, with an optional destination address. The PGN identifies the kind of message, such that a receiver knows how to translate the data fields, which are called Suspect Parameter Numbers (SPNs). The split up ID field makes it easy to identify what kind of message is being sent, how important it is, and what device is sending it. However, the identifier field is easily spoofable by any attacker. The data itself can be split unevenly across the data field. Figure 2.2 shows the contents of a common J1939 engine message. NMEA-2000, which is primarily used in naval vessels, uses the same structure but adds a few PGNs and SPNs. For the sake of clarity, we refer to PGN as the message type, the frame ID as the message ID, and SPNs as data parameters,

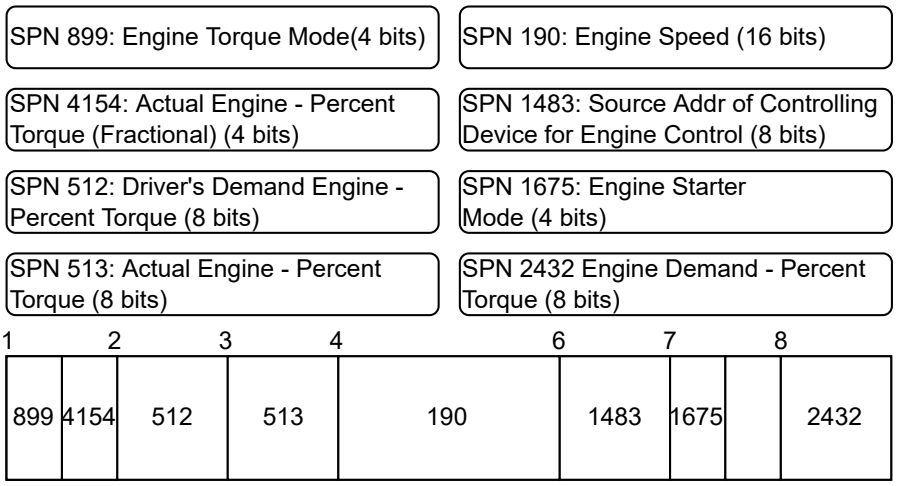


Figure 2.2: J1939 SPNs for PGN 61444, Electronic Engine Controller 1. Numbers 1 through 8 represent the location of the different SPNs, from byte 1 to 8.

or simply data, for the rest of the thesis. J1939 operates at 250KBits/s, making it a particularly slow protocol despite being used in most industrial ground vehicles.

The J1939 standard also adds a networking layer which enables a few attacks [55], and is easily faked by an attacker. Devices either claim an address using a particular PGN, or by having hardcoded addresses on startup. When claiming an address, a device will transmit an 8 byte chunk of data called a 'NAME' which uniquely identifies the source. Similarly to arbitration priority, the device who already has the address and the contesting device compare NAMEs and the lower NAME gets the address. The kicked off device will either claim a new address or send one address unclaimed message onto the bus and then stop speaking entirely. The outcome depends on whether the engineers designing the ECU accounted for networking logic. As for the addresses themselves, J1939 standardizes them for common ECUs based on their priority. Since arbitration is the same in J1939 as it is in CAN, lower source addresses have greater priority for speaking. For example, the engine is always source address 0 and so will always speak over another device trying to transmit the same PGN (type of data) as it. One can imagine how an engine manufacturer took this information and concluded they could hard code claiming address 0. The problem is that if an ECU is not capable of claiming a new address then it can be trivially taken offline, we refer to this as an address hijack attack.

2.3 MIL-STD-1553 Background

MIL-STD-1553 is a serial data bus protocol developed in the 1970s for fighter aircraft, though it has also been used in satellites, nuclear enrichment facilities, and the BART in San Francisco [102]. The procotocol received major updates in 1975 and 1978 for increased standardization, and 2018 to put a focus on adding cyber security to devices connected to the bus. The core design philosophy was providing a common message format for avionics computers to exchange data, with support for a redundant bus in-case one of the avionics computers is destroyed. These avionics computers are referred to as Remote Terminals (RTs), and the protocol can support up to 31 on a single bus, though a single remote terminal can be connected to multiple subsystems. Rather than have an arbitration process where different RTs compete to exchange data, MIL-STD-1553 has a Bus Controller (BC) which sends or requests data from each terminal. The BC is typically tied to a mission computer subsystem.

In terms of the physical layer, MIL-STD-1553 uses shielded twisted pair wiring for the bus with termination resistors and stub connections for each RT. The stub connections are either transformer coupled or direct coupled. The transformer coupling method puts isolation resistors separate from the RT, while direct coupling puts the isolation resistors inside the RT. Either way, the isolation resistors are connected in series with the RT and main bus line. Direct coupling is less resilient to shorting issues, and has a maximum length of 1 foot compared to the 20 feet of transformer coupling. It is generally advised to use transformer coupling. MIL-STD-1553 also supports up to two redundant bus connections which are completely independent wire pairs. All RTs are connected to all redundant buses, though only one bus should be active at a given time. Transmissions use Manchester encoding, with a peak to peak voltage of 18-27V. The bus operates at a bit rate of 1Mbit/s.

There are three components of MIL-STD-1553 messages, which are called words. These words are depicted in Figure 2.3. Each word is 20 bits, where the first 3 bits are a sync waveform that indicates the start of a word, the middle 16 bits are the actual contents, and the final bit is always an odd parity bit.

A command word acts as a sort of packet header so that receivers know what kind of message is about to be sent or requested. Importantly, all command words are started by the bus controller. A command word contains the address of the remote terminal it is addressing, a bit to indicate if the RT is going to transmit data or receive data, the subaddress to indicate what service or functionality the BC wants to access (similarly to ports in TCP/IP networks), and the number of data words that are going

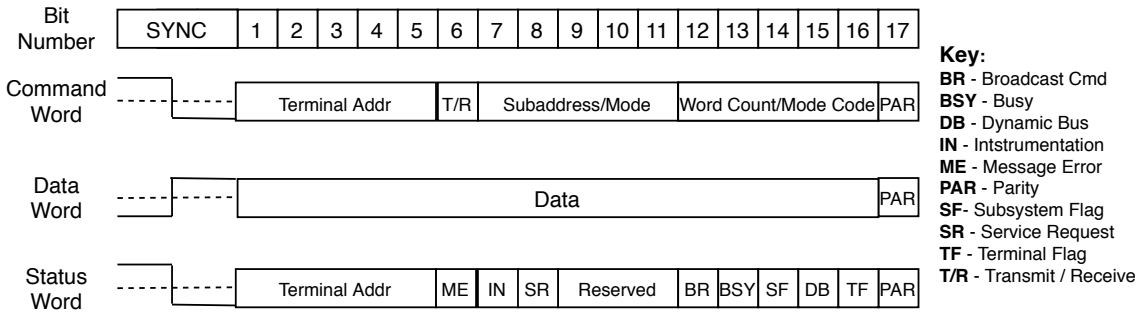


Figure 2.3: The content of the three MIL-STD-1553 word types: command, data, and status words. A standard transceiver passes along bits 1-16, with the sync and parity being processed out.

to be sent. The subaddress can also be set to 0 or 31 to indicate a mode code. We elaborate on mode codes later in this section.

The next word is the status word, which RTs transmit in response to command words, both so the BC can check that the correct RT is responding and to see if a RT is reporting any errors. A RT can report 8 different flags through the status word. These are useful for identifying errors on the system, but difficult to rely on if an attacker can in any way manipulate the contents of the bus. Message error is set when there is an error in the message or an illegitimate command. If message error is set, then none of the data received in the message is used. The instrumentation bit is used to differentiate between status words and command words since they start with the same sync waveform. If it is used, then it is a 0 in the status word, and 1 in the command word. However, this restricts the number of subaddresses available to the command word and so this feature is sometimes not used in modern devices. Service request is set to 1 when a RT indicates it needs servicing. Broadcast command is set to 1 when a broadcast command word is sent. This suppresses the transmission of the status word, but a BC can request the that a RT transmits their last status word to verify that the RT did not transmit a status word because of this bit, as opposed to some malfunction. The busy flag indicates the RT is too busy processing data and cannot keep up with the speed of the bus. It used to be set by the RT but is now set at the request of the BC when it knows a RT will be busy. A BC sets the busy bit to reduce latency issues on the system. The subsystem flag is a logical OR of all the subsystems the RT is connected to, informing the BC if any subsystem has health issues. The dynamic bus control flag informs the BC if the RT received a command to become the bus controller. Finally, the terminal flag informs the BC of an error within the RTs own circuitry.

The data word is relatively simple. It is 2 bytes of data per data word, with the command word dictating how many data words are sent. A maximum of 32 data words (64 bytes) can be sent in a single message.

The MIL-STD-1553 protocol allows for a number of advanced functionalities, but for a security tool the most important features are the ones required by the protocol. One of these functionalities is the mode code, a message, usually with no data, which orders a RT to do some action. Four mode codes are required for every MIL-STD-1553 computer: transmit [the last] status word, reset remote terminal, transmitter shutdown, and override transmitter shutdown. The first is used for diagnosing errors, while the rest are used to turn off RTs that are in a failure state such that they do not monopolize the bus or risk the safety of the system by transmitting erroneously. Their implementation in every system means that they are a helpful tool for an attacker. Mode codes are an easy way for the attacker to disable any RT on the system.

The other required functionality is that each RT must be able to determine if a message is valid. RTs do this by validating: the sync waveform, the Manchester encoding, and that each word is 16 bits long with a valid odd parity bit. These mechanisms are useful for detecting transceiver failures and bit errors, especially for MIL-STD-1553 which has no defined bus arbitration process. It is always assumed the bus controller initiates communication and decides who replies, so there is no need to communicate the intent to communicate. Network collisions are rare on the MIL-STD-1553 bus.

Roughly speaking, communication follows four different message patterns in MIL-STD-1555, where a message is a combination of command, status, and data words. The messages are BC to RT, RT to BC, RT to RT, and Mode Codes (with and without data words). These message patterns are depicted in Figure 2.4. Each message starts with the bus controller transmitting a command word which specifies which RT it is speaking to, whether the BC is expecting to receive data or send data, which RT capability the BC is targeting, and how much data (if any) is going to be sent. The RT responds with a status word containing its own address and a series of error flags to let the bus controller know if everything is fine. Data words are sent after the status word is the RT is transmitting, otherwise they are sent before. RT to RT communication is essentially the bus controller telling a RT to get ready to receive data, then telling another RT to transmit data. For all communication, the RT has 12 microseconds to respond with a status word before generating an error. Between messages there is a minimum of 4 microseconds. Beyond the four main message patterns, there are broadcast variants for BC to RT, RT to RT, and Mode Codes. Broadcasts are

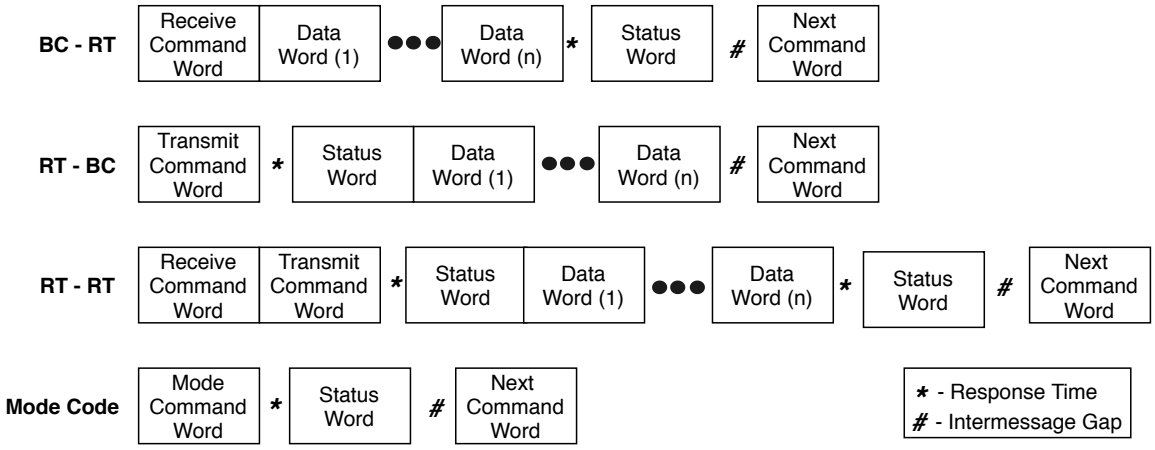


Figure 2.4: The possible patterns of traffic for MIL-STD-1553 messages. Each message begins with a command word initiated by the bus controller, then varies by whether the bus controller is transmitting or receiving data.

communicated to every RT and receive no status word response as that would clog the bus.

The pattern of traffic for MIL-STD-1553 messages is typically based on a major/minor frame cyclic transmission scheme. Each message is grouped by how often it needs to be sent. Minor frames, composed of a small number of messages, are sent as frequently as the most frequent message. Once every message has been sent once, this is called a major frame. Major frames typically last approximately 1 second. Generally, dead bus time is left at the end of every minor frame for aperiodic messages or for general upkeep by the BC. For instance, a BC may ask an unresponsive RT to transmit the last command and status word it received for error detection purposes. However, some implementations, particularly modern 1553 transceivers, inject aperiodic messages at any point in a minor frame instead of waiting for the dead bus time at the end of a frame. Depending on the manufacturer, MIL-STD-1553 message order may be determined by the bus controller prioritizing messages on the fly. The multiple methods for MIL-STD-1553 message sequencing make it difficult to build a security system around the protocol's message patterns.

2.4 ARINC-429 and ASCB Background

ARINC-429 and ASCB are avionics serial data bus protocols used in aircraft like the Boeing-737 and Dornier-328. We do not dedicate a full content chapter to these protocols like we do for CAN and MIL-STD-1553, but they still provide useful comparison points as we discuss protocol-centric security systems. We briefly describe

their key features and describe any other relevant details as they are brought up in Chapter 4.

The key feature of ARINC-429 is that it uses one-way peer to peer connections rather than a singular shared bus. This makes integration difficult as an IDS needs to connect to multiple ARINC channels, each of which use overlapping message labels that are translated differently based on what device transmits the message. From a security perspective the isolation of each wire makes analysis simpler, as it removes the competition and noise that is typically associated with bus traffic. Finding anomalies when only one device is meant to transmit on any particular wire is much easier.

ASCB is a bus controller framework which transmits a fixed loop of messages, which the protocol refers to as frames. The protocol refers to different groups of messages as frames 0 through 7. ASCB loops through the frames in sequential order. At a protocol level, ASCB is an interesting example because it strongly limits when an attacker can actually transmit a message. In theory a detection system could be as simple as checking that the order of messages remains constant. In practice, ASCB is an interesting case of how implementation differs from the written protocol. In our datasets we observe messages cutting off mid-frame, the frame number jumping around instead of incrementing from 0 to 7, and an inconsistency in which LRUs actually respond. This inconsistency between protocol documentation and implementation is a theme we see frequently throughout this thesis.

2.5 Attacking Serial Data Bus Networks

Any attack on a serial data bus network has two components: the attack vector and the attack payload. The vector determines where the attack came from, and this then informs what attack payloads are possible. The most common attack vectors are implants, remote attacks, and supply chain attacks. An implant is a new device connected to the network; a remote attack is a remotely corrupted existing device; a supply chain attack is a modified existing device where the modification occurs before the manufacturer gains ownership of the device. The attack payload itself, which we often refer to as simply 'the attack' in this thesis, is either an exploit designed to gain control of a device or a normal serial data bus message. Either way, the attack is being communicated using the normal protocol process. The lack of authentication on these networks makes attacks fairly simple. Let us imagine an attacker wishes to control the speed of an engine. All an attacker has to do is impersonate the engine, and then send the same message the vehicle uses to control engine speed. Impersonating another

device is called 'spoofing'. If an attacker has taken over a device which was already installed on the system, then we refer to the device as corrupted. Throughout this thesis we expand on how different attack vectors and attacks require different IDS capabilities. But first, we demonstrate how an attacker with bit-flip capabilities can bypass detection systems and take control of the network.

2.6 Threat Models

This thesis assumes an attacker is capable of attaching physical implants, remotely corrupting an existing LRU, and gaining control of a corrupted device before it is installed in the system. We assume that with physical access an attacker can add a new device and that the new device can have custom hardware. An attacker can also use this window to replace an existing LRU with a corrupted one. A core assumption is that the attacker is always adding or removing a device but not interfering with the wiring itself, as that would make any network data impossible to trust. Our threat model is slightly tweaked between chapters. Chapter 3 is focused on an attacker using custom hardware to flip bits on the wire. Custom hardware is unusual in existing work, though it is a reasonable assumption given the more sensitive nature of the industrial and military systems we focus on. Chapter 4 provides an overview of potential security sensor deployment locations and how they cover different threat models. Chapter 5 assumes that any device can be corrupted or use custom hardware. This chapter assumes more sophisticated attackers than existing work, without the need for two attacking devices [12], and the ability to predict when messages are normally sent [86]. In chapter 6 we assume the bus controller cannot be corrupted by a supply chain attack or remote attack vector. This is akin to existing MIL-STD-1553 focused work [88] [29], as corrupting the bus controller cuts out any security options beyond modeling and simulating the overall system.

Chapter 3

Abusing the CAN Error Process with Bit-Flip Attacks

For over a decade researchers have published work on how insecure the CAN bus is [42], [10]. It has no authentication on any messages, this means that any attacker who can transmit on the serial data bus network gains control over the vehicle. In response to this lack of security, researchers published security systems and intrusion detection systems [12], [14], [68], [72], [76], [58], [33]. In this chapter we propose a stealthy attack which bypasses these detection systems by producing no visible impact on the bus: no extra messages, no error frames, and no changes in timing. This new attack abuses the CAN error process in combination with bit-flip attacks to silently force an ECU into a state known as 'bus-off'.

The bus-off state turns off an ECU's transmitter. ECUs enter the bus-off state after a number of erroneous messages. The number of errors necessary to reach this bus-off state vary, with a minimum of 32. We expand on how the entire CAN error process works in Section 3.1. Regardless, if an attacker has a way to produce errors, then they can control the bus by interrupting messages and eventually having victims enter the bus-off state. This allows an attacker to effectively replace the victim on the bus, similar to a supply chain attack.

We propose a novel attack which allows an attacker to push a victim into the bus-off state without producing any errors on the bus. It does this by flipping multiple bits in a single message, rapidly increasing the error count while overriding their victim's error messages and setting the appropriate bits such that other ECUs register it as a valid CAN message. However, the technology to flip bits is inherently complex and may not be 100% accurate. We introduce a probabilistic attacker, which has a non-100% chance of success at flipping any particular bit. Through this imperfect attacker, we can understand how reliable an attacker needs to be at flipping bits to

take a victim off the bus, and avoid producing visible errors. We find that even with just 38% accuracy for flipping bits, an attacker can silently take a single ECU off the bus within a single message with over 99% accuracy. More details on this probabilistic attacker are in Section 3.4.

Our stealthy attack assumes the attacker is in a position where they can read the bus, wait for the appropriate time, then deliver the attack while modifying it on the fly. But this is not true for all attackers. Limited ECU exploits may only allow an attacker to transmit a fixed payload, as they do not have a large enough buffer to adapt the attack to the bus. Alternatively, an electromagnetic induction attack may remotely flips bits on the line with no way of knowing the impact of those bit-flips, or even if the flips succeeded [80]. As a result, many of these attacks rely on the bus being silent when the attack begins so they do not collide with other messages and mangle their own attack. We propose a new methodology to synchronize the bus for blind (write only) attackers, such that no matter what state the bus is in, the bus is manipulated into the start of a CAN packet. This changes the chance of success for blind attackers from the percentage of dead bus time, to 100%, making blind attacks a viable threat against the CAN bus.

Given how effective and stealthy our weak probabilistic attacker is, it is clear we need some way of detecting, or in some way preventing bit-flip attacks. Existing intrusion detection systems provide three options: timing analysis, data analysis, and voltage analysis. Timing analysis is ineffective, as bit-flips produce no timing differences between messages [86], [12]. Data analysis gambles on how much the attacker is changing the data and is inherently limited when the attacker controls every message the victim transmits [72], [98]. Voltage analysis is the most likely to succeed, but is historically prone to false positives in varied environments [14], [13], [41], [27] and requires retraining which can then be exploited [5]. More importantly, the exact tuning for how sensitive the voltage based intrusion detection system needs to be is a constant cat and mouse game based on evolving attacker capabilities. The ideal security update limits how the attack can be done, such that detection is guaranteed.

The inherent problem our stealthy bus-off attack exploits is that interrupting an error results in another error. We propose changing the error process such that each error only counts as one error. This means that if an attacker causes an error on the bus, that error will continue forever until it is allowed to complete. Our small change has no impact to the safety of the bus, but it changes how effective and how stealthy the attacker can be. By having error frames go until completion, the attacker now must be 99% accurate at flipping bits to have even an 80% chance at completing

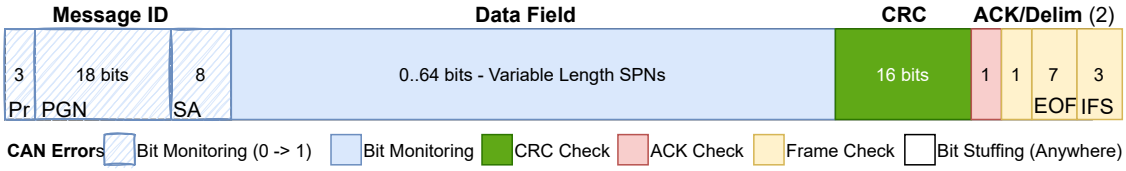


Figure 3.1: An extended CAN Packet with error sections highlighted. Bit Stuffing errors can occur anywhere in the packet while other errors are based more on the location. Generally errors occur in response to bit-flips, or CRC checks.

a single CAN compliant message. Now the attacker has to allow error frames to complete to have their victim enter the bus-off state.

By forcing the attacker to complete error frames to enter the bus-off state we effectively prevent our stealthy attack, ensuring no ECU can be completely disabled without providing an indicator of compromise on the bus. To account for error prone systems, we propose heuristic approaches for differentiating between error frames produced by an attacker and those produced by malfunctioning systems, as well as a method for detecting an attacker endlessly controlling the bus with bit-flip attacks.

We summarize our contributions as follows:

- A novel attack allowing an attacker to silently disable an ECU within one message by flipping bits, as well as a statistical analysis showing they only need 38% accuracy while flipping bits to have over 99% accuracy in silently disabling a victim ECU.
- A novel methodology which synchronizes the bus such that a blind attacker can guarantee the delivery of their malicious CAN messages regardless of the bus state at the time of the attack.
- A proposed change to the CAN protocol's implementation of error frames to remove the possibility for silently disabling ECUs.

3.1 The CAN Error Process

In this section we briefly summarize the relevant parts of the CAN protocol, with the main focus being on how the CAN error process works. We then go into existing attack and defense work, particularly the bus-off attack which uses a similar technique to our stealthy attack.

3.1.1 CAN Error Frames

At a high level, CAN Error Frames are used to cancel out an erroneous message and inform the rest of the bus that an error occurred. From the bus's perspective, this looks like six consecutive bits, usually six 0s, followed by an error delimiter of eight 1s to inform other ECUs the bus is now free, before finally retransmitting the erroneous message. The CAN error process works by having each ECU monitor the bus for specific problems and flag any errors they see. If an ECU registers enough errors, then that ECU turns off its own transmitter to avoid transmitting more errors onto the bus. After receiving enough error-free messages, that transmitter will eventually turn back on, though some implementations require restarting the vehicle to re-enable the ECU. This mechanism is designed for the safety of the vehicle, such that if any ECU frequently talks over other messages or is unable to properly transmit a signal, then it stops inhibiting the rest of the bus.

Let us examine what CAN considers an error. There are five possible errors [45]: bit monitoring, checksum check, bit stuffing, frame check, and acknowledge check. Figure 3.1 depicts where these errors occur. **Bit monitoring** registers an error if the value on the bus is different from what the transmitting ECU believes it sent. **Checksum check** determines if the sixteen bit CRC is valid. **Bit stuffing** checks that whenever five bits of the same value appear, they are 'stuffed'. This means that the opposite bit is transmitted and the transceiver knows to ignore this bit when assembling the CAN frame. CAN Error Frames use bit stuffing to inform the rest of the bus an error occurred. They do this by transmitting six of the same bit in a row without stuffing, raising the bit stuffing error for every ECU so they drop the victim message. **Frame check** monitors any parts of a CAN frame which are meant to be recessive (1) and transmits an error if they are instead dominant (0). Finally, the **Acknowledge check** transmits an error if the ACK field is not set to a dominant bit. Taken together, these errors are monitoring tools for when an ECU is no longer able to determine when to transmit, consistently set bits, or calculate a checksum.

Each ECU tracks the number of errors it sees via internal error counters. Generally speaking, each error results in that counter increasing by eight for the erroring ECU, while each error-free message decrements the counter by one. If the counter reaches 127, then error frames transmit recessive bits instead of dominant bits. This is called the passive error state. Passive errors make it easy for a transmitting ECU to talk over error frames and complete a message under the assumption the error prone device is in the wrong. Interrupting an error frame increments this counter by eight again. If the

error counter reaches 255, the ECU enters the bus-off state, turning off its transmitter. While this mechanism may make sense for safety purposes, it is a clear attack vector.

3.1.2 Avoiding Detection with CAN Errors

There is a great deal of work on hacking the CAN bus [42], [10], [8], [77] [18] remotely [56] [52] or with an implanted physical device. Generally this work relies on the premise that there is no authentication on the bus, so any transmitted message has some control of the vehicle. The problem is that then the attacker is conflicting with the existing devices on the bus.

As the field of CAN intrusion detection has matured over the last decade, this flood of new attacker messages presents an indicator of compromise. Several pieces of existing work are designed to detect new messages on the bus, usually through timing characteristics, such as the timing intervals between messages [86] [12]. This technique relies on legitimate ECUs transmitting periodically. The way around this is to prevent the legitimate device from speaking, that way the attacker always looks correct.

There are a few ways to prevent a legitimate device from speaking. Uploading malicious firmware to the victim device is effective for pivoting from the attacker system to the victim, but malicious firmware is simple to detect with a robust maintenance process. Industrial CAN standards allow for single network messages to disable a bus, but the attacker must use a very particular message, with a data field never seen in normal traffic, making it easy to detect [55]. In terms of silencing an existing device, the best way to do it in CAN is through a 'bus-off attack'.

The bus-off attack, introduced by Cho and Shin [11], synchronizes with a victim message during the arbitration ID, then transmits a dominant bit during the message to produce a bit monitoring error. By producing this bit monitoring error continuously, the victim enters the bus-off state and the victim transmitter shuts down. The original bus-off attack relies on the assumption that most messages are regularly transmitted as the attacker needs to predict when a message will occur and begin transmitting just before they do. This results in both devices simultaneously winning the arbitration process. Significantly, the bus-off attack allows error frames to complete. Cho and Shin propose a detection system for bus-off attacks based around error frames but rightfully point out that an attacker could slowly execute this attack, making individual error frames a poor thing to alert on. Other papers expand on the bus-off attack to make it more reliable [6], but they do not fundamentally change the attack. Takada proposes counter attacking a bus-off attacker with a bus-off attack [90]. However, many CAN controller drivers provide an interrupt to reset the bus-off state. This means that a

counter attack only works on limited attackers. In this chapter we propose a detection mechanism for the bus-off attack which alerts on exclusively the bus-off state, while also handling a more advanced bus-off attacker, capable of flipping any bit on the bus.

Another possible detection system is voltage based fingerprinting. While this technique is good at detecting new devices transmitting on the bus, or attackers transmitting messages they should not be sending, it does require frequent retraining to account for environmental factors [14], [13], [41], [27], [5]. This retraining period creates an easy attack window for our attacker, who can silence an ECU within the span of a single message (within 300-600 microseconds). Consequently, we believe a mechanism for preventing an ECU from failing silently is necessary, as opposed to a conventional detection system. That said, voltage based fingerprinting solutions would work well in concert with our proposed modification to the CAN error process.

3.2 System and Adversary Model

This section outlines how our system is configured, and the adversary's goals and capabilities, including how we collect traffic and how our CAN Bus is configured.

3.2.1 System Model

Our system model is a single CAN bus, with any number of ECUs connected to it. We assume all connections to the bus are unbroken, such that an attacker is not man-in-the-middling any traffic. All installed ECUs are CAN protocol compliant. We connect a single device to the bus, acting as a dedicated intrusion detection system.

For the purposes of this chapter we assume our system is running a basic allowlist IDS, which alerts if any new arbitration IDs appear on the bus. The allowlist contains all arbitration IDs the ECUs on the bus are programmed to transmit. This simple allowlist allows us to detect bit-flip attackers attempting to cheat the arbitration process, or transmit never before seen messages.

3.2.2 Adversary Model

We define a new attacker called a stealthy attacker. The stealthy attacker is capable of reading and writing to the bus, with a goal of disabling and spoofing ECU(s) without being detected. An attacker unable to meet its goal is deemed unsuccessful.

We assume a 0 to 1 bit-flip has a fixed chance of success from 0-100%, and a 1 to 0 bit-flip has a 100% chance of success. The justification for this is based on the

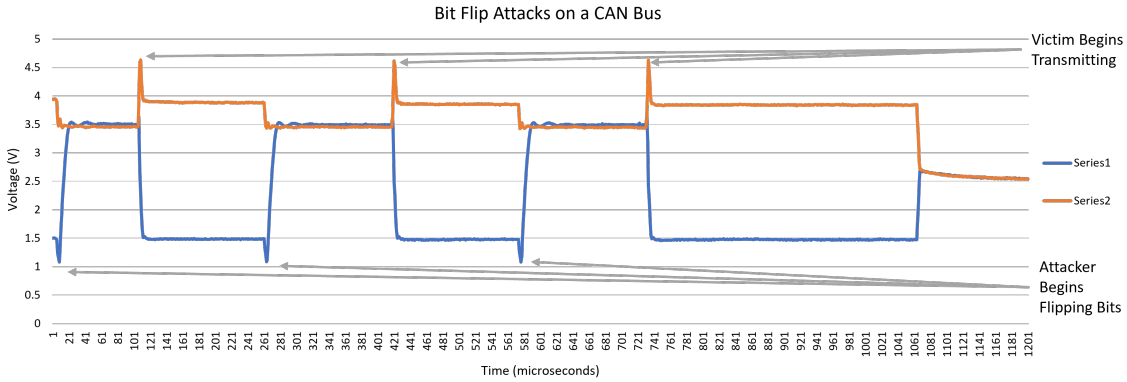


Figure 3.2: Three extended bit-flips attacks on a CAN Bus. The 2nd and 3rd bit-flips occur during an error frame, resulting in another error frame. This demonstrates that an attacker can rapidly produce errors by repeatedly interrupting error frames.

CAN protocol's defining of 1 as the recessive state, and 0 as the dominant. Any ECU transmitting a 0 while another transmits a 1 results in a 0. We expand on the idea of attackers with a not-100% chance of success in Section 3.4.

We make no assumptions about attack vector. It could be remote, an implant, or a supply chain attack as long as it can perform bit-flip attacks. The only restriction is that the attacker's access vector is not the ECU they are attempting to corrupt or take offline. That said, the nature of the attack vector often limits what the attacker can do. Remote attackers especially can have limited payloads which immediately transmit onto the bus. This leads to scenarios where the attacker's payload collides with other messages on the bus and becomes nullified by a CAN error frame. Our solution lies in Section 3.5 where we propose a new methodology to synchronize the bus. This allows even a blind attacker, unable to read the bus, to execute an attack and transmit a message on the CAN bus 100% of the time.

3.3 Stealthy Bus-Off Attack

This chapter introduces a faster, stealthier bus-off attack for our stealthy attacker. In brief, the original bus-off attack produces errors on the bus by performing a single 1 to 0 flip. Each attack increments an error counter by 8, and by executing the attack 32 times over 32 messages the attacker puts the victim into the bus-off state. The attacker can go slower to make detection more difficult, but the gist is they eventually get the error counter to 255 and the victim transmitter turns off. Now an attacker can spoof that victim without concerns over conflicting messages between themselves

and the victim.

Our stealthier version of the attack has the same goal of reaching the bus-off state, but can perform 0 to 1 bit-flips and is unwilling to produce errors on the bus. We also assume the stealthy attacker can arbitrarily flip bits, rather than having to win the arbitration process. The basis of this attack is that a CAN error frame, when interrupted, will produce another error frame, incrementing the victim's error counter by 8 each time it is interrupted. Figure 3.2 demonstrates this effect by performing 3 extended bits flips. The first produces an error frame which we allow to transmit for a few bits before interrupting the error frame with another bit-flip attack. We then repeat this process. Eventually the error frame transmits six bits in a row and is completed.

It follows that if an attacker can continuously interrupt error frames, then they can reach the bus-off state within a single message by flipping 32 bits. Once the victim is in the bus-off state, it will stop transmitting error frames, regardless of if any of them ever completed. Now, our attacker has turned off an ECU without any error frames transmitted on the bus and no more error frames being sent. The only remaining catch is ensuring no receiver errors are produced, both to avoid detection by an IDS and to ensure the rest of the bus processes their malicious message. Avoiding receiver errors means computing the CRC for their manipulated message and ensuring the message ends with 11 recessive bits. Both of these are trivial once the bus is in a bus-off state, as 1 to 0 bit-flips are guaranteed in our adversary model. Once the message is done transmitting, the bus continues operating as normal but now with the victim ECU offline. The attacker is free to spoof the victim, taking the victim's place as its receiver remains online and keeping the vehicle functioning while the attacker transmits malicious payloads to other ECUs. All of this happens within a single message, approximately 600 microseconds, with no indicators of compromise communicated on the bus.

Taking a victim offline within a single message, with no errors, ensures no CAN error based detection systems can detect this attack. Detection systems relying on the timing between messages fail, as the attacker can spoof the timing properties of their victim and the attack creates no timing variation in the victim message [12], [86]. Data based detection systems fail to account for small deviations in data controlled by the attacker (i.e., an engine control module is the root of truth for engine data) [98]. And an attacker can time their attack for the frequent retraining windows of voltage fingerprinting detection systems to avoid detection [5].

3.4 A Probabilistic Attacker

Flipping individual bits is an inherently hard problem on a serial data bus, particularly flipping the bus from a dominant state to a recessive state like our attacker is doing. In this section we display how accurate an attacker has to be to successfully take complete control over their victim. Being successful requires the attacker to produce zero indicators of compromise. This means not transmitting new arbitration IDs, not allowing CAN error frames to complete, and not generating errors in receiving ECUs.

Transmitting new arbitration IDs is trivially detected with a basic allowlist. Allowing error frames to complete provides an indicator that something is wrong on the bus, allowing the attacker to eventually be detected. Finally, any errors raised by receivers will also be raised by an IDS connected to the bus, making it immediately obvious an error frame is occurring, regardless of if the attacker interrupts it on the bus. Receiver errors are also problematic because they disable those receiving ECUs. The ECU transmitting the error has no reason to process any data observed while it is sending an error, as errors tell an ECU to drop whatever message it is processing. The consequence of this is that an ECU transmitting an error is no longer processing any messages on the bus. The receiver's errors will continue retransmitting until either the error frame is completed or the bus-off state is reached. Setting every ECU into the bus-off state is not necessarily a problem if the attacker is prepared to emulate the entire vehicle, but otherwise it is equivalent to a denial of service attack. Our question is if an attacker with less than a 100% success rate can avoid either transmitting a new arbitration ID, completing an error frame, or causing errors in other receiving ECUs. Answering this question reveals how realistic, or unrealistic, our stealthy attack is for attackers with less developed capabilities.

We are concerned with three actions our probabilistic attacker must do to complete a message successfully:

- Completing a CAN Packet (11 flips in a row)
- Interrupting an error frame (at least 1 flip in six bits)
- Transmitting a known arbitration ID

3.4.1 Completing a CAN Packet

The end of a CAN frame requires 11 recessive bits to be sent. This signals every other ECU that another message can now start. If a receiver sees a dominant bit in

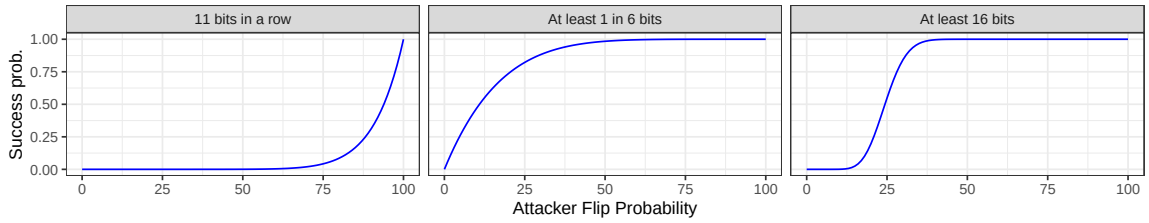


Figure 3.3: From left to right: the probability of a probabilistic attacker to do 11 bit-flips in a row. An attacker unable to do this cannot end a CAN packet without generating errors on non-victim ECUs. The probability of a probabilistic attacker to do at least 1 bit-flip per six bits. An attacker unable to do this cannot reliably prevent errors from being observed on the bus. The probability of a probabilistic attacker to perform at least 16 bit-flips in the 64 bit data field. An attacker unable to do this cannot bypass the need to perform 11 bit-flips in a row, or guarantee no errors during the arbitration process.

this period, then it will transmit a frame check error frame. The result is that if an attacker cannot successfully perform 11 dominant to recessive bit-flips in a row, then the rest of the bus (all receiving ECUs) will stop functioning until an error frame completes or the bus-off state is reached. The first image of Figure 3.3 demonstrates the likelihood of flipping 11 bits in a row for different attacker probabilities. We can see that while this attack is possible, if the attacker has less than 94% accuracy it quickly becomes worse than a coinflip. However, the attacker has another way to more reliably finish CAN frames. If the victim is put into the bus-off state or error passive state before the final 11 bits, then the victim will be either silent or transmitting a 1. These are effectively the same thing. The result is twofold: the attacker needs to do nothing to end a CAN frame, and bit-flips are easier to execute as the 1 to 0 bit-flip is built into the CAN protocol. A standard CAN transceiver can theoretically do this flip. We assume the attacker can flip a 1 to a 0 with 100% accuracy.

The only caveat is the attacker's ability to perform at least 16 bit-flips to reach the passive state before the CRC check. The number of bit-flips necessary to transmit an updated CRC depends on all prior successful bit-flips. However, much like the final 11 bits of the CAN frame, transmitting a correct CRC is trivial if the bus is in a passive error state. A few system specific bits in the arbitration ID aside, this translates to the ability to flip 16 of 64 bits. The third image of Figure 3.3 demonstrates that most attackers can do this with a high degree of certainty. The data field may be somewhat nonsensical with lower probability attackers, but after one strange message the attacker will completely control the victim ECU.

3.4.2 Interrupting an Error Frame

The ability to interrupt an error frame represents the ability of an attacker to continue transmitting a message without letting an error frame complete. An attacker needs to interrupt error frames to ensure their manipulated message is not dropped by the bus. Interrupting an error frame allows an attacker to pretend to be an ECU without producing any indicators of compromise. Eventually interrupting the error frame will put the victim ECU into the bus-off state. Fundamentally, interrupting an error frame translates to an attacker flipping at least one of every six bits. The second image of Figure 3.3 demonstrates that most attackers can interrupt a CAN error frame with a high degree of certainty. However, this comes with two caveats for our stealthy adversary.

The first is ensuring receiver errors (CRC, ACK, Stuffing, Frame Check) do not occur, as an IDS can easily detect it. Caveat two is based on bus arbitration. We will go into more arbitration issues later in this section, but in this case the main issue is interrupting another ECU attempting to transmit a message. If an attacker flips a 0 to a 1 where another ECU is transmitting a 0, they produce an error frame. The error frame continues until the attacker allows it to complete or the non-victim ECU enters the bus-off state. Both of these caveats are avoidable by having the victim ECU in the bus-off state by the arbitration process, or failing that, in the passive error state. For the purposes of avoiding CRC, ACK, Bit Stuffing, and Frame Check errors the passive error state makes it easy for our attacker to transmit the appropriate bits. For avoiding arbitration issues, the passive state also makes the arbitration process function normally. Once again, the final image of Figure 3.3 demonstrates that reaching the recessive error state is attainable for most attackers by flipping at least 16 bits during the data field. In the case of starting a new message, at least 16 bits in the data field of the previous message.

3.4.3 Transmitting a Known Arbitration ID

Finally, we must consider the attacker's ability to transmit a realistic arbitration ID. If an attacker transmits a message with an ID which has never been seen on the bus before, it is trivial for any on-looker to identify that something is wrong on the bus. It follows that our attacker must transmit an arbitration ID which is normally transmitted by the victim ECU. For simplicity, we will assume the attacker has a list of all known message IDs. The easiest and safest approach to this problem is to assume the victim is in the passive error state, and to perform arbitration normally.

Doing otherwise risks accidentally flipping a non-victim ECU's 0, or failing to flip a particular bit and ending up in a scenario where no valid arbitration IDs exist.

3.4.4 Summary

From our three scenarios we can conclude that an attacker capable of flipping bits with as low as 38% accuracy can successfully put a victim into the bus-off state, complete the message, then transmit as that now silent victim over 99% of the time. This is done without non-victim ECUs transmitting any error frames, any error frames completing, or any other indicators of compromise being transmitted on the bus. This shows that this stealthy attack is realistic, even with an adversary with under developed technology that works less than half of the time. The existing error frame process is insufficient for detecting attackers flipping bits on the bus. Thus, we conclude this process needs to be redesigned to handle actual attacks. But before discussing potential redesigns, we will further prove out the exploitability of the CAN error process by looking at blind attackers and how errors make them more reliable.

3.5 Blind Sync Methodology

This chapter introduces a new methodology which synchronizes the bus for blind, write-only, attackers. The fundamental problem of attacking a serial data bus without reading it first is that the attacker has no way of knowing what state the bus is in. Let us take the example of a remote attack where the attacker flips arbitrary bits on the bus using electromagnetic interference. Assume we are using a CAN bus with an extended ID field and 70% bus utilization. Thirty percent of the time the attacker transmits a full message, no messages are being transmitted on the bus, and the attack is successfully transmitted. The other 70% of the time, the attacker message is mangled as it collides with another message and an error occurs. If the attacker continues flipping bits they will likely put that random transceiver into the bus-off state. This is not ideal for actually controlling the victim vehicle, only performing a denial of service attack. If we want to actually inject an attack, then we need to guarantee that our attack results in transmitting a complete message. To do so we need to make that 30% chance a 100% chance by ensuring the bus is silent regardless of what state the bus is in when the attack starts. This means we need to find a series of bits that an attacker can transmit to force the bus to be either silent or at the start of a frame, regardless of what state the bus is in when the attacker transmits.

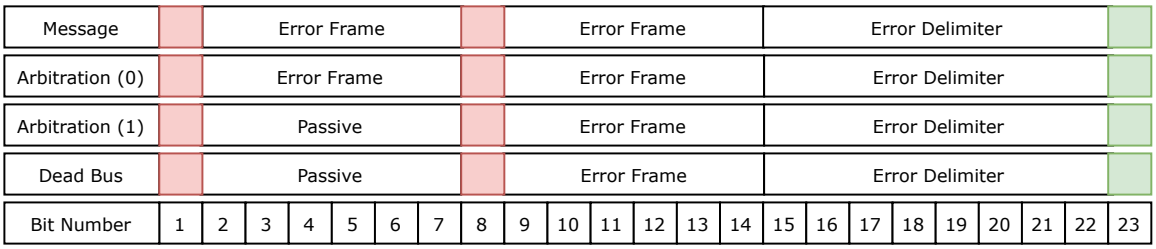


Figure 3.4: The Blind Sync Methodology which synchronizes the bus without reading any data, regardless of when the attack is executed. The red boxes at bit 1 and 8 indicate bit-flip attacks. The green boxes indicate the point where the bus is synchronized.

3.5.1 Synchronizing the Bus

For CAN we break the possible states into 4 categories: during a message, arbitration (0), arbitration (1), and dead bus. Figure 3.4 depicts bit timings for each of our 4 states, with red representing an attack and green representing the time to start transmitting. Let us begin with transmitting during a message. Interrupting a message results in an error frame, after this error frame an error deliminator occurs for 8 bits, then normal operation occurs again. Error frames are an ideal way to synchronize the bus. Interrupt a message once, let the error complete, and then the attacker is confident in the state of the bus until the next error occurs. The trick is lining this error up so it syncs up with the other states. Flipping a 0 bit during the arbitration process causes the same result. However, flipping a 1 bit during arbitration results in the attacker winning arbitration. The attacker cannot effectively use this control of the bus, as they have no way of knowing how far into the message they are. Instead, the attacker needs to intentionally produce an error to once again try to synchronize the bus. Because they won the arbitration process, there is no other ECU transmitting a message that can be interrupted to produce an error frame. Instead they need to stop transmitting for six bits. This causes six consecutive 1s to appear, resulting in a bit stuffing error. Summing the attack bits, the error frame, and the error delimiter results in 21 bits. arbitration (0) and message errors only last 15 bits. To sync this up we need to add another bit-flip attack. Interrupting any part of a CAN error frame or the error delimiter results in another error appearing. By adding an attack on bit 8, right as the bit stuffing error frame starts and just after the bit monitoring error frame ends, we extend the length of the bit stuffing error frame by 1 bit and create a new error frame for the message and arbitration (0) case.

Now the message, arbitration (0) and arbitration (1) cases are synchronized. This

leaves the dead bus case. The attacker cannot simply transmit a message during the dead bus time, as that only works 30% of the time. Instead, they need to intentionally produce errors to sync with the rest of the bus. This case works the same as the arbitration (1) case, except the initial bit-flip starts a CAN frame. Considering the bus was silent, no other devices are competing to speak. Thus, the attacker can copy the same technique as arbitration (1), where they stay silent for six bits and raise a bit stuffing error. These techniques have the same timing property. Now by flipping a bit, waiting six bits, then flipping the next bit, the entire CAN bus is synchronized until the next error frame. The only evidence of this attack is a single error frame completing, possibly two depending on the level of information provided by the CAN controller. The blind sync methodology can be used for any attack where the attacker cannot read the bus and cannot queue messages.

3.5.2 Transmitting An Attack

Once the bus is synchronized, the blind attacker can transmit any message they wish with the caveat that they are likely to produce errors unless they win the arbitration process by setting the first few bits to 0. This will create new arbitration IDs on the bus which makes the blind attacker easy to detect, but only so far as knowing an attack occurred. Traditional CAN logging has no mechanisms for identifying how this attack occurred. The other option is picking a high priority but low frequency message. Now the attacker is likely to win the arbitration process and will not accidentally share that win with a legitimate ECU. Regardless, our blind sync methodology ensures a blind attacker can have their desired impact on the vehicle 100% of the time, regardless of what is happening when the attack is launched. Our methodology is a vast improvement over the blind attacker's previous 30% chance of success.

3.6 Detecting Bus-Off Attacks Without Bit-Flips

Before exploring how to detect bit-flip attacks, it is important to understand how one might detect the original bus-off attack. This provides insight into the complications of counting CAN error frames. For a normal CAN controller using normal CAN error frames the only notification of an error occurring on the bus is a single bit flag at the start of a message. It is not clear who transmitted the error, simply that an error occurred. This is unhelpful, as normal CAN controllers also drop the message with an error in it. This means that the application layer has no way of conclusively connecting the dots. Without some attribution for errors, it becomes difficult to gauge

how many errors any individual LRU is producing and whether it is close to the bus-off or passive error states. Alerting on errors before these critical points could produce false positives, as an error could be indicative of a one-off bug from a transceiver synchronizing incorrectly or a loose connection. But alerting on these critical points is less error prone, as we assume the system functions in normal driving conditions, which it could not do if it regularly moved into the passive error or bus-off states.

There are several approaches one can take for differentiating one-off error frames from attacker induced error frames. One could: monitor for error frames and develop an internal counter for each LRU, heuristically monitor for a jump in error frames, or modify the CAN transceiver to pass on greater levels of error information. Each technique builds on the knowledge of the one before, providing greater error detection capabilities for our IDS.

3.6.1 Standard Error Counter

A bus-off attack can cause consecutive errors on the bus. A standard transceiver drops any messages with errors, so the only indication that an error occurred is an error flag on the message after the attack. The error flag gives us no way of knowing how many errors occurred. This shows that we cannot always rely on counting error frames whenever we receive a message. That said, our transceiver is still registering these completed error frames as receive errors. Whenever a CAN receiver observes an error, but it is not actively transmitting, the receive error counter (REC) increments by 1 (as opposed to the transmission error counter incrementing by 8). A simple driver call allows our IDS to read the REC register. After each message our IDS reads the REC register. If the register increases by more than 1, then we know that multiple errors have occurred in-between messages. We still have the problem that we do not know where the errors are coming from, but even if the system is prone to errors it is unlikely that any ECU would have multiple errors occur in a row. This leaves the possibility of an attacker slow-rolling their bus-off attack, only executing one error in a row and simply making sure that 1 message of every 9 is an error for their victim ECU. In this scenario receive errors do not give us any extra information. Instead, we need to use our knowledge of the CAN error process.

When a message is interrupted with a CAN error frame, the victim ECU immediately attempts to retransmit the message. At the same point our IDS's receiver sets a flag indicating the last message contained an error frame. This allows us to extrapolate that whenever a receiver gives us an error flag, the error flag is likely paired to the message our IDS is receiving. An attacker could delay the retransmission by sending

a message with a higher priority, but the extra message would be trivially detected by spoofing detection systems [86]. Our IDS implements its own error counter for each ECU. Every error free message from an ECU reduces the counter by 1, and every message with an error increases the counter by 8. Our IDS alerts whenever the error counter reaches 128. The main benefit of this approach is that it uses a standard CAN transceiver and is thus applicable to all existing CAN systems.

3.6.2 Heuristic Error Counter

Our second approach compares the number of observed errors to those observed in clean datasets. We assume that regularly occurring CAN errors are consistently coming from the same ECUs. A bus-off attack requires at least 32 errors, meaning we should expect 32 more errors than usual in a condensed period of time. It is important to note that even the slowest version of a bus-off attack occurs over 255 messages. On a standard 250Kbit/s CAN bus, 255 messages takes roughly 0.15 seconds. An attacker cannot slow the process down, or legitimate devices will transmit good messages, decreasing the TEC of their victim system. To train our detection system we split our clean dataset into 255 message chunks and identify the average REC across our dataset. But averaging can provide an attacker a window to execute the bus-off attack. To account for this, we alert on the passive-error state instead of the bus-off state. The passive error state is still indicative of something going wrong on the bus and we can reasonably expect it to never occur in normal bus operation. If the REC ever reaches the trained average plus 16, we alert. This approach has the benefit of accounting for error prone buses.

3.6.3 Modified Transceiver Error Counter

The above approaches successfully detect the original bus-off attack, but they are unable to identify the victim ECU. The root cause of this problem is that whenever an error occurs the message is dropped and we have no other information. To solve this problem, we propose an error detection approach which uses a modified CAN controller to pass error information up the stack. Our custom controller does not drop messages with error frames. Instead it passes along any bits transmitted before the error occurred. This way we can identify what ECU the error is in and use our own error counters for each ECU. We increase the error counter by 8 for each error from an ECU and decrease it by 1 for every good message from that ECU. The attacker could initiate a bit monitoring error before the source address is transmitted in the

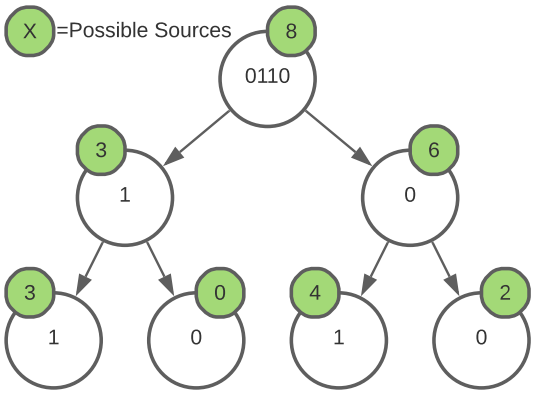


Figure 3.5: A tree view indicating how early into a CAN packet one can narrow down the possible source address of the transmitter without any voltage fingerprinting. The tree starts on the 4th of 29 bits in an extended CAN packet, where bits 21 to 29 indicate the source address. The data is from a data set with 17 sources.

arbitration ID, but we can extrapolate from the bits that were transmitted before the error to determine which ECU was transmitting.

Figure 3.5 demonstrates this logic by applying it to one of our datasets with over 10 million CAN packets. There are 17 ECUs initially, and by parsing the priority field (the first 3 bits) we narrow this list down to 8 ECUs. Choosing another 0 narrows the list to 6, and then another 0 gets us to 2 possible source addresses. We can continue this process to eventually narrow the list to 1 possible source address before we enter the source address field. An attacker can use this technique to reliably silence a single victim ECU as early as possible. A defender can use this technique to identify what device was transmitting or where an error occurred without any voltage fingerprinting. As simple a technique as counting ECU errors ourselves is enough to detect the original bus-off attack. This is because the traditional attack lacks the ability to produce more than one error at once, and cannot drive the bus.

3.7 Bit-Flip Attack Detection

From our probabilistic attacker section it is clear that most attackers capable of flipping bits on the bus can silently turn an ECU off without any errors appearing on the bus. The attacker can then seamlessly pretend to be that device with on-lookers none the wiser. The critical detail is that the stealthy bus-off attack is only easy for the majority of attackers because they can put the victim ECU into a passive error state. The passive error state makes it trivial to finish CAN messages, perform a

Algorithm 1 Error Resistant Error Frame Error Counter Logic

```
1: lastError ← 0
2: procedure OnError
3:     if lastError > 0 then
4:         pass
5:     else if lastError <= 0 then
6:         TEC ← TEC + 8
7:     lastError ← 6
8: procedure OnBit(bit)
9:     if ErrorInBit(bit) then
10:        OnError()
11:    else
12:        lastError ← lastError − 1
```

normal arbitration process, and interrupt CAN errors. If we can prevent this passive state, then only highly accurate attackers can properly end messages. Based on these conclusions, we require a change to the CAN error process to prevent our stealthy attacker from being able to reliably control their victim. A change that ensures the detection of any bit-flip attacks and prevents attacks from occurring within a single message, even for attackers with 100% accuracy for flipping bits.

3.7.1 Error Resistant Error Frames

We propose Error Resistant Error Frames (EREFs), which change the error process in CAN such that if an error is interrupted then the victim ECU will re-transmit an error frame without increasing its Transmission Error Counter (TEC). We demonstrate the algorithm for EREFs in Algorithm 1, where the 'OnBit' procedure is run whenever a bit is received. EREFs drastically change the 3 scenarios outlined in Section 3.4. For our first two scenarios, it means the easy way out of having the victim stop transmitting 0s midway through the message is gone. Instead, in our 'completing a CAN frame' scenario the attacker must be able to flip 11 bits in a row or an IDS will detect the frame check error and no message will complete. Even if an attacker can drive the entire bus, the transceivers would not actually receive data because every receiving ECU starts transmitting errors. We demonstrate this cascade of errors in Figure 3.6, where EREFs cause all receiving ECUs to continue transmitting dominant error frames and normal error frames cause receiving ECUs to enter the bus-off state. Continuing dominant error frames from all receiving ECUs mean that EREFs make bit-flip attacks unreliable for any attacker that cannot perform 11 bit-flips in a row with near 100% certainty. Based on Figure 3.3, we can see that EREFs make ending a

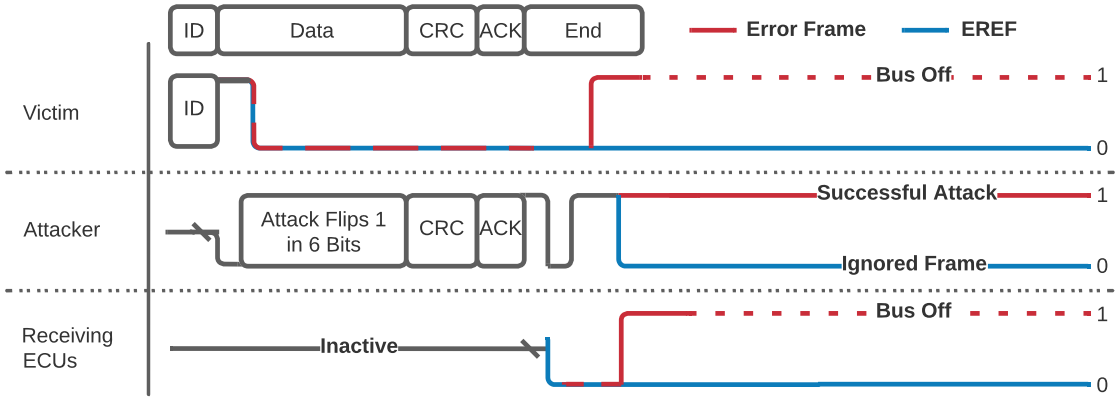


Figure 3.6: The effect of an attacker inducing errors onto the bus where red lines indicate normal error frames and the blue lines indicate error resistant error frames. This shows that an error during the end of a CAN frame results in all receiving ECUs generating errors. The normal error process enters the bus-off state, while EREFs continue transmitting dominant error frames. With receiving ECUs in the bus-off state the normal bus will process the attacker payload while EREFs cause it to be ignored

CAN frame without errors unlikely for most attackers, with even a 99% attack chance only having an 80% success rate over 11 bits. The story is similar for the 'interrupting CAN error frames' scenario. A successful attacker would be assembling a reasonable message, with a valid CRC, while also interrupting 1 of every six bits and adjusting on the fly for attacks that do not work. This is easier than ending a CAN frame, but prone to errors, especially once the attacker reaches the CRC and needs a very specific set of 16 bits. Failure during the CRC means either detection from an error frame completing, or an IDS seeing a receiver error.

EREFs place particularly strong restrictions on the attacker while trying to pick a valid arbitration ID. We can no longer assume the bus is in an error-passive or bus-off mode. This implies the attacker is actively flipping a 0 to a 1 during the arbitration ID process, even when they aren't attempting to transmit a message from their chosen victim ECU. If at any point another ECU attempts to transmit a 0 and the attacker performs a bit-flip, then that ECU will begin to transmit a bit monitoring error frame as its 0 is flipped to a 1. This error ensures that ECU will stop receiving data until the error frame completes. Because the ECU cannot complete an error frame without providing an indicator of compromise, they have no way of re-enabling an ECU transmitting errors. But the attacker must perform bit-flips during the arbitration, not only to prevent the CAN error from completing,

but because otherwise it will transmit previously unseen arbitration IDs which provide an easy indicator of compromise. This means that the attacker must predict the exact order of messages going over the bus, or risk other transmitting ECUs generating errors or transmitting an unknown arbitration ID. This is true for even an 100% accurate attacker, as their reliability does not affect the cascading error problem. The difficulty in starting a message as a result of EREFs demonstrates that EREFs make bit-flip attacks infeasible for our stealthy attacker.

3.7.2 Detecting Driving the Bus

An attacker avoiding detection may end up in a scenario where they are continuously driving the bus, as they are unwilling to let error frames complete. Now, many of the ECUs on the bus may not be functioning, as they are transmitting error frames, but the attacker can still transmit messages on the bus as if ECUs were listening. This is a good way of fooling an IDS. While an observer may easily be able to tell the vehicle is not working, the IDS has no way to know with just bus data. We assume the attacker can emulate every ECU on the bus.

To get around this, our IDS needs a way of knowing if an attacker is driving the bus. We propose having the IDS randomly transmit unused CAN IDs with the first 3 bits being 0. Because the attacker has no way of distinguishing between the 0 of a CAN error frame and the 0 of our IDS transmitting, they will inevitably flip a bit in our IDS's arbitration ID to 1. The random transmission times ensure an attacker cannot predict our security message. This results in our IDS generating an error frame and the attacker being detected.

For the safety of the rest of the CAN Bus, our IDS cannot transmit too frequently. Otherwise, legitimate ECUs may not be able to transmit their data in a timely manner. Given most high priority CAN messages repeat periodically within the 10-100 millisecond range, we believe once every 1 to 5 seconds is slow enough to detect an attacker quickly without impacting bus performance.

3.7.3 Completing Error Frames

Our adversary model assumes that any error frames are an indicator of compromise, and so the attacker is unwilling to let any complete. It follows that EREFs, which practically guarantee cascading errors for an attacker, make bit-flip attacks equivalent to denial of service attacks. However, this does rely on the idea that error frames are inherently rare, which varies from system to system. For example, the original

bus-off attack paper has to produce at least 16 error frames. The authors make the assumption that alerting on any individual error frame would result in false positives, as they are somewhat common in normal traffic [11]. While we have not observed this in our data, we do acknowledge it as a possibility.

Any attack that produces error frames is inherently more detectable by an incident response process because it contains some indicator, while our stealthy attack produces none on a normal CAN bus. Our previous analysis of bus-off detection strategies demonstrates that we can detect an attacker producing error frames and differentiate attacker-induced errors from legitimate errors.

3.7.4 Bit-Flip Detection Complications

We have discussed multiple techniques to detect the original bus-off attack as well as a technique to detect an attacker continually flipping bits to drive the bus. The problem comes when our attacker is okay with errors completing, and also drives the bus for short periods of time. An attacker driving the bus for multiple messages has a small cascading error effect, but the attacker can quickly recover by completing an error frame. Driving the bus for a short period of time ensures our random transmission scheme is ineffective. Additionally, the exact error count for each LRU is unclear because our CAN controller is not aware of when the CAN error started, only when it ended. With neither detection scheme capable of detecting our less-stealthy attacker, we cannot apply a strong security guarantee to systems with common error frames and an attacker capable of flipping arbitrary bits. In Chapter 5 we demonstrate how knowledge of the CAN bus allows a defender to reliably detect the bit-flip attack itself, without a full voltage detection system. This removes the need for EREFs with the compromise of needing additional hardware for the detection system. However, EREFs are still valuable for forcing attackers to execute the bus-off attack in a way which produces indicators of compromise on standard ECUs.

3.8 Stealthy Attacker Security Analysis

This security analysis examines what attacks we are capable of detecting regardless of the attacker having full knowledge of how our detection system works. The goal of the stealthy attacker is to remain undetected while disabling ECU(s). In order to disable an ECU, the attacker must increment the TEC to greater than 255. Flipping a bit on the bus increases the transmission error counter by 8, and transmits an EREF. The stealthy attacker must interrupt the error frame continuously to avoid detection.

Because no error frame is ever completed, this error frame is always transmitting dominant bits. If the attacker generates a CRC, Frame Check, or ACK error, then our IDS, along with all receiving ECUs, generates an error which results in detection. The detection occurs regardless of if the attacker interrupts an error during these times. This is not a problem for our 100% success attacker, but is for all other attackers, with only an 80% chance for the 99% attacker to not produce a frame check error.

Now the attacker has completed a frame by continuously interrupting the EREF. At the start of the next frame the attacker must continue flipping the EREF's dominant bits to produce a known arbitration ID and prevent the error frame from completing. On flipping a bit in the arbitration ID, if another ECU intended to transmit a dominant bit then an error occurs. This cascades the original bit-flip error into another ECU. On another ECU transmitting an EREF, that ECU stops receiving data. As an attacker cannot complete an error frame without being detected, this results in ECUs being disabled through every arbitration process unless the attacker perfectly predicts which ECUs are going to transmit on the bus. The attacker cannot perfectly predict the order of messages on the bus, as shown by existing intrusion detection work failing to do so [86]. It follows that ECUs will be disabled as lower priority value messages are transmitted on the bus. Eventually, our IDS will transmit a randomized arbitration ID with the first 3 bits set to 0, as outlined in Section 3.7.2. Our IDS will always win this arbitration process in a non-error state. The attacker can not differentiate the 0 from our IDS from the 0 of an EREF, and will flip it to transmit known messages. This flip produces a bit monitoring error in our IDS, allowing us to detect the stealthy attacker. In summary, EREFs ensure our stealthy attacker has no way of succeeding in its goals, as it is always detected.

3.9 Discussion

This chapter demonstrates the effectiveness of using the CAN error frame to hide and enable attacks. The fundamental problem is the inability of ECUs to reliably communicate their own error state on the bus. Any attempts to communicate what error state an ECU is in would be overwritten by an attacker capable of flipping bits or simply denying messages. This can be solved a couple of ways. Adding authentication to the bus means that an attacker cannot overwrite a bit without knowing some secret key. The secrecy of that key is in question for corrupted ECUs, but stealthy bit-flip attacks require custom hardware. This makes it unlikely they would be corrupting an existing device. However, adding authentication requires modifying ECUs and uses up

valuable bus bandwidth. The other option is a secondary channel for communicating errors. This approach also requires modifying ECUs and comes with additional wiring costs. Arguably, this approach just moves the attack somewhere else. The simplicity of EREFs is the main appeal over these other two techniques. EREFs are a mechanism for ensuring the protocol works as intended, with dominant CAN error frames actually completing. On a normal bus, EREFs do not take up bandwidth or have any visible impact at all. EREFs only appear when the attacker does, making them an ideal security solution.

3.10 Conclusion

In this chapter we propose a novel attack, a methodology for enabling blind attackers, a change to the CAN protocol which prevents attackers from stealthily removing ECUs from the bus, and three techniques for differentiating legitimate error frames from attacker induced error frames. Altogether, this chapter provides a thorough examination of an attacker capable of flipping bits and what they can do on a CAN bus, even if the attacker is incapable of reliably flipping bits. As the CAN protocol exists currently, an attacker can reliably and silently disable any number of target ECUs, even with just a 38% chance of successfully flipping an individual bit. Our addition of error resistant error frames then puts the attacker in a position where they have no choice but to inevitably complete an error frame or have the bus stop functioning. This allows us to detect the attack, while our CAN error detection mechanisms differentiate the legitimate error frames from attacker induced error frames. Outside of detection we propose a novel attack methodology which allows blind attackers to always have an impact on their victim, regardless of what the CAN bus is doing as they initiate their attack. This increases the blind attacker's chance of success from the percentage of dead bus time to 100%. Through the totality of this work, future researchers can more accurately model against bit-flip attacks.

Chapter 4

A Methodology for Designing Protocol-Specific Detection Systems

Chapter 3 demonstrates how effective protocol-specific attacks can be. Our defensive solution is an efficient patch for the aspects of the CAN protocol that enable those attacks. EREFs detect our stealthy bit-flip attacks and adequately address the problem of CAN error frames, but EREFs do not take advantage of the CAN protocol so much as change the CAN protocol. Rather than change the protocol, a defender should be able to do exactly what our stealthy attacker does: exploit the protocol to increase the efficiency and reliability of a technique. In this chapter, we demonstrate how a defender can factor the protocol into their design just like the attacker. We then use the lessons learned from this chapter to produce two intrusion detection systems in the next two chapters. As part of our lessons learned, we develop a five step methodology for designing a protocol-centric intrusion detection system. We summarize the contributions of this chapter as follows:

- An identification of the resources and deployment strategies available to a defender as they construct an IDS.
- An exploratory analysis of common protocol elements to identify elements which simplify IDSes or enable attacks.
- A methodology for combining different IDS technologies while expanding security coverage.

4.1 Protocol-Centric IDS Methodology

In this chapter we discuss IDS techniques, deployment strategies, protocol-based alerting, threat modeling, and IDS design. Together these points help a defender

design and build an IDS from scratch that is best suited to their own needs. We can summarize the actions a defender needs to take into five steps. The rest of this chapter elaborates on the rationale behind these five steps and how a defender can go about performing each step.

1. Select a deployment option
2. Identify which IDS resources are available for the chosen deployment
3. Analyze the protocol to determine possible attacks and methods for simplifying an IDS
4. Define the threat model, determining what attacks and attack vectors are possible
5. Build the IDS, starting with an allowlist and spoofing detection techniques

4.2 Generic IDS Development

Before diving into how the protocol shapes what detection systems are possible or advisable, we present an overview of the generic, protocol-agnostic options available to the defender. Regardless of the protocol, the defender always needs to decide what data they will base their IDS on and how they will deploy it to the system. The protocol informs which resources and deployment strategies are best.

4.2.1 IDS Resources

We identify four core resources available to any serial data bus IDS: timing, data, voltage, and protocol knowledge. We select these four because they are low level pieces of information that are communicated anytime anything happens on the bus. Every message is sent at a time, every message includes some data it is trying to communicate to the rest of the system, the signal is carried via the voltage produced by an LRU, and the protocol is the puppet master dictating how all of this is done, or at least how it is meant to be done. An attacker has to impact at least one of these components whenever they interface with the bus. Analyzing some subset of these resources for anomalies is the foundation of any detection system. Let us breakdown what kind of attacks each of these resources is best suited to detecting.

Timing - The ordering and frequency of messages on a serial data bus network is usually predictable. An attacker adding new messages to the bus creates deviations in message ordering and message frequency that allow a timing-based IDS to detect

the attacker. Because timing detects the absence of messages or the creation of new messages, it is well suited to detecting spoofing, flooding, and denial of service attacks. However, timing is a poor choice for detecting a legitimate device taken over by an attacker. Taken over devices (corrupted LRUs) can transmit malicious messages without creating any 'new' messages; instead they can transmit the same messages but with malicious data. Timing as a resource has the benefit of it being incidentally collected simply by reading messages on the bus, making it an obvious choice for initial IDS development.

Data - Devices communicate on serial data bus networks in order to share data with each other. From this data a defender can create a physical model for how the system functions and alert on any deviations from that model. This approach alerts on the actual effects of an attack rather than the meta indications that something came from an attacker. As a result, data as a resource for detection is the most expansive in terms of what attacks it can detect, while being the most difficult to apply a robust security analysis to.

Voltage - Physical layer signals are typically used to fingerprint every computer on the bus. This allows a defender to detect whenever an attacker is spoofing another system. Voltage fingerprinting also provides source attestation and passive implant detection. Source attestation allows a defender to know what device is transmitting what. Knowing what device is transmitting allows the defender to safely target any effects from their IDS, e.g, an IPS which uploads clean firmware to a victim. Passive implant detection occurs due to the impedance of the line changing as a new device is connected. Passive detection strategies are a useful way of detecting an attacker before they have time to execute an attack or before they can stealthily collect data. Voltage analysis is typically complicated by the need for additional hardware, as well as a training process which needs constant tuning due to environmental factors [14].

Protocol Detection - The protocol dictates how messages are communicated on the bus and how the system handles errors (e.g., CAN errors). Protocol detection often acts best as a rules-based detection technique for detecting specific attacks and as a system constraint that can produce simpler and more effective detection systems when combined with other resources. Section 4.3 expands on how protocols can be used for detection.

4.2.2 Deployment Options

What detection systems are possible to implement depends on how the IDS is deployed on the system. If the defender has limited access to the vehicle or is limited by the

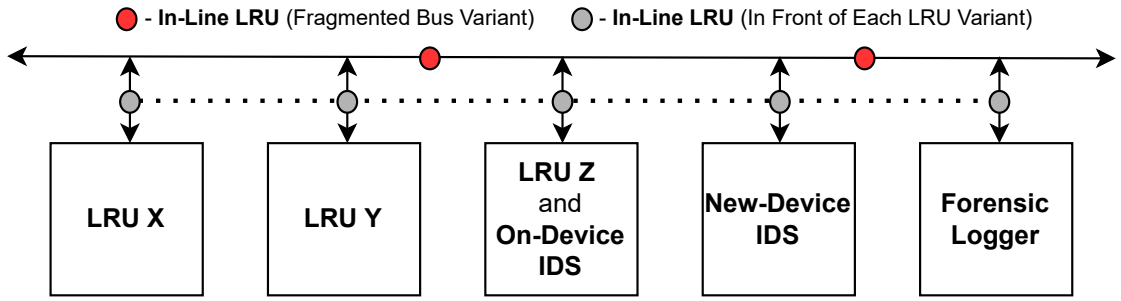


Figure 4.1: A serial data bus with each of our deployment options. The on-device IDS is paired with LRU Z, the new-device IDS is added to the bus by the defender, a forensic logger is also a new device but with no detection capabilities, and the red and grey dots represent in-line LRU deployment variants. The dotted line represents the communication of in-line LRUs.

hardware already on the system, then some detection systems are simply not possible. Picking a deployment option is first in our methodology as it decides which of our IDS resources a defender can actually use. We identify 4 possible deployment options based on existing academic and industrial IDS deployments: on-device, new-device, in-line, and forensic. These deployment options are shown in Figure 4.1.

On-Device takes an existing LRU and loads a security system onto it as a software upgrade. This deployment option is primarily viable for system maintainers or manufacturers who are trying to add security to their own system. It is difficult to use an on-device deployment as an outsider, in part because system maintainers can be hard pressed to modify existing LRUs when they do not have the resources, training, or contractual right to do so [93]. The LRU may not even expose a method for running additional software and require hardware hacking via some leftover UART connection to actually change what is running on the device. But let us assume the defender can make any modifications they want to the system. On-Device deployments come with a few unique advantages over other deployment options. The first advantage from being able to add software to existing devices is adding cryptography to the bus. Cryptography requires a mechanism for exchanging secure keys, this is impossible to add to legacy systems without modifying existing LRUs. The security benefit of cryptography is that it adds authentication to the serial data bus network; authentication ensures an attacker cannot forge or spoof messages on the bus unless they takeover an existing device. If we assume the defender has a unique way of identifying each LRU, then an attacker facing message authentication can only transmit messages their corrupted device would transmit. The second advantage is that an IDS can be aware of what the LRU it is installed on is doing. A recurring problem in serial data bus security

is knowing who sent what [12], [14]. On-Device deployments allow an IDS to easily determine if the device they are deployed on sent a given message. The main negative of On Device deployments is that they are limited by the hardware already on the system. Many vehicles have under powered computers using outdated architectures. These underpowered computers limit how computationally complex the IDS can be. Additionally, on-device deployments lock the IDS to using standard hardware. Most standard chips do not expose voltage information and operating behind a standard transceiver can hide attacks, as the chip drops attacker mangled messages.

New-Device deployments add a new device to the bus as if it were a standard LRU being connected. This deployment strategy is the default behavior for most academic and industry solutions [12], [30]. It is easier to test and deploy new devices to systems than deal with the complications of integrating into an existing system. The main advantage of a new-device deployment is being able to use custom hardware to analyze voltage data and ignore any inconvenient protocol specifications. Custom hardware ensures the IDS does not drop any messages or risk being disabled by a clever attacker. Making protocol changes to an existing LRU as part of an on-device deployment risks violating the safety principles of the overall system. A defender can counter this risk by altering new devices such that they are unable to transmit data. Removing transmission capabilities reduces the risk of the IDS being repurposed by an attacker.

In-Line deployments are essentially new devices, but work by man in the middling traffic. They sit either between individual LRUs and the main bus line or in a configuration which segments the main bus line [40]. This approach has two main advantages: an existing device spoofing another LRU is obvious when there is a detection system deployed in front of every LRU, and prevention is built in by default, as the in-line system could choose to not pass on the message. The main drawback of this deployment style is implementation cost. Taking an existing system and adding a computer between every LRU, or even multiple around the main bus, can be expensive and time consuming. In-Line deployments simply do not scale across large fleets. That said, it can be an extremely effective deployment option for detecting and preventing attacks without complicated detection strategies, only simple communication between in-line nodes. In-Line detection systems are poorly researched, and there are likely many interesting security guarantees that could come from communication between multiple nodes spread throughout the system.

Forensic deployments assume that no detection system actively has to run on the vehicle. All of our previous deployment options are inherently limited by the space

and weight constraints of a vehicle. This limitation impacts the level of computational power an IDS can bring to bear. A forensic deployment deploys a software upgrade to an existing LRU or deploys as a new device. Regardless of the deployment strategy, the data logger only logs data; all data analysis is done at a later point. More computationally intensive approaches, such as neural network and machine learning solutions, are best suited to forensic deployments. Any deployment can, and should, do full logging of data, but what data is logged depends on the design of the intrusion detection systems.

4.3 Using the Protocol for Detection

In this section we explore what aspects of protocol documentation are useful for defenders. Along these lines we discuss how the design philosophy of the protocol can reveal protocol-specific attacks and ways for a defender to simplify their detection system. Additionally, we breakdown common elements of protocols including: normal communication patterns, manufacturer specific information, hardware nuances, networking implementations, and methods for intrusion prevention.

4.3.1 Design Philosophy

The most important aspect of any protocol is why it was designed how it was and why people continue to use it. After all, if a protocol has existed for over a decade then there must be some merit to the design that makes engineers continue to use it. Economic inertia can keep a protocol in use as companies save money by re-using assets, but that only goes so far. Design philosophy highlights what the original engineers believed was important, not only how the system was designed to function, but how they assumed it would function. We can see this play out for some of our different protocols.

MIL-STD-1553 was designed to provide interoperability between avionics computers, such that groups could design avionics computers without a great deal of assistance from the US government. To maintain exacting control over the bus, the designers established a bus controller. This computer initiates all traffic and informs other LRUs when it is their time to speak. MIL-STD-1553's design is focused on maintaining control of the aircraft. For contrast we can look at other avionics systems. ASCB is designed by Honeywell and uses a static frame structure where a bus controller tells computers when to speak, but there are blocks in the protocol that do not even allow an LRU to speak at an incorrect time. The static structure of this protocol

implies that the designers did not expect to have to work with other manufacturers or frequently add new devices to the network. From a security lens, we would expect ASCB to be more difficult to inject an attack into, as the protocol was designed around that not being possible. But this is an assumption, an attacker might still be able to inject an attack. The critical detail is in how the protocol responds to the behavior that the designers very explicitly attempted to make impossible. Often illegal behavior results in an error state but not always. A defender needs to understand how the design of the system matches with the implementation of the protocol and any assumptions being made.

4.3.2 Protocol Information

Many protocols have a number of edge cases that are easily exploited [55], [11], and this thesis only adds to the list of edge cases with Chapters 3 and 6. As a result, many security systems require protocol-specific detection mechanisms to guard against a knowledgeable attacker. But protocol knowledge is a double edged sword. A defender can use the same protocol knowledge as the basis for their IDS design, either by adding new capabilities or by simplifying existing ones. The rest of this section covers protocol-specific ways of handling information and how an attacker or defender might abuse them.

4.3.2.1 Normal Communication

Normal communication is the pattern of traffic typically observed on a serial data bus network. The pattern of traffic is important for understanding when an attacker can inject messages onto the bus and for developing a baseline that an IDS can depend on. Broadly speaking, bus traffic operates on two paradigms: *arbitration* where at any point any device can speak and *structured* where a pattern of traffic is either hardcoded into the system or set by a higher entity, like a bus controller. In an arbitration paradigm the attacker can transmit at any time, as long as they can win the arbitration process. Generally it is safe to assume an attacker can synthetically win arbitration whenever they wish, as they can lie about any transmitted priority schemes. Structured bus traffic provides limited attack windows. An attacker needs to either be on the device that is asked to transmit data or wait for dead bus time to transmit their attack. Dead bus time is usually built into the protocol for user initiated and diagnostic messages or to give time for devices to process traffic. Let us imagine some features for a detection system based on these communication schemes.

If an attacker can transmit at any time, then an IDS needs to analyze every message for possible anomalies. But synthetically increasing the priority of an attacker message should be obvious, as it implies a new kind of message being transmitted onto the bus. A basic allowlist for known messages ensures that an attacker can only transmit normal messages. Now an attacker has to to queue their attack, but they can still eventually transmit. This is where our generic IDS resources come into play. A defender can analyze the bus traffic to see what IDS techniques best fit the system.

Structured traffic implies that any spoofed messages can only occur in a set time window. Because all spoofing attacks are adding extra messages to the bus, a spoofing focused detection system can limit their analysis to these time windows. Limiting when a detection system runs can be a helpful way to reduce the false positives produced by the system. The rest of the detection system can then be focused on detecting the malicious data inside of messages, without wasting processing time on predicting messages which occur in a set order.

Each protocol has nuance to how it transmits messages and how often it expects those messages. By building a detection system around the protocol we can more efficiently detect large swathes of attacks and invest detection resources into the harder problem of messages with malicious data. We will exemplify this in Chapter 6, where we use the MIL-STD-1553 structured scheme to detect any new messages initiated by an attacker, regardless of when those messages are transmitted.

4.3.2.2 Manufacturer Specific Information

Decisions in a protocol are sometimes handed from the designers to the manufacturers; suggestions may exist in the protocol but there is no guarantee that those are followed. Manufacturer specific information is extremely difficult to build a security system around, as it makes applying research from one system to another a complete gamble. Anything manufacturer specific should be flagged as requiring retraining from system to system. In the case of optional functionalities like mode codes in MIL-STD-1553, the system should be re-evaluated with those functionalities to ensure that they do not create any gaps in security.

4.3.2.3 Hardware

When designing detection systems it is easy to operate at the application layer and ignore the nuances of the physical medium. But hardware informs how the protocol is designed to work and can provide key insights. For example, in CAN the hardware is setup such that a CAN transceiver can easily override a 1 with a 0, but not the other

way around. From an attacker's perspective this shows that one kind of bit-flip attack is easy to perform. From a defender's perspective this demonstrates that the reverse, a 0 to 1 transition state, is inherently anomalous.

4.3.2.4 Addressing and Networking

Many protocols have a form of addressing to identify each LRU. Addressing is a staple of modern computer networks, but it is not guaranteed for serial data bus protocols. CAN avoids networking entirely, J1939 adds a network claiming process with arbitration, ARINC-429 has a convoluted system of unique device IDs which are not included in messages but are necessary to translate messages, MIL-STD-1553 has addresses and subaddresses as an approximation of an IP address and port, and ASCB has a different request and reply address for every LRU. Often the addressing process exposes a vulnerability [55], [73]. J1939 allows an attacker to claim an address, but many systems are designed without the ability to claim a new address. The result is a single message method for disabling important LRUs. MIL-STD-1553 has several messages to disable specific LRUs based on their address, as well as a bus controller architecture where any LRU can dynamically claim to be the bus controller.

4.3.2.5 Intrusion Prevention

Being able to prevent an attack from happening is the ultimate goal of many detection systems. The key questions to be asking are: what happens when a message collides with another message, and are there any built in ways of disabling LRUs. The first informs what happens when a prevention system tries to actively a cancel a message by invalidating it on the bus. Protocols like CAN replay the victim message after transmitting an error frame and eventually disable themselves. Other protocols like MIL-STD-1553 have no collision handling built into the protocol beyond setting an error flag on transceivers. The latter is perfect for a simple prevention system where mangling a message results in the message being dropped with no other consequences. If an attacker tries to emulate the IPSes behavior, then there is an error flag to let the defender know. The former requires a more complicated system, or at least the understanding that preventing an anomalous message is the same as creating a dead bus for an extended period of time. Built in methods for disabling an LRU should be alerted on by any functional IDS. Failing to do so would expose an obvious and deadly attack surface. However, a defender can use these built in methods to disable attackers. We expand on the caveats around protocol-based prevention schemes in Chapter 7.

4.4 Enforcing the Protocol

A trend in this chapter is an attacker abusing or outright ignoring the protocol to execute attacks. A simple security win for every protocol is a detection system that monitors the bus for any deviations from the protocol as written or any behavior that simply does not make sense for a static system. Using formal access control policies, such as the Clark and Wilson model [17], to enforce the protocol is outside the scope of this thesis as implementing an access control policy would require modifying legacy systems.

Take our earlier example of networking in J1939. The protocol technically allows for a device to claim a network address, but in what world is a new device going to claim the network address of the engine? The CAN protocol assumes that no malicious actor is on the bus and that nobody would try to claim the address of critical devices. However, a rogue address claim may not be from a malicious actor. It is possible to imagine a scenario where a legitimate device has networking issues or a mounting issue which causes it to disconnect and reconnect to the bus. This ambiguity leads into a broader conversation about errors on the bus. How does one distinguish the malicious actor from the erroneous computer? In this case, the nuances of the J1939 networking protocol would allow a defender to differentiate between an attacker and maintenance flaw easily. All the defender needs to do is monitor for a lower NAME than previously observed for a given LRU. But is the differentiation between an attack and an error necessary? Understanding which errors are from an attacker versus a legitimate error on the bus is always ideal and detection systems should always strive for this, but every new detection system has to factor in the concept of risk. Generally this risk calculus is simple. A new detection capability needs to detect enough attacks or provide enough value to justify any noise (false positives) that it could produce. Otherwise the IDS is a net negative to the system.

The risk calculus for error conditions is entirely dependent on how the defender perceives their detection system. Flagging on built in error codes inevitably leads to flagging on legitimate error states. On one hand, legitimate errors are errors and worth the IDS raising as a problem on the system. Additionally, these errors should be rare presuming the IDS is deployed on a functioning system. The more rare an error is, the more valuable it becomes as an indicator to a defender or system maintainer that something is truly wrong. On the other hand, errors are not malicious attacks, which is what is implied by an IDS. If errors are frequent, then the IDS produces a

large number of alerts that a system maintainer can do nothing about. We refer to these alerts as non-actionable alerts; an IDS should avoid producing these.

It is important to distinguish the issue of producing actionable alerts from the issue of the baseline system producing errors. If a baseline system is producing many errors but those errors are actionable, then alerting on them is acceptable. If the baseline errors are frequent but the defender cannot do anything to address them, then alerting on them is pointless if not actively harmful. In the case of rare errors or rare events, the defender may not be able to perform any immediate actions. However, the regularity of a serial data bus network implies that rare events are the result of external impacts on the system. These impacts may represent an attacker or a real emergency. For example, a defender might observe an error message built into MIL-STD-1553 which shuts down an LRU. A legitimately failing LRU would cause this message to trigger, but an attacker could have simply lied and sent the message. Regardless, we know this event does not naturally occur very often and so can safely alert on the event without the expectation of generating much noise.

We can see a positive and negative example of a protocol enforcement IDS in J1939. J1939 has a feature called CAN error frames which we discuss at length in Chapter 3. In that chapter we show that individual error frames may occur somewhat frequently and lack much meaning, but a critical mass of them may indicate an attack as serious error states are approached. In this scenario, the IDS acts as a filter on built in errors that determines when an error is worth alerting on. A more negative example can be seen when we try to build an IDS enforcing the J1939 specification. J1939 provides a reference document with an operational range for most of its 8192 possible engineering parameters. Alerting on any data outside of this operational range seems like an easy way of detecting an attacker making a mistake. A detection system should certainly catch the low-hanging fruit of an attacker obviously breaking the protocol, but the issue is that these operational ranges are suggestions for the manufacturer and not real guidelines. A system maintainer can do nothing about false alerts generated off of operational range because they do not have control over the code running across the devices on their vehicle.

From our examples we can derive two acceptable states for alerting on deviations from the protocol and protocol produced errors: when the error is actionable for the defender or when the error is indicative of an attack. We include rare errors in this as being indicative of an attack. By judging every potential error against these two criteria, a defender can ensure that common errors are ignored until they are proven to

be an attack, while serious or actionable errors are alerted on so they can be addressed quickly.

4.5 Threat Models

A threat model describes the goals and capabilities of an adversary. A defender typically uses a threat model to determine what attacks they should realistically be concerned about. Considering the lack of authentication on serial data buses, threat models usually boil down to how sophisticated an attacker is and what attack vector they are using.

4.5.1 Attack Sophistication

Attacker sophistication is the measure of how much more an attacker can do than a standard LRU. We define a standard LRU as a device which can transmit messages and must act in accordance with the protocol. From the defender's perspective, we can broadly split attacker sophistication into two categories: capability and adaptability. A capability is a new type of attack or method of interfacing with the bus that an attacker can use to evade detection. Capabilities might include outright ignoring the protocol or interrupting or modifying messages as they are transmitted. A defender is forced to develop new detection systems based around more advanced capabilities. Adaptability is the attacker slightly modifying their attack to avoid detection. Often this looks like an attacker replicating IDS technologies to find gaps, such as an attacker monitoring the timing between messages to find possible gaps in a timing based detection system or creating their own physical model of the target vehicle to identify where small changes in vehicle data can sneak through. Adaptability forces the defender to build their IDS around the assumption that an attacker is actively trying to circumvent it.

For the purposes of a threat model, we can envision attacker capabilities and adaptive skills as two checklists, filled with anything an attacker could theoretically do. While everything on those checklists is theoretically possible, a defender might conclude that they are not practically possible. The defender only needs to defend against the list of capabilities and adaptive skills that their intelligence informs them is possible. For example, the capability to modify messages as they go across the line requires custom hardware. A defender may believe that no attacker can feasibly connect custom hardware to their system and so they do not need to defend against it. The same idea holds for adaptability. An attacker would need the time, engineering

resources, and intelligence resources to build their attack around whatever detection system is deployed on the system.

4.5.2 Attack Vectors

Where the attack comes from impacts which detection technology a defender should choose. This is because the artifacts that an attack produces change based on how the attack is introduced to the bus. We identify three common attack vectors: implants, remote, and supply chain.

Implants are new devices connected to the bus, usually connected via some breach in physical security. In terms of executing attacks, this attack vector requires the least sophistication, as the attacker has complete control over the code running on the implant, and it is connected as if it were a new LRU. The consequence of being a new device on the bus is that every attack an implant sends is incidentally a spoofing attack. An implant is always pretending to be someone else when it transmits a message, but implants are not limited to spoofing attacks for sophisticated attackers. Implants are more configurable and customizable than our other attack vectors because the attacker is not battling with existing hardware and firmware. A sophisticated implant can include new capabilities, such as modifying or cancelling messages as they are transmitted across the bus or using customized firmware on a transceiver to conveniently ignore the restraints of the protocol. Implants are often not included in threat models, as defenders can be skeptical of someone physically connecting a new device to their system. However, red team testing of security devices often uses a new device to simplify testing and all implant detection strategies will work on a newly connected red team device.

The remote attack vector is when an attacker takes over an existing device without needing to connect to the bus or ever physically interact with the victim. Usually some level of transience is implied by the attack vector being a remote attacker. However, remote attacks which establish persistence on the victim system are possible (e.g., through a remote firmware upload [95]). At the lowest sophistication level, a remote attacker is only able to immediately transmit messages on the bus without any method for reading the bus and adjusting the attack. By taking over an existing device and transmitting the same messages as the victim, fingerprinting and cryptography based security systems are circumvented. For lower sophistication remote attacks, timing based systems may find a deviation in how frequently the attacker transmits messages. Higher sophistication levels allow an attacker to adjust their payload and timing based on the state of the system. Remote attacks inherently require some communication in

the RF spectrum that an attacker exploits to pivot onto a vehicle computer. If there are no communication devices and the vehicle is not connected to the internet then remote attacks can easily be written out of the threat model. A defender can also harden known remote connections using conventional security techniques to reduce the risk of remote attacks reaching the serial data bus network.

Supply chain as an attack vector is when an attacker maliciously interferes with a device as it is getting produced or delivered. It implies a near total control of the victim device, as well as a persistent foothold. The base benefit of a supply chain attack vector is the same as a remote attack vector, where transmitting as the victim enables the attacker to avoid artifacts of spoofing. As sophistication increases, the persistence of the supply chain attack vector allows an attacker to stealthily create problems on the vehicle. An example would be running an engine improperly such that it requires maintenance more frequently. A remote attacker can technically do the same thing as a supply chain attacker if it is able to establish persistence on its victim; however, supply chain attacks are more difficult and expensive to mitigate against due to the complexity of supply chain security.

In our analysis of these attack vectors we discuss the advantages for IDS circumvention, but it is worth remembering that any attacker can always elect to perform a spoofing attack even if they have corrupted a pivotal vehicle computer. Detection systems need to be built around intelligent actors but still catch the obvious mistakes.

4.6 Building the IDS

At this point in the chapter we have discussed threat models, the different resources and deployment strategies a defender can deploy, and how the protocol a system uses might guide the development of an intrusion detection system. By combining these concepts together a defender can develop a robust security analysis against any attacks in their threat model. This section discusses the layering of detection technologies, the prioritization of those technologies, and designing for specific end goals. In particular we focus on designing an IDS for either powering an intrusion prevention system, or for powering an incident response process.

4.6.1 Designing the Layers of an IDS

A security system is composed of multiple detection strategies all created to detect different attacks. By layering these detection strategies on top of each other in a deliberate way, a defender can ensure that an attacker is unable to transmit any

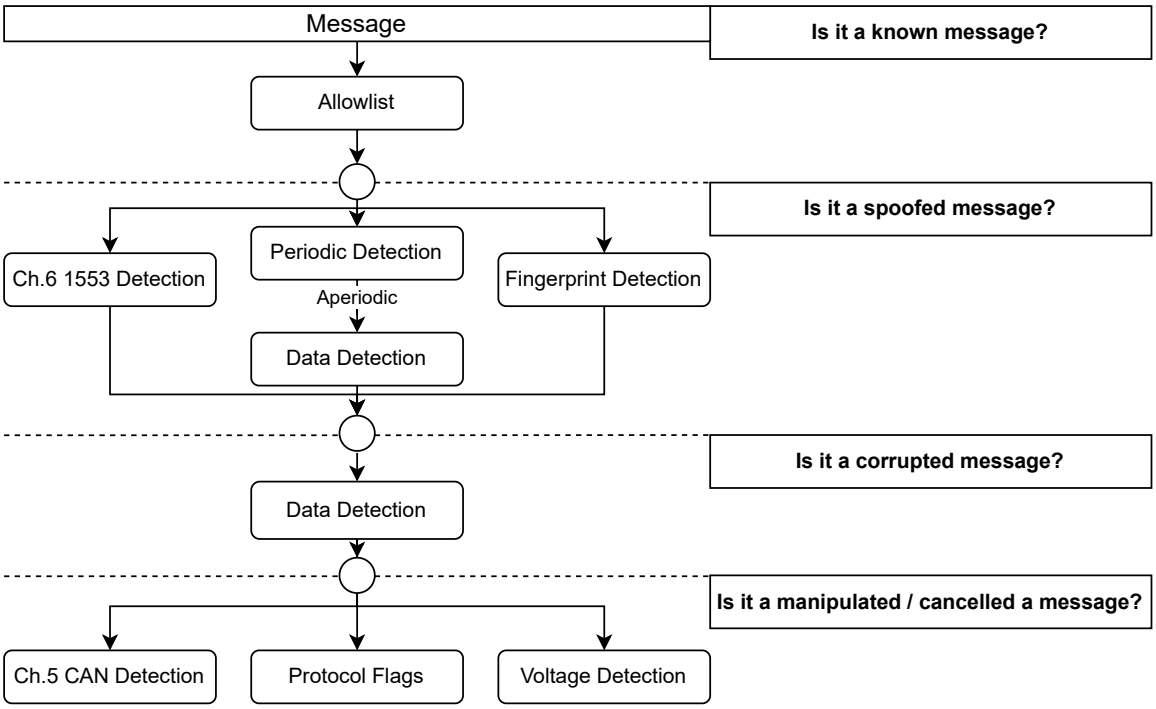


Figure 4.2: A flowchart depicting different detection systems suitable to detecting different attack types. Periodic detection represents a timing-based detection system. Protocol-specific detection systems are highlighted under spoofing detection for our MIL-STD-1553 solution, and under message manipulation detection for CAN.

message without being detected. The layering approach ensures the defender knows where security protections do and do not exist. Figure 4.2 is a flowchart which guides a defender through the thought process for detecting different attacks, and the technologies which detect those attacks. Each box in the flowchart is a different detection system, where different detection systems are grouped based on what kind of attack they are useful for detecting. Multiple detection systems under an attack is indicative of an or statement, where any of the approaches could be useful for detecting the attack.

On receiving an attack message, the easiest determination a defender can make is whether or not they have seen the message before. Serial data bus networks are regular and one would not expect new messages to appear on the bus after training on normal data, with the exception of some error messages. A defender can either pre-allow error messages so as not to alert the IDS, or alert on them. A simple allowlist technology allows a defender to easily detect an attacker making a mistake in regards to what type of message is sent. From there, the defender can either use (periodic) timing and data analysis, or voltage and data analysis to accomplish full coverage against spoofing and

corruption attacks. Each approach has unique advantages and disadvantages but the approaches are complementary to each other. From the allowlist step in the flowchart, all the defender knows is that the observed message is a message it has seen before. A fingerprinting solution can then inform the IDS if the fingerprint has been seen before. If the fingerprint matches and it is a known message, then the only remaining option is that an attacker has corrupted a legitimate device and the defender needs to check if the message contains malicious data. Data-based detection techniques can then ensure the message is legitimate, or at least not dangerous to the system. Alternatively, the defender can take a timing-based approach. If a message transmits periodically, then timing-based analysis can predict when the message should occur and alert on any deviations. If the message is not sent regularly, then data-based detection can be used to augment timing detection such that more messages are predictable. We expand on this technique in Chapter 5. However, if the message is being transmitted in the correct time interval, then the only remaining option is that the attacker has corrupted an existing device and data-based analysis is necessary to determine if the message is malicious.

Both of our approaches require data-based analysis to detect corrupted devices transmitting data and otherwise are attempting to identify spoofing attacks in different ways. Timing analysis and voltage fingerprinting are both designed around detecting new messages or devices on the bus, but with tradeoffs for each. Voltage fingerprinting requires hardware beyond a standard transceiver and continuous retraining, but it is incredibly good at detecting one device spoofing another, regardless of the type of message. This makes voltage fingerprinting ideal for detecting one-off error messages and diagnostic messages being transmitted by an illegitimate attacker. Additionally, voltage fingerprinting provides source attestation capabilities which are useful for intrusion prevention systems and incident response broadly, as well as passive implant detection.

In contrast, timing-based detection is more flexible, as it requires no custom hardware and can handle new devices being added to the system far more simply. In part because timing based training is easier to configure than the complexity of a voltage fingerprint. The difficulty with timing detection is that any messages with complicated timings require data analysis. A translation manual for serial data bus data can often be difficult to acquire, as standardized data fields are a rarity in the field. Additionally, timing-based detection systems need to be built very deliberately such that there is no condition where an attacker can sneak in a message if a legitimate

device happens to transmit more slowly than usual. Voltage fingerprinting does not have this complexity in its security model for spoofing attacks.

If we assume data analysis, an allowlist, and one of our spoofing detection options, then the only remaining attacks are protocol-specific attacks and ones based on cancelling or manipulating messages as they go across the line. It is difficult to build generalized detection systems which somehow cover every protocol-specific attack or technique. Often the best way to approach this is a simple rules system akin to snort [16], where the rules are built based on known attack papers and the protocol analysis techniques we describe earlier in this chapter. As for manipulating messages, the most straightforward detection approach is voltage analysis. An attacker trying to interfere with a message as it is transmitted will surely result in a fingerprint mismatch from an anomalous shift in voltage. The alternative is using the error process built into the protocol. Relying on built in errors should only be done if that error is not common on the system, or there is a method to differentiate common errors from attacker generated errors. That said, protocol-specific flags for detection have the same boon as timing detection compared to voltage fingerprinting in that protocol flags do not require custom hardware, as collecting error flags is built into the standard method for collecting messages. By implementing some combination of the aforementioned techniques, the defender can achieve a complete security analysis where the IDS detects any actions an attacker could possibly take. But the generic way is not necessarily best.

In Figure 4.2, we can see a single MIL-STD-1553 specific detection system is the equivalent to the rest of the spoofing detection solutions. Chapter 6 describes the 1553 detection system in detail, but in brief: the 1553 protocol has a specific method for transmitting messages, and our detection system can detect any attacker initiated messages with a simplified voltage fingerprinting system that only has to monitor the magnitude of the voltage from one LRU. This approach comes at the cost of assuming that one LRU, the bus controller, is trustworthy. But from that single assumption, a defender can dramatically simplify their detection system while still achieving the same level, if not a greater level, of security than the generic approach. The protocol guidance presented in this chapter can guide a defender to efficient detection systems which fully utilize the protocol.

4.6.2 Constructing the Layers of an IDS

This section so far has described 5 different detection systems acting in different combinations to achieve complementary layers of security. But defender resources

are limited and the full security system will surely not be written and deployed all at once. A defender must decide which systems to build first. Generally, the best start is reading protocol documentation. Any engineer trying to understand a new protocol will likely have to read the protocol documentation anyway to understand message formats and message translation. Making the security engineer one of the engineers working on ingesting a new protocol aids the engineer in developing protocol knowledge, while providing something constructive to do with that initial protocol knowledge. The goal of the defender while reading the protocol is twofold: develop rules based on protocol assumptions and errors, and look for any shortcuts that would simplify a detection system. Usually this simplification takes aspects of the generic flowchart and reduces them to a single detection system. The protocol analysis also establishes requirements for the IDS. If serious protocol-specific attacks are possible but the attacks are not rare enough to alert on out of hand, then the IDS needs a way of ensuring they always alert on the attack. For example, the MIL-STD-1553 system presented in Chapter 6 needs to be designed in such a way that it can always determine whether the protocol messages which turn off LRUs (mode codes) are legitimate or not. Otherwise, the obvious hole in security coverage makes the rest of the detection system close to useless.

In terms of the order of detection system development, it generally follows Figure 4.2. A basic allowlist is simple and ensures an attacker does not transmit a random source address or make up messages. Additionally, the allowlist serves as a safety net for protocol-specific attacks the defender may have missed. Typically the messages these attacks rely on do not appear in normal training and an allowlist ensures the IDS alerts on them. From there, the defender needs to prioritize detection systems based on what kind of attacks are most likely to happen. A rules-based detection system is good value for time if there are well known attacks or the defender has access to high quality threat intelligence to predict what kinds of attacks will happen. Even without threat intelligence resources, rules are a good strategy for manually covering some of the gaps left by undeveloped detection systems and for injecting domain specific knowledge into the IDS. Failing any specific attacks, the best option for detecting an attacker is focusing on spoofing detection. Spoofing detection ensures an attacker detects all attacks from an implant and any attacks where the attacker lands on one device and pretends to be another. Spoofing attack detection also enables the detection of an attacker pivoting from a remote foothold. Afterall, remotely accessible devices are not typically high priority LRUs like the engine or the transmission. An attacker must spoof non-remote LRUs to execute critical attacks. The same principal

is true for supply chain attacks, in that the attacker does not always land on the most critical LRU. Spoofing detects cheap corrupted auxiliary devices attempting to take over critical devices.

Timing-based intrusion detection is the simplest to implement in a short period of time and with limited data sets, as it requires no custom hardware and standard methods for collecting data include a timestamp. Timing-based detection also detects denial of service and flooding attacks implicitly. These are some of the more commonly proposed attacks on serial data bus networks [42]. However, if the end goal is a prevention system which needs to verify what device sent what message, then starting with voltage fingerprinting will be more beneficial in the long run. This only applies to prevention systems which need to communicate to an address, such as those that use error codes or do firmware uploads.

After developing one spoofing detection system, a defender has little reason to immediately do the other spoofing option, as they provide similar security guarantees. The only caveats are that a voltage-based detection system will likely want a denial of service attack and flooding attack detection mechanism, and a timing-based system will likely need some data analysis to cover rare messages with complicated timing characteristics. The next step is developing a mechanism for detecting attacks from corrupted LRUs. We assume all characteristics of a corrupted LRU's messages appear normal except for a malicious data field. Initial steps in this area may search for large or unusual jumps in data fields, such that attackers must make incremental changes to go undetected. There are two other tiers of data detection systems a defender can design. The first tier asks if the data makes sense given what is happening at the time. This is effective for detecting time bomb attacks or any attack that triggers while the vehicle is in operation. The second tier is simply asking if the data makes sense. Defending against a persistent adversary who is corrupting an LRU and is transmitting malicious data from moment one is much more difficult than detecting a shift in data.

Next a defender needs to decide the processing order for their detection systems. The general idea is that a defender should attempt to detect the attacks they think are likely first, as after classifying a message as an attack a defender can short-circuit the IDS process to save on processing time. The only caveat is that the defender needs to factor in the baseline complexity of different IDS techniques. At a high level our order is: allowlist, protocol flags, periodic detection, data detection, voltage detection, with periodic detection and voltage detection being swappable based on if source attestation capabilities are necessary. An allowlist and protocol flags are such simple

checks that they are worth doing on basically every message. Periodic detection can also be computationally simple, but the main benefit is that periodic detection detects three attacks (message flooding, silence on the bus, and spoofing). Data detection can be skipped if the attacker fully believes that no corrupted devices could exist on their system, but generally we do data detection before voltage detection because the security benefits of voltage detection overlap with periodic detection. Once a defender develops all of these detection systems, all that is left is ensuring that the performance of the overall system matches what it is intended for.

4.6.3 Designing for Prevention and Real Time Alerting

Preventing attacks or providing real time alerts to a user can be a serious engineering challenge. These goals require the IDS to finish processing either before the message is done transmitting for an IPS, or before the next message is done transmitting just to keep up with the bus. The timing restrictions complicate IDS development. In the case of an IPS, the IDS needs to read a message a bit at a time to find anomalies before the end of the message. Doing so requires custom hardware which the defender will need to program all the niceties of a standard transceiver into if they still want proper error handling and built in flags. The more of a packet that needs to be transmitted before the IDS can make a decision, the faster the overall system needs to be. This makes most detection schemes for corrupted devices poorly suited to becoming prevention systems. Data-based analysis creates extremely tight timing windows, especially if the data needs to be enriched. From this we can conclude that a defender may want to split detection schemes based on which are fast enough and accurate enough to act as a prevention system. We elaborate on how a defender can evaluate their detection system and how suitable it is to becoming a prevention system in Chapter 7.

4.6.4 Designing for Incident Response

If the goal of an intrusion detection system is not real time alerting, but instead a robust incident response process, then the defender needs to be able to answer three questions: what is the attack vector, what is the impact on the system, and what remediation is possible. A detection system for IPS or real time alerting does not necessarily need to provide contextual understanding, and so alerting for incident response needs to be designed differently. Voltage-based systems are a higher priority, as source attestation is a valuable capability for determining where an attack came from. Attestation can enable remediation. Data analysis and domain expertise is

necessary for understanding the impact on the system, but computationally identifying the period of time an analyst should look at is arguably a higher priority. The priority level depends on the scale of the response team and how quickly the response needs to happen. The smaller the team, and the faster the response is needed, the more important it is to programmatically identify areas of interest for the incident response team.

The incident responder needs to understand the attack while the prevention system only needs confidence that it is an attack. From a protocol-specific perspective, this means that based on how a protocol communicates errors and drops messages a detection system may understand that something bad happened but be unaware of exactly what happened and where in a message it occurred. Remedying this requires either custom collection hardware to ensure messages with errors are not dropped or a parallel raw voltage collection which can be analyzed after the fact.

An IDS designed for incident response does not need to act quickly. This means that an IDS can detect anomalies across large collections of messages rather than evaluating a single message at a time. One example of this is having timing-based solutions examine the statistical density of messages over a trip. Additionally, more computationally expensive options such as deep neural networks and recurrent neural networks become feasible when done in post.

4.7 Conclusion

Using the methodology described in this chapter, a defender can develop more efficient and comprehensive detection systems. Our methodology guides a defender from selecting how their potential IDS is deployed, all the way to the building and layering their detection systems. In between the methodology identifies what resources are available based on their deployment, enables an exploratory analysis of their system's protocol, and defines the defender's threat model. These steps in our methodology allow a defender to simplify their detection system and preemptively discover protocol-specific attacks. Simplifying an IDS reduces the barrier for implementing it and increases accuracy and attack coverage. Building the IDS with knowledge of protocol attacks ensures a defender cannot trivially bypass the detection system as they might with a generic approach. The actual building step of our methodology further reduces the barrier for adding security to legacy vehicles by helping resource-limited defenders maximize their attack coverage. Over the next two chapters we apply our methodology

to a CAN system and a MIL-STD-1553 system. These chapters demonstrate how our approach to IDS development changes from protocol to protocol.

Chapter 5

Detecting CAN Attacks on J1939 and NMEA 2000 Networks

In this chapter we show how knowledge of the CAN protocol can simplify pieces of our generic IDS development methodology. Specifically, we are looking at industrial CAN. Industrial protocols, such as J1939 and NMEA 2000, take the small attack vector of consumer car hacking and expand it across millions of trucks, ships, and even military systems by standardizing communication. Now, not only is reverse engineering not necessary, an attack that controls the engine on one make and model of truck also can control the engine of another truck with completely different hardware. This means that any remote attack payload no longer only works on one vehicle, but entire fleets of heterogeneous industrial and military vehicles. The value of these assets is far higher than their consumer automobile counterparts [21], [26], while simultaneously being an easier target through their remote attack vectors, and standardized data fields.

An example of a remote attack vector is the Electronic Data Logger (EDL). This device is required in US trucks, often plugs directly to the OBD-II port, and then sends data to a fleet management system [96]. A keen observer might note that an EDL is extremely similar to the limited, highly targeted, physical implant style attacks used by car hacking researchers. But now the researcher's implant is mandatory and installed across fleets. An EDL is one example among many. The desire for real time access to bus data is growing and has the potential to be a scalable attack vector [91], [66], [79], [95].

The question is then how to secure the CAN bus in a way that supports the decades of systems we already have. One method is adding authentication, but cryptographic mechanisms would impact bus performance and limit possible bus

configurations [58], [70], [33], [19]. This makes cryptography impractical for existing systems, especially with the bandwidth overhead from J1939 running at 250kb/s.

The alternative to authentication is adding an intrusion detection system (IDS) which detects attacks as they go across the bus. The proposed solutions in this area range from timing analysis [12], [86], [101] to signal fingerprinting [14], [76], [88] to physical invariance models based on the data from the bus [72], [98]. While these solutions work for some types of messages, they have serious gaps.

For example, let us discuss a message in J1939 which enables cruise control and sets the cruise control speed. Our electronic data logger attacker could transmit a single cruise control message which says that the brake is off, the cruise control is enabled, and that the cruise control speed is 1mph. This causes the vehicle to slow down dramatically, which could create dangerous situations in typical driving environments. In our datasets, the cruise control message has three different transmission intervals centered around 35ms, 100ms, and 150ms. Previous timing, data, and ECU fingerprinting techniques do not cover this attack. Prior timing-based work fails to account for these three different timing intervals, meaning the timing-based work either generates false positives or ignores the message entirely. Data-based work relies on detecting changes that violate the physical invariance of the system, but cruise control controls the vehicle using normal mechanisms. Finally, ECU fingerprinting techniques fail to account for corrupted legitimate devices and are prone to high false positives from varied external environments. The fundamental problem is that these techniques do not cover all message and attack combinations when used in isolation. Our IDS, CANDID, uses a combination of timing, data, and voltage-based techniques in a novel way to ensure we apply some security guarantee to every message, not just ones with regular transmission intervals from physical implants.

The first step is detecting our EDL sending our cruise control attack. CANDID detects the EDL, or any ECU, spoofing the messages of another ECU by calculating the time interval between messages. This interval-based detection is improved by correlating data to the timing of messages. Data correlation allows CANDID to select the appropriate timing for the state of the system instead of assuming fixed timings. An EDL does not usually transmit messages. This means that we can detect any message the EDL transmits, as it is always spoofing another ECU. All normally transmitting attackers can no longer spoof another ECU's data fields. CANDID leverages this guarantee by drawing dependencies between data fields, such that even if an attacker takes over a critical ECU the data is restricted to what could be expected given the data of every trusted ECU. The final consideration is how to detect an attacker which

has control over the voltage of the bus, meaning they could arbitrarily change any and all bits on the bus. Rather than a full voltage fingerprinting mechanism which is prone to false positives and requires constant retraining, CANDID detects these bit flips through CAN error frames and voltage transitions which never occur in benign traffic.

We summarize our contributions as follows:

- An integrated timing and data solution which more accurately predicts message timing based on the state of the system and dependencies between data. Using these features, we are able to detect an adversary spoofing trusted ECUs and limit attackers in control of legitimate ECUs.
- A practical detection mechanism for bit manipulation attacks which does not rely on continuous training. Our improvements on bit manipulation attacks to silently force ECUs into the bus-off state underscore the importance of this detection mechanism.
- A proof of concept implementation of CANDID. The implementation is tested on multiple vehicles, as well as adversarial data generated on a testbench.

5.1 System and Adversary Model

In this section we define our system and adversarial model. The system model defines our CAN bus infrastructure and how CANDID is connected. The adversary model defines the capabilities of the adversary. The system and adversary model differ from the ones we depict in Chapter 3 as the system model now includes an amperage monitor on our IDS's pullup resistors and J1939 specific assumptions, and the adversary model is now defined in terms of spoofing, masquerade, and manipulation attacks. Manipulation attacks are the same as bit-flip attacks.

5.1.1 System Model

Our system model is depicted in Figure 5.1. CANDID is connected to the two wires of our CAN bus, allowing it to monitor all messages transmitted on the bus. CANDID is able to interpret any observed data using the J1939 standard. All CAN packets and alerts are stored and compressed on-disk for future analysis. CANDID is secure from wireless threats, and can not be removed from the system. Our IDS is also capable of monitoring the amperage going over the pullup resistors depicted in our system

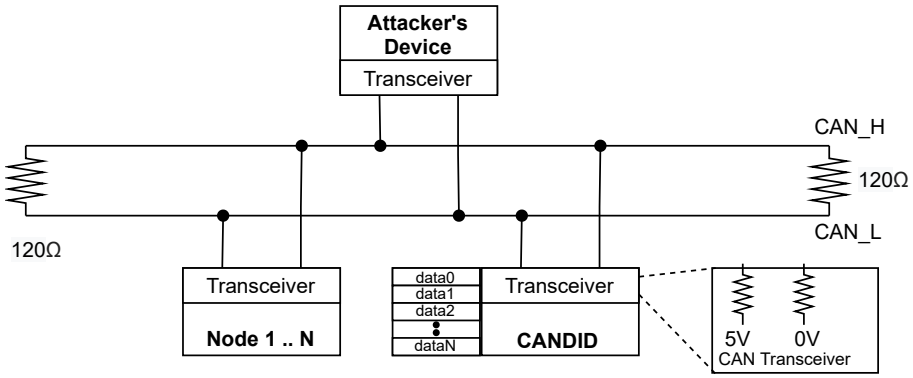


Figure 5.1: The system model for a CAN system. Each node represents an ECU with a CAN transceiver, though attackers may have custom hardware.

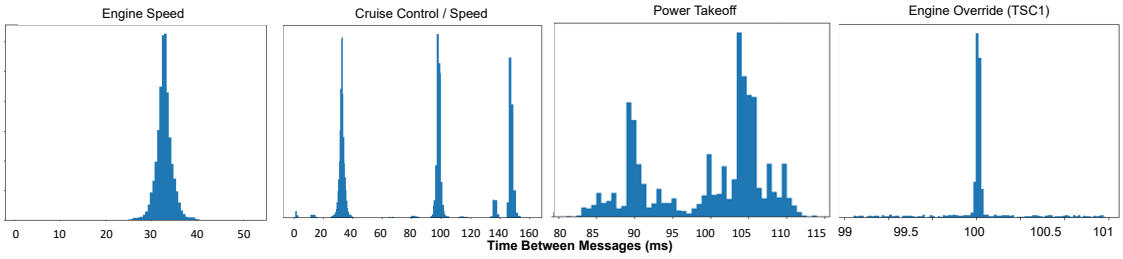


Figure 5.2: The interval data from four J1939 messages. The engine speed message and engine override message are easy to predict with a single distribution, the rest require an understanding of when messages are transmitted based on the data going over the bus.

model. Our system requires a brief training period, we assume the attacker is not active during this period.

CAN transmits data at regular interval, though messages can have multiple transmission rates or a wide interval. We can see examples of this in Figure 5.2, which graphs the observed message intervals for four message types from a single data set. The first and fourth messages are consistent and would be considered periodic by existing work [12], [86]. The other two have varying transmission rates and may be considered aperiodic by the same work. Notably, J1939 transmits sensor data at a fixed rate. This is not necessarily a single fixed rate, but J1939 will reliably transmit messages rather than transmitting sporadically whenever a message value is updated.

We define the state of the system to be CANDID's estimation of how the system is operating. For each message CANDID updates its internal variables and tracks how those change over time. A full explanation for this process of tracking state is in Section 5.2.

5.1.2 Adversary Model

We define the goal of the adversary as changing the state of the system. The state is changed if a dependent data field is modified as a result of the attack. An attack is successful if the state is changed without CANDID detecting it.

An adversary has the power to manipulate voltage over the bus directly, send and receive messages, and modify their attacks based on the state of the system. In a realistic environment, the capabilities vary between attack vectors with remote attackers being limited in their adaptability and corrupted ECUs having the most control over their victim. However, we deliberately assume the strongest adversary model possible regardless of attack vector and base our security analysis on how that attacker affects the bus.

At a high level an attacker has three options for transmitting an attack across the bus. They can transmit a new message, increasing the overall bus utilization, corrupt an existing device and transmit malicious data as the victim device (effectively replacing a legitimate message with a malicious one), or arbitrarily manipulate the voltage of the bus. In this chapter we define these attacks as follows: **spoofing**, where an attacker transmits a new message for an ECU they do not control, **masquerading**, where an attacker controls an ECU and transmit the messages it would normally transmit with different data, and **manipulation**, where an attacker controls the voltage to modify individual bits.

This categorization does not limit possible attacks, instead it provides nuance for our detection system. For example, a corrupted Engine Control Module (ECM) is able to perform a masquerade attack for engine messages, but if it attempts the same on a trusted ECU then that trusted ECU is still transmitting and our attacker is increasing bus utilization. The increased bus utilization makes it a spoofing attack. Through our attack categorization, we can define security guarantees for our attacker regardless of how they connect to the bus.

5.2 CANDID's Design

This section outlines the design considerations for CANDID, and expands on the chosen designs in the subsequent subsections. The core design is motivated by the three broad attack categories laid out in the adversary model: spoofing, masquerading, and manipulation. Previous work clearly indicates timing is an effective mechanism for detecting spoofing; an attacker transmits a message on the bus, that message is regularly transmitted by a trusted ECU, and the result is an anomalous timing

interval [86]. When applied to every ECU on the bus, this technique detects any attack vector pretending to be a trusted ECU. However, if a trusted ECU transmits with multiple timing intervals then an attacker can easily circumvent the detection system. This indicates a CAN message ID is not enough to detect attackers for all messages. We conclude that CANDID must be able to use the data it sees on the bus to understand messages with multiple timing intervals.

The next challenge is masquerade attacks, where a corrupted device transmits fake data at the appropriate time. Fake data covers a wide range, from increasing the Engine speed to 8000 Revolutions Per Minute (RPM) to minor tweaks creating small inefficiencies. In some ways, always detecting those minor tweaks is implausible with physical systems. Minor fluctuations happen constantly and calculating the impact of every data field on the system is too computationally expensive for a lightweight IDS to do in near real time. Instead, we take the approach of constraining attackers and limiting damage. Our spoofing detection covering all messages ensures that an attacker cannot transmit the IDs, or the data, of another ECU. From this foundation, CANDID can safely examine the data going across the bus and calculate correlations between the data fields. These correlations effectively create a dependency tree. If the data from one dependent ECU does not match the others then CANDID can alert. This limits the scope of an attacker's fake data, without attempting to model the entire system.

Manipulation attacks bypass both of our intended detection mechanisms, as an attacker could overwrite every bit of the bus in a way that changes no timing characteristics and maintains a consistent narrative in the data field. The first option is a voltage based intrusion detection system which fingerprints every system on the bus. However, these solutions can be unreliable in real world environments without retraining, and have the same issues with masquerade attacks as our easier to train timing solution. For CANDID we choose a more targeted approach to manipulation attack detection. To perform a bit flip, i.e., change a logical 0 to a 1, a higher than normal amount of current will flow through the termination resistors in the connected ECUs, which can be detected by CANDID. To avoid complicating the description of our timing and data analysis, we describe how manipulation attacks are executed and detected in Section 5.3. CANDID's unique approach of integrating and improving upon these three techniques ensures we more accurately detect more advanced attackers, regardless of if they are trying to avoid our IDS and know how it functions.

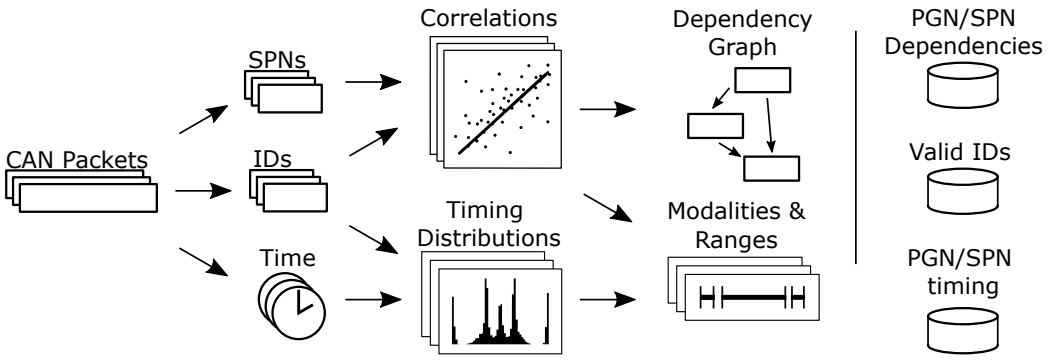


Figure 5.3: A high level representation of CANDID's training phase. Correlations are drawn between data and IDs, resulting in a dependency graph based on which data fields change in response to a given data parameter changing. Simultaneously a timing distribution of intervals for each ID is calculated and combined with the correlations to identify the data fields which result in a change in timing. Our training is stored as PGN/SPN dependencies, a list of valid IDs, and PGN/SPN timing.

5.2.1 CANDID Training

For CANDID to function it must train on a clean vehicle dataset. By training we mean fine tuning CANDID based on the parameters of the vehicle, not a machine learning based solution as seen in recent literature [68]. Without training our solutions risk being too generic. This would allow attackers to sneak by if an ECU transmits with a slight delay or if an ECU has more variance in its data field.

CANDID has 2 main components: determining the dependencies between data parameters and the expected timing distributions for each ID. We depict these components in Figure 5.3. In this subsection we describe how CANDID determines timing distributions, how those distributions are affected by the data going over the bus, and the correlation between parameters. CANDID trains off of a continuous dataset with times calculated based on arrival time in microseconds from the perspective of the process communicating with the transceiver. This process need to be redone for each vehicle.

For high quality training CANDID requires CAN data from the vehicle in normal operations (battery on, idle, driving in various conditions) and importantly the transition between states. Any new types of messages or vehicle states will be seen as anomalous. This is generally fine for rare messages, as we expect an operator would be fine with being alerted to a legitimate error state, but any core vehicle functionality needs to be included in the training set to avoid false positives.

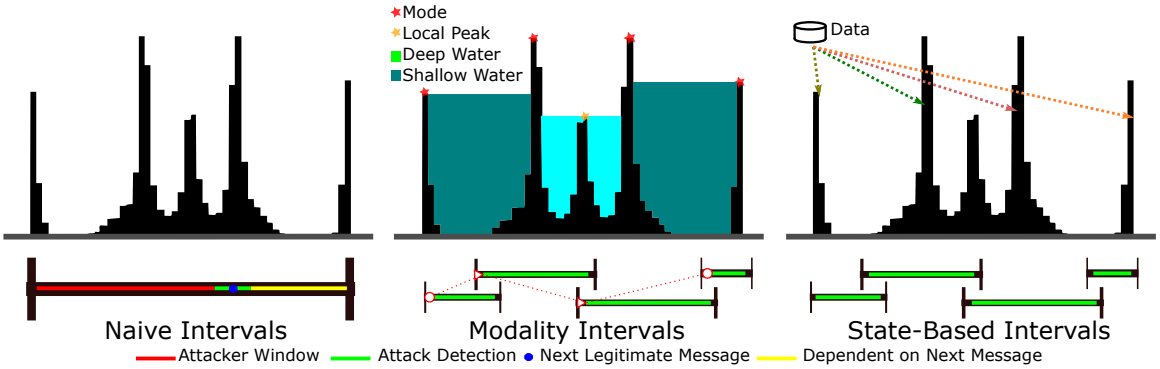


Figure 5.4: The differences between naive intervals [86] derived from the max and min of all the interval data, modality intervals derived from a low-land detection algorithm [1], and CANDID's state-based intervals which use vehicle data to restrict an attacker from jumping between intervals and to solidify the upper and lower bounds of each interval. Naive intervals where the width of the interval is greater than the lowest bound allow an attacker to insert attacks such that a later legitimate message will result in no timing anomalies. Instead the IDS will observe two legitimate messages unless the message is completely outside the interval.

5.2.1.1 Timing Interval Training

The first step is calculating the timing distributions, or intervals, as shown in Figure 5.3. CANDID relies heavily on the idea of fixed timing intervals. At face value, an interval sounds overly simplistic and prone to false positives. An easy to imagine alternative is viewing the distribution of timing intervals and determining if a number of messages seen over some time period matches that distribution. The problem is that many CAN attacks are a single CAN packet, making this alternative unlikely to detect the attack. Still, timing is an ideal basis for security as all attacks, outside of manipulation attacks, affect the timing of messages. A timing interval, enhanced by knowledge of the data field, allows the defender to detect attacks without predicting the exact order of messages.

CANDID uses the ID field of each packet to group messages and calculate the interval between each message with the same ID, resulting in a list of intervals and timestamps for each ID. The list serves two purposes. Firstly, it gives us a list of valid IDs. Any IDs not seen during training are assumed to be anomalous. The second purpose of the list of ID intervals is determining when a message is going to be transmitted. We do this by taking the maximum and minimum for each list of intervals and defining them as the upperbound and lowerbound respectively. Of course, determining when a message should arrive is not quite so simple. If the space between the bounds is too large, then an attacker may be able to sneak in. If they are

too small, then the model may be prone to false positives. CANDID's timing-based training must ensure that for a given interval an attacker's message must result in 2 or more messages within that interval, or one legitimate message within an interval and the attacker's message outside the interval. Having this interval ensures that an attacker cannot inject a new message onto the bus without being detected.

Let us use Figure 5.4 to demonstrate how existing work draws intervals and how CANDID draws intervals. The naive interval depicted in our figure is used by Song [86]. This interval takes the minimum observed interval from a dataset and uses that as a lowerbound. Any messages sent faster than that lowerbound are alerted on. In our figure we add an upperbound as well as it improves the security of the system and Song is not focused on late messages as an attack. The problem lies in the width of the interval. Let us consider the example of a message that occurs between 10 and 50 milliseconds. If the legitimate device is transmitting closer to 50ms, then the attacker can transmit at 10ms, 20ms, 30ms, and 40ms undetected. The security system restarts the interval clock on each new message, meaning that when the legitimate message transmits the security system will assume an allowable 10ms interval. We can solve for this issue by defining a bound on how wide intervals can be. We define this width such that the lowerbound is at least as high as the width of the interval. Bounding based on this width is an improvement on existing timing based work, which assumes messages are periodic and focuses on messages occurring quickly instead of outside of a bounded time range [86]. But not all messages will fit within these bounds, as some messages might actually transmit between 10 and 50ms. However, messages are roughly periodic. They transmit with averages centered around 10 and 50ms not randomly within a wide range of timeframes. An ECU transmits more or less frequently based on the needs of the system. Our earlier timing figure, Figure 5.2, clearly demonstrates that messages transmit around modes. We define a mode or modality as the grouping of timing intervals around different times that occur most frequently. The next step is identifying those modalities.

The middle image is a depiction of how CANDID identifies the different modes of an interval dataset. CANDID identifies modalities by transforming the set of intervals for a given message into a Harrell-Davis quantile estimator [37]. This quantile estimator is a way of estimating the shape of the interval dataset for each message ID. We can then apply a low-land detection algorithm [1] to that quantile estimator data. In brief, the algorithm uses the quantile estimator to identify local maxima in our dataset, then metaphorically fills the valleys of our dataset with water to identify any peaks. The low-land detection algorithm then narrows these peaks down to the modes

of the dataset based on how much water filled the different valleys. CAN interval datasets tend to be noisy, with the risk of modes being close together. The low-land detection algorithm allows us to identify the modes of n-modal distributions and by extension which messages fit within each of those modes.

This gives CANDID n groups of messages, where n is the number of modalities for when a given message occurs. Now CANDID knows that a message can appear at roughly any of these times, but CANDID still needs to identify two things. CANDID does not know how to identify what message is going to be sent at what mode and CANDID has no knowledge of how to draw a timing interval around this mode such that it captures the appropriate messages without being too encompassing. The middle image of Figure 5.4 demonstrates the problem with stopping at this point. While we can guess at the lowerbounds and upperbounds of an interval based on our bounding critera, we do not know the real edges of those intervals. This allows an attacker to jump from interval to interval without being detected, as we lack any way of knowing which of two overlapping intervals a message is in. Fixing this problem requires a contextual understanding of why a message changes its transmission rate. This is where the correlation block of Figure 5.3 becomes relevant and combines with timing distributions to form modalities.

5.2.1.2 Modality Based Interval Training

At this point in training CANDID has lists of messages with timing intervals at each modality. Because the mode is by definition a common state, CANDID can search for when a message goes from transmitting in one modality to another. Our IDS takes the list of messages that occurred in the timeframe between the modality switch, then grabs the last time CANDID saw that message before our chosen modality switch and the most recent occurrence after our chosen modality switch for each of those messages. Now CANDID has a list of every message that occurred between a given ID's modality switch, as well as the most recent occurrence of each message in that list before and after the modality switch. CANDID then enriches the data fields of those messages to identify what data fields are changing as the modality changes. To pare down the list of potential messages, CANDID starts with smallest time between each modality switch for a given message and ignores any data fields not included in that switch when training on future observations of that switch. We refer to the data fields that influence the timing characteristics of a message as state parameters. We short-circuit the processing to find these state parameters by first examining discrete parameters, as these often reflect the state of the vehicle. If that parameter

changes as the modality changes, then we mark it as a state parameter. More than one parameter may change for each modality, which can make it difficult to establish a causal relationship, but CANDID is indifferent as long as the change is consistent. The more modality switches in the training dataset, the more CANDID can whittle down the possible list of state parameters. CANDID stores these state parameters for the evaluation process.

Now that CANDID has the state parameters for each modality change, it needs to learn the timing interval it applies to. Doing this brings us to the third image of Figure 5.4 and our modality training process. CANDID iterates through the training dataset for each message with state parameters, identifying which state the system is in when the message appears and lowering the lowerbound or raising the upperbound accordingly. After iterating through the whole dataset, each modality for each message with multiple transmission rates now has a base lowerbound and upperbound. From there, if the modalities are far apart then we expand the interval to its maximum to be more forgiving of variations, favoring a higher upperbound as attacks are more likely to be faster messages than slower messages. If the modalities are close together and have the same state parameter and ID, then they are merged together. With modalities close together, they can often have overlapping intervals as a particularly fast message from one interval lines up with a particularly slow message from another. This does not present a security problem for our interval based analysis because we assume that a message does not rapidly flip states. Rapidly being defined as the upperbound of the original transmission interval. Limiting how often a message can flip states ensures that an attacker cannot bypass our interval-based guarantee by pretending the system is in another state and transmitting at that state's interval, as they could do with Figure 5.4's modality intervals. After the modality training CANDID has two types of trained intervals. CANDID has the base interval training for single transmission interval messages. For more complicated n-modal messages, CANDID has a list of state parameters to determine which of n timing intervals it applies to a given message.

The observed upperbound and lowerbound may be more rigid than necessary. CANDID expands intervals such that the lowerbound is only just greater than the width of the interval. This expansion favors increasing the upperbound. A message is more likely to be delayed as the bus tends to get busier than the baseline. The more comprehensive the dataset, the less this expansion matters for reducing false positives. Generally some expansion is desired to disentangle abnormal bus loads from attacks. If there is no viable boundary, then CANDID defaults to only using a lowerbound.

Algorithm 2 CANDID Data-Dependency Training

```
1: //transform dataset into timestamped list of SPNs
2: dataset ← readTrace()
3: spnData ← getListOfEachSPNValues()
4: //calculate correlations for each SPN
5: for each spnList ∈ ∫pnData do
6:    for each tSpnList ∈ ∫pnData do
7:       //Loop through lists and duplicate the slower message
8:       extendSmallerList(&spnList, &tSpnList)
9:       listOfCorr ← calculateSpearmanCorrelation()
10:      //Save correlation if strongly correlated
11:      if abs(avg(listOfCorr)) >= 0.85 then corrSpns[spnList[ID]] ←
   corr
12:      //Calculate CUSUM threshold based on max diff from avg
13:      corrSpns[cusumThreshold] ← maxDiff(listOfCorr)
```

This lowerbound ensures that even if a message has a wide timing interval CANDID will still detect an attacker transmitting that message too frequently.

The ramification of our modality training is that an attacker cannot spoof any messages regardless of the ID. The inability to spoof means that an attacker has to corrupt the device they want to send messages as. This need to corrupt is because traditionally adhoc messages, such as the power take off or the engine override (TSC1) messages highlighted in J1939 hacking work [8], are typically transmitted at predictable timing intervals. We see both of these messages in Figure 5.2 as images three and four. We expect that in datasets where this message is only transmitted in emergency situations CANDID would train on data that indicates that situation. Let us use an engine diagnostic light as an example aperiodic message. Our data-based training allows CANDID to see the data which results in that light turning on and alerts if that data is not true when the engine light message is transmitted. If the attacker is not directly controlling the engine, but it is instead controlling the gauge cluster displaying the light, then the attacker has no way of setting the engine conditions that result in that diagnostic light without being detected by the timing anomaly from a spoofed engine message.

5.2.1.3 Data-Dependency Training

The final phase of CANDID's training is identifying how a change in one parameter affects a future parameter. This is the last block of Figure 5.3, the dependency graph. We include this because if an attacker can corrupt an ECU, then CANDID needs a

mechanism for detecting when they are sending malicious data. We identify deviations in the data by calculating the correlations between parameters. After all, vehicle operations are communicated on the CAN bus because ECUs rely on this data to function. If the data ECUs output does not match with the flow of data going over the bus, then there is an anomaly. Algorithm 2 presents the algorithm behind this data-dependency training. As our data relationships are often non-linear, we use Spearman correlation to calculate the relationship between parameters. To do this we take our training dataset and pair each data permutation together, duplicating and extending the slower message to pair with each faster message. Duplicating the slower message allows us to determine how much a faster message is concerned with older data as the system continues to run. To reduce our risk of producing false positives we only consider pairs with a correlation of an absolute value greater than 0.85, though this value could be changed depending on the desired sensitivity and computational restrictions. The correlation values for these messages are stored by our training process so we can compare the value in our evaluation phase.

The last step to ensuring correlations are effective is accounting for naturally fluctuating scalar values. The RPM of an idling engine is an inherently unstable data point, yet an idling engine will only fluctuate so far, as the physical component attempts to maintain a steady value. Parameters correlated to engine speed are fluctuating along with engine speed, but these messages are not always transmitted at the same speed. We need to ensure an attacker cannot slowly change engine speed such that each individual step looks like normal fluctuations without the fluctuation averaging back to normal. Our solution is to compute the difference between the correlated and observed value, where we assume an expected difference of zero for dependent messages transmitting more frequently than their correlated message. By applying the CUSUM algorithm [65] to this data, we can alert on both small errors propagating across a large period of time, and large variations from the correlated value. The alert threshold is determined by training on normal data and taking the maximum observed error in either direction. The result is no matter how slowly or quickly an attacker changes the data, we will alert before the system is seriously impacted.

Let us examine the ramifications of this approach for some engine messages. If an attacker corrupts the engine, then they effectively control most of the vehicle. Many of the messages an engine transmits are dependent upon inputs from the rest of the system, such as the transmission. It follows that if an attacker tries to increase the engine speed without sending the transmission message engine speed is dependent

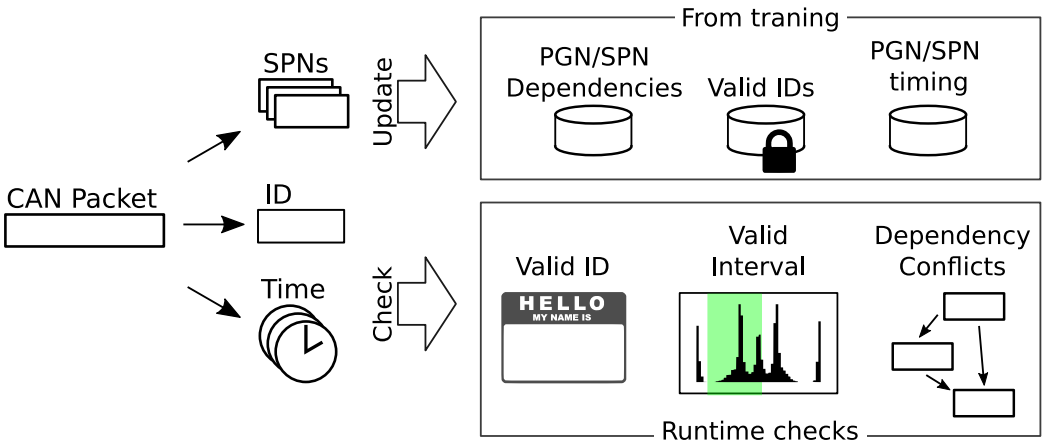


Figure 5.5: The high level overview of CANDID's Evaluation Phase. On receiving a CAN packet CANDID extracts the ID, time, and parameters. Training is checked to determine if the message has a valid ID, if the interval between the observed ID and the last occurrence matches what CANDID expects from our PGN/SPN timing training given the state of the system, and if the changing data values are correlated according to CANDID's PGN/SPN dependencies training. Using the observed data we update what state our training believes we are in. Valid IDs does not change as the system operates.

on, then CANDID will alert on the deviation between the expected correlation and what the attacker sent. Our timing guarantees ensure that a corrupted engine control module could not then spoof a transmission message without being detected. That said, there are high level messages which take control of systems, like the engine override message mentioned in the previous subsection. However, the engine sends that message and it sends it regularly. Because of our improved timing guarantees, we can predict when that message will occur and the attacker has to take over the engine specifically to send an engine override message. The impact of the data correlation security benefits are best demonstrated with the plethora of other ECUs on a CAN bus. It is important to ensure that any cheap ECU cannot start lying about its data to an extent where it has an impact on the vehicle. This forces the attacker to stick closely to what the bus is saying for most ECUs. For the rare ECUs with override messages, the attacker now needs to corrupt that ECU specifically to perform their desired attack.

5.2.2 Evaluation Phase

The evaluation phase determines how CANDID functions on each incoming message. A high level view of what the evaluation phase considers can be seen in Figure 5.5.

Essentially the evaluation phase checks that the observed ID is valid, that the timing interval makes sense given the current state, and for any deviations in the trained data dependencies. In terms of timing, CANDID calculates the interval between the observed ID and the last two times it was seen. The appropriate bounds are then selected by querying the last value for the trained state parameter. If the interval is within the lower and upper bounds for the 2nd from last message or is not within those bounds for the last message then CANDID alerts. We examine the 2nd from last message to ensure that an attacker cannot start a new message to reset a timing interval. CANDID also alerts if the state parameter has changed for each of these messages. This is indicative of an attacker changing the state parameter to change the expected timing bounds, followed by the legitimate ECU transmitting in the original bounds. In terms of data, the expected correlation for data parameters with strong correlations is queried from our training database for each parameter. If the percent change between an observed parameter and the percent change in the correlated parameter deviates beyond the trained CUSUM threshold then CANDID alerts. This percent change is then stored in memory. In summary, CANDID stores a tuple containing the timestamp and state parameter for the last two messages from each ID, as well as the percent change of each parameter for each ID. All other necessary data is pulled from CANDID's training.

Taken together, CANDID is a robust intrusion detection system which combines timing and data analysis to detect spoofing attacks and masquerade attacks where a corrupted ECU is transmitting messages correlated to trusted ECUs. This leaves the manipulation attack, which manipulates arbitrary bits as they are transmitted on the bus. In the next section, we demonstrate how simple it is to perform this attack and the CAN features that simplify detecting it.

5.3 Manipulation Attacks

We describe how to perform and detect manipulation attacks in a separate section as it represents a much more advanced threat model than what CANDID is addressing and consequently a detection system that requires more than a standard CAN transceiver. If we look back at our methodology in Chapter 4, this section is representing a different level of complexity as it has no overlap with spoofing or masquerading attacks. Manipulation attacks can change the contents of a message without changing the timing metadata. Consequently, if manipulation attacks are not addressed then spoofing attacks are difficult to detect. Additionally, repeated manipulation attacks

can lead to a stealthy bus-off attack [11] scenario as shown in Chapter 3. This disables an ECU within a single message without generating a single error frame or dropping the message.

This section describes how a manipulation attack is done, the possible consequences of them, and how our IDS detects it.

5.3.1 CAN Mechanisms Supporting Detection

Before we describe how manipulation attacks work, we must reiterate a few CAN features. These features simplify our detection mechanism, such that we only have to detect 0 to 1 manipulation attacks to cover all manipulation attacks.

The CAN bus consists of two wires labeled CAN-L and CAN-H. The differential voltage between CAN-L and CAN-H determines the logical output. A '1' is the default and referred to as the recessive state, with a differential voltage of 0V. This recessive state means that no ECU is actively driving the CAN bus. The passive state is a relatively constant voltage on the CAN-L and CAN-H lines for each CAN bus. A 0 is the dominant state with a differential voltage of approximately 2V. If any ECU is transmitting a 0 signal it overrides the recessive state and transmits a 0 across the wire. An attacker overriding a recessive bit with a 0 after the arbitration field generates a CAN error.

For the purposes of this chapter, we are concerned with two forms of CAN errors which are implemented in all ECUs. We use these errors to force the attacker into a scenario where they must manipulate a dominant bit to a recessive bit. Our detection system is based around detecting that specific bit transition.

A Bit Monitoring Error occurs when an ECU transmits one bit and sees a different value on the bus. The only time bit monitoring errors do not apply is when an ECU transmits a logical 1 during the arbitration period, as a 0 from another ECU is designed to indicate a higher priority message. This ensures that the higher priority message wins arbitration without generating errors.

A Bit Stuffing Error occurs when 6 of the same bits are observed in a row. After transmitting 5 of the same bits in a row, the transmitting ECU is meant to insert the opposite bit then resume normal transmissions. All CAN transceivers know to ignore the stuffed bit as 6 bits in a row are an error state.

When an ECU detects an error it sends a CAN error frame which consists of 6 dominant bits (0) in a row. The error frame itself generates a bit stuffing error, alerting every ECU on the bus that it should ignore the message it just transmitted. The bit stuffing error means a 1 to 0 bit manipulation attack, which generates a CAN

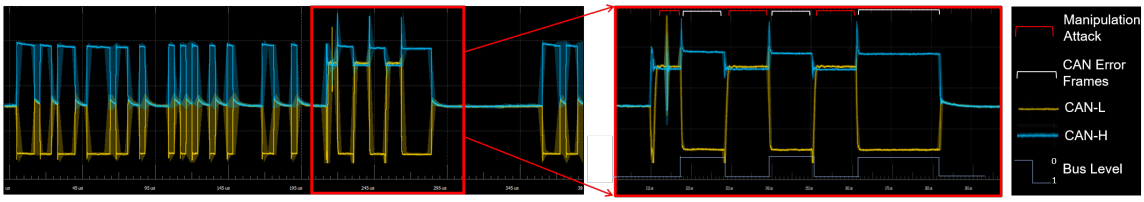


Figure 5.6: This shows a manipulation attack, performed 3 times, forcing a dominant to a recessive bit. The right figure, a zoomed in section of the left figure, depicts a 4μs attack, followed by an 8μs pause, 8μs attack, another 8μs pause and attack, before allowing the error frame to complete. When a CAN error frame is interrupted, we see the ECU transmit the error frame again. After a complete error frame is sent, the line goes passive for approximately 60μs before retransmitting the manipulated message.

error frame, inevitably forces the attacker to perform numerous 0 to 1 manipulation attacks, lest an error frame is completed and the message is dropped. Notably the victim ECU will retransmit a message interrupted by an error frame. This makes interrupting messages an ineffective technique for silencing legitimate ECUs. The bus-off attack [11] complicates this, but it is covered in our security analysis.

We make the distinction between 1 to 0 and 0 to 1 manipulation attacks as the former is intentionally simple in CAN and difficult to distinguish from a normal bit change. On the other hand, a 0 to 1 transition requires the attacker to drive the bus against a victim ECU which is actively providing power to CAN-H and sinking CAN-L as it transmits a 0. For the rest of this section we focus on performing and detecting 0 to 1 manipulation attacks, as they are the inevitable consequence of any manipulation attack where the attacker wants the bus to process the manipulated message.

5.3.2 Performing a Manipulation Attack

An attacker has 3 options to manipulate a 0 to a 1: ground CAN-H to CAN-L, increase CAN-L to CAN-H, or meet somewhere in the middle. In terms of detection, a manipulation attack which significantly deviates from the passive bus voltage is immediately obvious, as the passive voltage level is relatively constant, but the attack still works.

In Figure 5.6 we demonstrate a manipulation attack on a real CAN bus, changing a 0 to a 1 three times by increasing the voltage of CAN-L such that the differential voltage between CAN-H and CAN-L is 0V. In terms of hardware, we used a standard CAN transceiver to time the attack and P-channel MOSFETs to source/sink voltage for overriding the existing signal. Each manipulation attack produces an error frame and manipulating this frame results in another error frame. In Figure 5.6 we can see

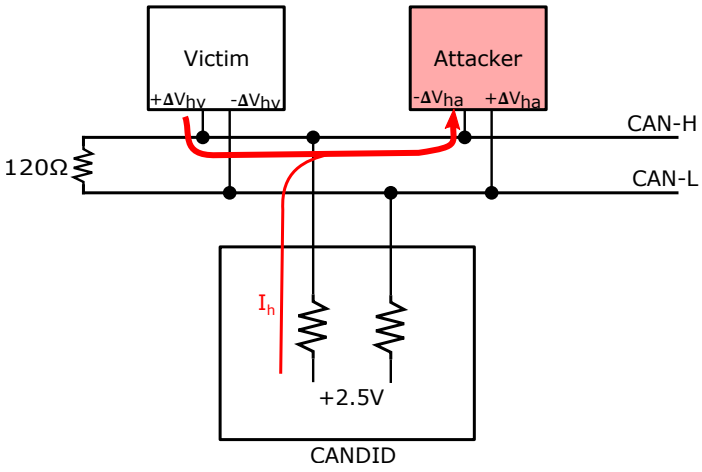


Figure 5.7: Depicts an active 0 to 1 manipulation attack on the CAN bus. The changing potential from the attacker overriding the victim's transmission from high to low results in a current draw from CANDID pullup resistors. This changing potential does not occur on a passive bus.

these repeated error frames from each manipulation attack. As explained in Chapter 3, Error frames will stop transmitting a dominant bit after an error count of 127 and ECUs will stop transmitting entirely after an error count of 255 [11]. The errors caused by a manipulation attack increase the count by 8, meaning it only takes 16 bit manipulations for passive bit errors. Once in a passive bit error state, we can simply transmit dominant bits. After 16 dominant bit transmissions the victim ECU will enter a bus-off state and stop transmitting until reset.

Manipulation attacks allow us to reach a bus-off state much faster than previous work, which took 32 interrupted messages. Additionally, the original attack required the attacker to predict when messages would transmit and have two messages win arbitration. Our attack can apply to aperiodic messages, as it only needs to read the ID of a message as it goes across the bus [11]. We complete a bus-off attack within the span of a single message, without allowing any error frames to complete, and without having the bus discard the manipulated message.

5.3.3 Detecting a Manipulation Attack

In Chapter 3 we detect a manipulation attack by changing the CAN error process such that an interrupted error frame does not increase the error frame count until the error frame is completed. While this approach is effective, it is impractical to implement on existing systems. It follows that we need a more generic solution for detecting bit-flips/manipulation attacks. Voltage fingerprinting and monitoring

techniques would almost certainly detect a manipulation attack, but this is far too over-engineered for what we need to do. After all, voltage fingerprinting's main benefit is detecting spoofing attacks and our timing analysis does this already. Instead, we can build a more targeted detection system using our knowledge of the CAN protocol.

We detect a manipulation attack by monitoring the current during a dominant to passive bus transition, as shown in Figure 5.7. Normally this transition is from no ECUs transmitting, but during a manipulation attack the attacker is actively driving the bus towards the passive state. This drive toward the passive state never occurs on a normal CAN bus. The result of this drive is an anomalous change in electric potential. CANDID can see this change in potential because each ECU, including CANDID, is connected to the bus in parallel. This means that any change in potential results in a measurable change in current from CANDID's pull up resistors. An attacker must perform a 0 to 1 manipulation attack because the initial bit flip induces the 6 zero bits of a CAN frame and if that error frame completes then any standard CAN transceiver easily detects it. By keeping our own error counter for each ECU we can alert when an ECU is approaching the bus-off state. If an attacker attempts to circumvent error frame detection by continuously flipping bits, then they must perform a 0 to 1 bit flip as long as the error counter is below 128. We alert if we see any ECU reaching an error count of 128. To summarize, after a single bit flip the victim will produce 6 zero bits. An attacker can either continue flipping bits to ensure the message is not dropped, at which point we detect the 0 to 1 transition which does not occur normally on the bus, or the attacker can allow error frames to complete until error frames start transmitting 6 one bits instead, which we detect by keeping a counter of errors produced by each ECU. If the attacker decides to let the message drop after one bit flip then there is almost no impact on the bus, as the victim ECU will immediately re-transmit the dropped message. By having our solution take advantage of CAN errors and passive bus levels, we are able to detect these attacks on any CAN system without attempting to fingerprint each ECU and dealing with the complications of electrically fingerprinting electronics. Our solution avoids the cat and mouse game of adapting to how accurately an attacker can recreate a signal.

5.4 Security Analysis

In the adversary model we define a successful attack as changing the state of the system without CANDID alerting. To provide a security guarantee against an attacker we analyze spoofing, masquerade, and manipulation attacks separately. Recall that

our detection system is connected as a normal ECU to the bus, with an amperage monitor on the pullup resistors in our CAN transceiver. This security analysis assumes the attacker knows CANDID is installed.

5.4.1 Securing Against Spoofing Attacks

The attacker has somehow added a device to the CAN bus and wishes to modify the state of the system by impersonating existing ECUs. To perform a spoofing attack, an attacker must send a periodic or aperiodic message. The security model is the same for both but our approach for detecting them is different. Let us discuss the approach for an adversary transmitting periodic messages. First the attacker must select a message ID. Because of CANDID's timing training, the attacker must select a periodic ID previously observed on the system. When the attacker transmits their desired periodic ID, our IDS checks the transmission interval of that message. This interval is selected based on the state of the system and defined during training. If the time between the attacker's message and the previous time we saw that ID is not within the known transmission interval then we alert on that message. If it is within the known interval, then the original device will send a message. The original device's transmission guarantees an alert by the end of the interval on whichever message came second. Given that the original transmission causes the attacker to be detected, an attacker must ensure that no other ECUs are sending their desired ID. It follows that to transmit their desired ID the attacker must prevent the original ECU from speaking. An attacker can do this by colliding with the original message, performing an address hijack attack, or uploading malicious firmware. Collisions generate a CAN error frame, which are covered under our security analysis for manipulation attacks. An address hijack attack causes the victim ECU to transmit a specific 'address not found' message and requires the attacker to send a completely new data field. Mangling this alert message to hide the address hijack attack is secured by the same logic as collisions. Firmware upgrades are trivial to detect if we assume our IDS knows all legitimate firmware versions and can calculate a hash of each CAN frame involved in the firmware update. This means an attacker can not prevent an ECU from transmitting without being detected. Consequently, if a legitimate ECU is transmitting periodic messages then CANDID can reliably detect an attacker injecting periodic messages.

For aperiodic messages there is no existing ECU transmitting the message. Once an attacker injects an aperiodic message, CANDID's data correlation training verifies that the provided aperiodic message should have been transmitted based on the state of the system. This is done through a series of data dependency checks. To

bypass this check the attacker must inject additional messages prior to their desired aperoidic message which set the appropriate data dependencies. However, these data dependencies belong to periodic messages. This means that our security analysis for periodic messages now applies to aperiodic messages. After detecting periodic and aperiodic messages, our attacker is unable to inject any messages without being detected.

5.4.2 Securing Against Masquerade Attacks

For a masquerade attack the attacker has taken over an existing ECU and aims to send malicious messages on the bus as the victim ECU. The masquerade attack's security analysis is the same as the spoofing attack's security analysis with three distinct differences: periodic message detection relies on data dependencies, the attacker can lie about the physical defects in the system, and CANDID is unable to detect small changes to messages with no data dependencies.

The attacker transmits messages within the known transmission interval of the victim ECU. As there is no other ECU transmitting this normally periodic message, CANDID's timing features no longer detect this attacker. However, the injected periodic message has data dependencies. To bypass these dependencies the attacker must transmit the parent messages, or transmit within the error bound allowed by the CUSUM analysis for the chosen message. Because the parent message is coming from a trusted ECU, the attacker would need to perform a spoofing attack as that ECU. This attack is protected by the aforementioned timing-based spoofing detection mechanisms. This leaves the attacker with the option to transmit messages within the CUSUM error threshold. By definition, attacks within this CUSUM error threshold have limited impact on the system. This is because the error threshold is determined by training on the innate fluctuations of a normally operating system. Given that the attacker cannot successfully transmit messages with dependencies, they can only transmit messages with no data dependencies to non-corrupted ECUs. This means a corrupted ECU has control over what its victim transmits, but cannot reach outside of that domain due to data dependencies and CANDID's timing analysis.

The attacker is capable of lying about the state of the system for its independent messages. However, this lie must be physically possible in the system. Take an attacker omitting messages reporting an oil leak. The rest of the system will believe these omissions until the lie is physically impossible for the rest of the system to maintain. At this point CANDID will alert on the dependent, trusted ECUs as their data diverges from what is expected by the corrupted ECU's messages. The damage

may already be done, but this way the defender is aware that an attack is occurring and can properly remediate the problem and prevent it from spreading across a fleet. Otherwise a cyber attack may be treated as a normal maintenance issue.

5.4.3 Securing Against Manipulation Attacks

To perform a manipulation attack, the adversary must modify the voltage of CAN-H and CAN-L such that the differential voltage translates to their desired bit. Any attempts at manipulating a bit result in the victim ECU raising a bit monitoring error, as the bit on the bus is different from the transmitted bit. If the error frame completes, then CANDID detects an error frame from the victim ECU and CANDID's internal error counter increments for that ECU. CANDID alerts if the error counter for any ECU exceeds 127 (where errors start transmitting recessive bits). This cutoff ensures that CANDID does not transmit false positives for one-off legitimate errors, but it does alert if any ECU reaches a serious error state. Allowing a number of error frames less than 255 to complete is inconsequential to the state of the bus, as every interrupt message is ignored and then retransmited. This means that the attacker cannot achieve their goal without allowing enough error frames to complete that CANDID detects it. Now that an attacker cannot allow error frames to complete, they are left with one option - interrupting error frames. Interrupting an error frame requires flipping a 0 to a 1. Each bit-flip during an error frame results in an error count increase which CANDID does not observe. Each manipulated error frame results in the victim ECU transmitting another error frame for at minimum the length of an extended CAN packet. The necessity of flipping a 0 to a 1 to interrupt error frames ensures CANDID does not need to be able to detect a 1 to 0 bit flip.

Let us discuss how the attacker performs this manipulation attack. When modifying the voltage over CAN-H and CAN-L the attacker can set the voltage of CAN-H and CAN-L to any value. However, the voltage of a passive (1) CAN bus is consistent. This can be seen in Figure 5.6, where CAN-H and CAN-L reset after we stop our attack and the bus begins transmitting a recessive bit. If the voltage of CAN-H and CAN-L are significantly different from the passive voltage for longer than a microsecond then the attack is obvious. Now the attacker must sink CAN-H and source CAN-L such that the voltage of both are approximately the same as a passive bus.

To sink CAN-H and source CAN-L the attacker must overpower the victim ECU. The victim ECU is actively driving CAN-H high and CAN-L low to transmit an error frame. The change in potential from transmitting an error frame results in current flowing towards the attacker's CAN-H connection and away from the attacker's

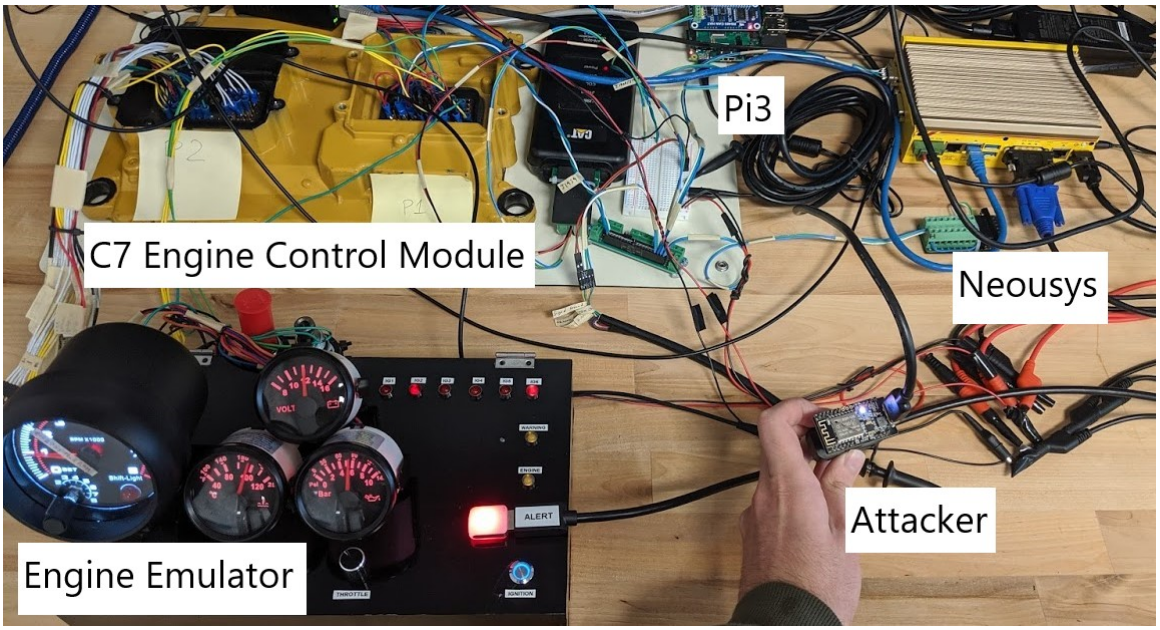


Figure 5.8: Our testbench with an engine emulator in the bottom left, C7 Engine Control Module in the top left, collection devices in the top right, and an implant attacker in hand in the bottom right. The red light in the middle left represents a mechanism for alerting the operator of the vehicle.

CAN-L connection. We observe these draws in current through the pullup resistors in CANDID, as each ECU is connected in parallel to the CAN bus. In contrast, the passive bus has no transmitting ECUs. Zero transmitting ECUs means a near zero change in potential and current. CANDID monitors the current across its pullup resistors whenever the bus is transmitting a 1 signal. If that current is different from the near-zero current of a passive bus, then CANDID alerts. Given that a critical mass of CAN error frames result in detection, the attacker must eventually force the bus to a 1 and the current will be different from that of a passive bus. This leaves the attacker unable to perform a manipulation attack undetected. Now an attacker must inject new messages onto the CAN bus to perform a successful attack. Our security analysis for spoofing and masquerade attacks covers these new messages being injected onto the bus.

5.4.4 Security Analysis Conclusion

CANDID reliably detects detecting all periodic messages and data dependent aperiodic messages for spoofing attacks. For masquerade attacks we detect data dependent periodic messages and data dependent aperiodic messages. Our ability to detect any manipulation attack ensures CANDID can rely on the data from trusted ECUs.

5.5 Implementation and Experimental Results

This section tests the performance of our implementation, demonstrates our experimental results, and compares CANDID to existing work.

5.5.1 Implementation

Our implementation works on both a dump of CAN packets and a live connection from the CAN bus. That live connection uses one of two collection devices: a Neousys ruggedized vehicle computer [60] or a Raspberry Pi 3 CAN Hat [99]. For development and attack testing we use a testbench with a C7 Caterpillar Diesel Engine Control Module, an engine emulator, an OBD-II computer acting as an implant, and both aforementioned collection devices connected to the same CAN bus. This can be seen in Figure 5.8. All live testing was done on a 250Kbit/s CAN bus.

For our manipulation attack detection we install an amperage monitor in series with the CAN transceiver. Alternatively, we could connect a normal oscilloscope to the CAN-H and CAN-L connections with a common ground and measure the difference in voltage. But this oscilloscope connection requires knowledge of the impedance of the bus line. This makes it ineffective against implants.

All J1939 specification parsing and IDS functionality is written in Python, primarily for rapid feature development. CAN frame collection is written in C++ and uses Linux PCAN drivers to process incoming messages. We test the performance of our system using profiling on a data capture and by testing on a fully operational moving vehicle with two CAN buses. By testing to see if the time it takes for CANDID to process a CAN packet is less than the time for another CAN packet to be transmitted, we know if CANDID can be used in real world settings. Our profiling indicates CANDID enriches and evaluates an individual CAN packet in less than 500 microseconds, even when duplicating the CANDID process to work on both CAN buses. On a 250Kbit/s bus, a single CAN packet takes 584 microseconds. This time means CANDID is fast enough to not fall behind the bus. To test CANDID's performance on a real bus, we turn off the rest of the bus while a separate device transmits a test alert message. We observe this test alert instantaneously, indicating CANDID's processing does not fall behind the bus.

5.5.2 Experiment

Evaluating CANDID requires us to test it for each attack and against a data set without attacks. To this end, we test CANDID against 6 representative attacks and

verify CANDID's behavior against a clean dataset. We do not perform experiments on manipulation attack detection beyond executing the attack and validating the repeated CAN error frames as described in Section 5.3.

For our evaluation we have access to datasets from a truck and four tactical vehicles. CANDID is also evaluated while deployed on a full system to ensure CANDID can keep up with real bus speeds, as described above. From a detection evaluation perspective, CANDID's timing and data capabilities perform the same on collected data captures as a real bus. We have three hours of driving data from our truck dataset and 145, 55, 170, and 16 data captures for the four tactical vehicle respectively. These captures average 200,000 CAN packets each, with varied driving conditions including: off-road, hill, track, and road testing with faults. Across our datasets we observe: 124 different message types, 328 data parameters and 72 million packets. The observed message types are reflected in both datasets, though the tactical vehicles datasets include 41 types not seen in the truck dataset.

For our evaluation we need to create a training set off of these datasets. Based on the differences in observed messages, training is not transferable between vehicles with different ECUs. We trained on roughly one million CAN packets (1-2 hours of driving time) for each vehicle. The most important aspect of this training is ensuring all message types are seen, and that any transitions between vehicles states are included in the traffic. To verify that a normally operating system does not result in false positives we run CANDID against our truck and tactical vehicle datasets. We do not run attacks on vehicles while collecting this data to establish a baseline and for safety reasons.

For spoofing attacks we inject one of five attacks into a random spot in the aforementioned truck dataset then run the dataset against CANDID. We use randomized spots in the dataset to emulate attacker behavior, but analyze the intervals produced by CANDID's training to ensure they meet the bounding criteria necessary for the security principles around timing intervals to hold. We only use the truck dataset, as the difference between vehicles is inconsequential for detecting attacks once the training mechanism can be trusted. Our five attacks are chosen to demonstrate CANDID's ability to detect messages with one timing interval, messages with multiple timing intervals, aperiodic messages sent as a condition of the data field, completely new messages, and J1939 specific attacks. These attacks cover the entirety of spoofing attacks and give us confidence in CANDID's ability to detect any other spoofing attacks, as well as common mechanisms for disabling existing ECUs.

We repeat each of the five spoofing attacks 100 times. The five attacks are: spoofing periodic engine message 61445, spoofing the multiple timing cruise control message, spoofing an aperiodic diagnostic message which enables the check engine light, transmitting a message type not previously seen by the engine, and an address hijack attack. The aforementioned attacks were tested on our testbench and work as expected. To test against a masquerade attack we change the engine speed of engine PGN 61444 at random places in the dataset. We modify 30 data fields and add an additional case for an attacker incrementally changing the data.

5.5.3 Results

Our first tests involve running CANDID against truck and tactical vehicle datasets to test if any alerts are generated on a clean dataset. After training, we process each CAN packet for each dataset. We observed no false positives as a result of messages operating outside their normal timing bounds. This indicates the predictability and consistency of timing intervals.

Our data analysis implementation uncovered a small number of proprietary message types in our data set (from military vehicles), which led to false positives as CANDID had no way of interpreting their data fields. Proprietary messages are not a problem for the equipment manufacturer, but CANDID lacks documentation and ignores them if they do not immediately fit within one timing interval.

We found that 4.76% of IDs in our datasets have data dependent transmission times. These 4.76% of IDs, including messages like the cruise control message from our introduction, would cause false positives in existing timing-only research. The benefit of CANDID is even more pronounced for our NMEA-2000 data. We observed 11.86% of our NMEA 2000 data had data dependent transmission times. CANDID was able to correctly identify the data fields which modify message timing, such that we can safely process these IDs.

To test the success of a spoofing attack we inject a single attack randomly into our truck dataset, scan the dataset with CANDID, then scrub the dataset. We repeat this experiment 100 times for each of our 5 attacks for a total of 500 injected messages. In terms of interval training, we found that CANDID was able to produce intervals with lowerbounds greater than the width of the interval such that the security principle holds. For the masquerade attack, we modify a data value in a message rather than inject a new one. We do this 30 times, plus an extra 5 messages in a sequential replacement attack. Table 5.1 depicts the results of our experiment. For the spoofing attack we achieve 100% recall and precision. For the masquerade attack we achieve a

Description	Attack	Precision (%)	Recall (%)
Periodic Engine Message	Spoofing	100	100
Cruise Control Message	Spoofing	100	100
Enable Engine Diagnostic	Spoofing	100	100
New Message Injection	Spoofing	100	100
Address Hijack	Spoofing	100	100
Corrupted ECU	Masquerade	100	93.3

Table 5.1: A summary of results for 6 practical attacks. For each attack, a message is injected or replaced randomly in a trace of 1.5 million packets. Each attack is repeated 100 times.

93.3% recall and 100% precision. Attacks that change the value to something within the variance of the transmission are not detected, but such attacks have a limited impact on the system.

The *Periodic Engine Message* attack demonstrates detecting fixed rate periodic traffic, where each attack causes the victim ID's intervals to diverge from the fixed rate. The *Cruise Control Message* attack uses the same principle as the Periodic Engine Message attack, with multiple intervals. Detecting it requires CANDID to track the state of the system, select the appropriate time interval, then alert on any messages outside that time interval. The *Enable Engine Diagnostic* attack extends the periodicity proven by our periodic engine message to an aperiodic message, as the chosen diagnostic only turns on when the engine speed is over 2400 RPM. The *New Message Injection* attack proves that CANDID can detect unknown IDs, this emulates the heartbeat functionality of a popular CAN cryptography solution [70], but for extended CAN packets and without limiting the system configuration. The *address hijack* attack requires CANDID to save the observed data for each address message (PGN 60928) and alert on any new data fields.

Explaining the corrupted ECU attack and what we do and do not detect requires more detail. The results can be seen in Figure 5.9. In this case our data analysis finds engine speed is dependent on the rate of change of transmission input shaft speed, meanwhile it is in the engine idle state. This state means that the engine speed has some natural fluctuations, which we account for with a CUSUM error threshold of 1.2%. A chain of incremental attacks, seen in the zoomed in section of Figure 5.9, remained undetected over successive smaller changes until the final message where the cumulative error was larger than possible in that idle state time period. CANDID does not detect some small deviations, resulting in false negatives. Even so, the attacker is

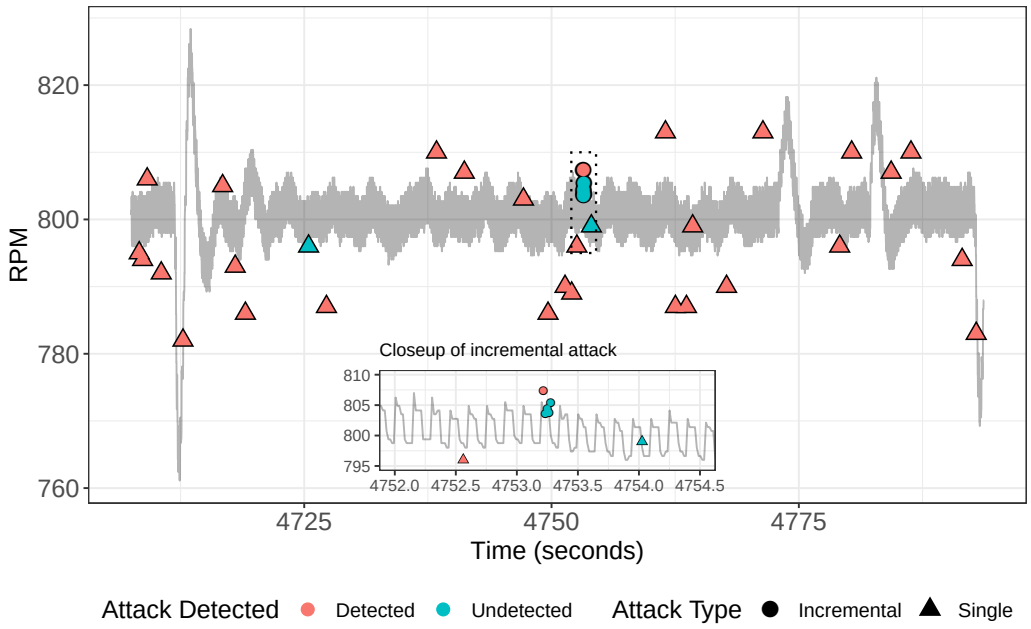


Figure 5.9: A masquerade attack, with a zoom in on the incremental attack. Here an attacker modifies the RPM as a corrupted engine ECU. Attacks are detected using CUSUM on deviations from trained correlations. Undetected attacks have an insignificant impact on the system.

limited in their desired engine speed despite controlling the engine control module. Further restricting changes to the data field is likely to result in many false positives, though further restrictions may be necessary for systems where small changes to scalar values are of consequence.

In summary, tests against a clean dataset result in false positives from enrichment errors with proprietary messages, but the tests have high accuracy on standard traffic. While an implementation can approximate out proprietary values, it depends on how sensitive the defender is about small fluctuations in data. We are able to detect all of our spoofing attacks. This includes periodic, multi-periodic, and aperiodic messages. Identifying these data dependent periodic messages makes the bus more predictable for the defender. Our experiment against a masquerade attack is detected through data dependencies to periodic messages. Detecting multi-periodic, and aperiodic messages from any attacker, as well as masquerade attacks from periodic messages, is a significant improvement on previous CAN intrusion detection research.

	Techniques				Attack Coverage			
Proposal	Timing	Data	Fingerprinting	Crypto	Spoof	A-Spoof	Masquerade	Manipulate
Song et. al [86]	✓	-	-	-	●	○	○	○
Cho and Shin [12]	✓	-	✓	-	●	○	○	○
Groza et. al [33]	-	-	-	✓	●	●	○	●
Wasicek et. al [98]	-	✓	-	-	○	○	●	◐
Choi et. al [14]	-	-	✓	-	●	●	○	●
Sagong [76]	-	-	✓	-	○	○	○	●
Zhou et. al [103]	✓	✓	-	-	●	○	○	○
Quinonez et. al [72]	-	✓	-	-	○	○	●	◐
Daily et. al [19]	-	-	-	✓	●	●	○	●
CANDID	✓	✓	-	-	●	●	●	●

Table 5.2: Comparison between CANDID and related CAN IDS research by technique and attack coverage. Half Circles indicate that we believe their research may cover these attacks, though our adversary models differ. Spoofing attacks pretend to be another ECU on the bus, masquerade attacks control an existing ECU and send that ECU's normal set of messages, and manipulation modifies individual bits from existing messages. A-Spoof is aperiodic spoofing and encompasses messages without a single regular transmission interval. This table is sorted by year of publication.

5.5.4 Comparing CANDID to Existing Work

To contextualize the results of CANDID, we examine the three modules of CANDID and compare them to existing work. A summary of this comparison can be seen in Table 5.2. We define our three modules as a timing module (which encompasses the interval training, and modality based interval training), a data-dependency module, and a manipulation attack detection module. The modality-based timing approach is an improvement on existing CAN timing based IDS approaches, as it removes the assumption that messages are strictly periodic. This assumption is known to be false for some subset of messages [44]. While post-detection analysis can reveal aperiodic false positives [12], it does not provide a security guarantee to those messages. CANDID provides an improved timing interval bounding approach, which is then extended to aperiodic messages via this modality training. CANDID establishes our time bounding in a way that is unspoofable in our system model, unlike clock skew analysis which can be faked [77]. An attacker cannot fake an interval as long as the legitimate ECU is still transmitting. Our IDS is designed to catch the attacks that would disable a legitimate ECU. The result is every message has a security guarantee that is non-spoofable. An attacker cannot avoid CANDID simply by choosing a message with a complicated transmission interval.

Now let us examine our data-dependency module. Existing CAN work applies data analysis via an Artificial Neural Network (ANN) which learns how data is expected to deviate in sliding time windows [98]. CANDID's timing analysis exists separately from

the data-dependency training, which ensures that an attacker cannot overload the bus with illegitimate messages. Quinonez's physical invariance research has similar timing issues [72]. CANDID's use of dependencies between data fields, with CUSUM applied to the deviation from the expected value, is novel in that it is applied across the entire vehicle. The physical models presented by Wasicek [98] and Quinonez [72] are applied to specific subsystems and simplistic line following robots respectively. This results in assumptions about which systems are interconnected and risks gaps in detection coverage. Our layering of data contextualized timing analysis further limits the set of messages an attacker can send, while ensuring an attacker is limited when sending malicious data correlated to legitimate ECUs.

Our manipulation attack detection is most similar to voltage based detection systems. These monitor the voltage of the bus to create electrical fingerprints of the system [41], [14]. While not explicitly designed to detect manipulation attacks, the sudden change in voltage required to flip a bit would likely modify the fingerprint of the message and trigger the detection system. CANDID offers two novel improvements. The restriction of CANDID's monitoring system from fingerprinting the whole system to only 0 to 1 bit flips reduces false positives by removing the need for constant retraining. Overlapping our manipulation attack detection with timing-based spoofing detection ensures we do not lose any of the security guarantees delivered by voltage fingerprinting. The second improvement is using the current in the pull up resistors during 0 to 1 transitions. In a generic sense, monitoring only current would create a false positive ridden detection system. However, the nature of the CAN bus and its error process ensures that CANDID knows that a 0 to 1 transition must occur in response to a bit flip and that when a 0 to 1 transition occurs any deviations from the passive state must be erroneous. This comparison to a passive state ensures that our approach is suited to detecting voltage IDS circumvention techniques like those used by Bhatia [5], who uses multiple attackers to hide the voltage being changed. Any attempt to overpower the grounding of a CAN line is inherently anomalous during a 0 to 1 bit transition.

5.6 Discussion

In this section we elaborate on CANDID's training process, CAN incident response processes, and issues with alerting on the J1939 specification.

5.6.1 Training

As described in Section 5.2.1, CANDID has 3 main training components: training on timing intervals, using data to split those timing intervals, and training on a CUSUM error threshold for correlated data fields. This training has to be done for each vehicle, preferably with a representative dataset of the system: powered on, idling, and driving in variable conditions.

The question is then what happens if the vehicle diverges outside of our training. We are unconcerned about this for two reasons. CANDID trains on normal behavior then expands the timing intervals to the maximum possible while still maintaining our outlined security principles. This ensures our timing intervals are not prone to false positives from poor training sets, but are still effective at detecting spoofing attacks. Additionally, CANDID accounts for fluctuations in vehicle data by using CUSUM, as we can safely assume the vehicle will self correct and minor errors will not propagate through a vehicle indefinitely. Our results show that an attacker can make small changes to the data field, but none more significant than the natural fluctuations of normal data. These measures increase our confidence that even if our training did not capture all outliers, it is resilient to future outliers. This makes CANDID an effective and resilient IDS.

5.6.2 After an Alert

The question with any intrusion detection system is how an alert should be communicated and what can be done about it. While a full incident response process for CAN is outside the scope of this thesis, we can make a few suggestions. Any communication of the alert to the operator of the vehicle must be done through a mechanism external to the compromised CAN bus. In our testbench, seen in Figure 5.8, we use an LED light, but more context can be helpful. For example, if an alert indicates the communications system is no longer transmitting the appropriate messages, then the operator can know to react accordingly. This is particularly true for military systems, where operators are trained to react with specific maneuvers if mobility, communications, weapons, or other systems are disabled. Notably an attacker may abuse this operator training by falsely raising vehicle indicators. Cyber-aware response maneuvers would need to be added to operator training.

As for broader incident response, CANDID compresses and logs all CAN packets observed on the bus. This makes it a simple process to upload the messages to any traditional incident response platform such as Splunk [87] or ELK [23]. The alerts are

logged with the alerting message and its timestamp, such that analyzing the messages surrounding an alert is trivial. Voltage based systems would want to log the raw signal surrounding an alert.

5.6.3 Specification Deviations and Specification Rules

The risk of basing a detection system off a standard is that a system may deviate from that standard. We saw this in our early CAN IDS development. Our first iteration of CANDID was a rules framework automatically generated by parsing the J1939 Digital Annex [85]. After running these rules against multiple CAN datasets, we found a few violations. The J1939 standard sets the value of uninstalled and disabled features to their maximum value, but some implementations set disabled values to 0 instead. SPN values were frequently outside the 'operational range' specified in the standard. The timing intervals specified in the standard were also horrifically inaccurate and became the inspiration for the current iteration of CANDID.

The intent of these rules was to catch attackers making mistakes. An attacker being careless in how they craft their payload would be detected as the IDS notices an uninstalled feature suddenly reporting being enabled, or a data value which does goes outside the operational range of the system. We programmatically produced over 30,000 rules with this detection philosophy. But after iterating on this concept a few times it became clear that while these rules could detect an attacker making a mistake, it was far more likely to be a source of false positives and a pain point every time we wanted to integrate CANDID onto a new vehicle. From a practicality perspective, running these rules against every message was too computationally expensive for the security they provided. Far better to detect an attacker generally interacting with the bus, than to detect the attacker making a silly mistake on the bus. Specification-based rules still have a place, but not based on the entire specification. Rules based on known error states are a good fit for early alerting, as this allows for short-circuiting the detection system. Rules which detect attacker mistakes are a good fit in front of a data analysis module for similar reasons. They are computationally less expensive than a full data analysis process, but also represent edge cases which may look like small deviations to protocol-oblivious data analysis. A defender using specification rules needs to base their training process on which specification information is manufacturer specific. However, it is difficult to discover this information without comparing vehicle datasets from multiple manufacturers.

5.7 Conclusion

In this chapter we propose CANDID, a novel IDS which detects anomalies on the CAN bus via timing, data, and voltage techniques. Combining, improving, and developing these techniques ensures CANDID is more accurate than previous work, as it has greater context for how the system is meant to operate without the cons of any individual approach. CANDID's timing analysis isolates transmission influencing data to more accurately bound when messages are going to appear. This more accuracte bounding ensures that CANDID captures safety critical messages, such as cruise control, even if they transmit with multiple timing intervals. CANDID's data analysis takes our much larger set of parameters into account, performing correlation analysis to determine when an attacker is modifying data influenced by the rest of the system. This correlation analysis limits what even the most advanced attacker can do on the system. Finally, the voltage analysis avoids traditional fingerprinting to use a more focused approach. It specifically monitors for an attacker manipulating a 0 to a 1, using the physical properties of the CAN bus to the defender's advantage. This capability is particularly important in the context of our advanced bus-off attack contribution, which puts an ECU into the bus-off state within a single message without producing any error frames. CANDID was tested against multiple vehicles and data captures and was able to detect all injected attacks without producing false positives beyond small data translation limitations. The success of our tests, our device not requiring the modification of any ECUs, and the performance of our implementation demonstrates that CANDID is a practical and effective solution to security for existing and future CAN systems.

Chapter 6

Efficient Attack Detection for MIL-STD-1553 Networks

The MIL-STD-1553 protocol [61] and its derivatives control the aircraft and weapon systems used by many global powers [102] [51] [49]. This protocol suffers from many of the issues that plague more commonly researched serial data bus protocols like CAN [7], including a complete lack of authentication on the bus. This means that that any attacker capable of transmitting a message across the bus, be it via an implant, a supply chain attack, or a remote exploit, can effectively control the aircraft [22] [48]. But the problem is actually worse with MIL-STD-1553 than other protocols. MIL-STD-1553 contains multiple mechanisms for disabling Remote Terminals (RTs), which are the computers exchanging data on the bus. Additionally, MIL-STD-1553 utilizes a Bus Controller (BC) which dictates all traffic on the bus. An attacker can easily spoof the BC to take control of the bus, and the BC does not give any indication that something is wrong.

An attacker managing to transmit data on a fighter jet may seem like fiction, but the US government has done nothing but stress their fear of exactly this. In just 2018, a Government Accountability Office report stressed the insecurity of US weapons systems [81] and the US Department of Defense issued the first update to the MIL-STD-1553 protocol since 2008, calling for improved cyber security for RTs [61]. MIL-STD-1553 has been widely used in aircraft since 1973 [102]. Replacing or retrofitting nearly five decades of aircraft is not possible with modern resources. We propose an intrusion detection and prevention system tailor designed to MIL-STD-1553 to provide a cost effective security solution.

CANDID is a good demonstration of what can be done by using the data going over the bus, but we lack that option with MIL-STD-1553. Interface Control Documents (ICDs), which allow a defender to translate MIL-STD-1553 data, are locked by the

Original Equipment Manufacturers (OEMs) and classified networks. Even if we did gain access to ICDs, they are frequently stored in formats that are difficult for a computer to parse. It is a daunting manual task to accurately translate every message used by a fighter jet; instead this chapter focuses on how we can address the attacks likely to hit a MIL-STD-1553 bus without having to translate any data. In fact, we out perform the timing-based approach of CANDID by taking the architecture of MIL-STD-1553 into account when designing our IDS.

Our IDS solution has three base requirements. Our solution must be fast enough to work as an intrusion prevention system. This means that the IDS must be capable of mangling the signal of an attacker's message before it completes to cause the rest of the system to ignore it. Our solution must be capable of detecting on the basis of a single message, such that an attack is immediately identified. Finally, our solution must be capable of detecting any one-off attacks. The intent of this final requirement goes back to our earlier example of a single message which disables a RT. This disabling message is a legitimate error message the system can use when an RT is malfunctioning. Our detection system must be able to distinguish between a legitimate error state and an attacker initiated error state.

To have confidence in a prevention system, particularly one running on an aircraft, the detection mechanism must be highly accurate. The level of accuracy demonstrated by past serial data bus detection systems is not good enough, particularly with MIL-STD-1553 being four times faster than the average CAN bus. Any number of false positives is unsustainable. To achieve higher accuracy we must take full advantage of the nuances of the protocol, specifically the nature of the bus controller and the stubs connecting it to the main bus line.

The BC is effectively a source of truth and trust for the bus because it initiates every message, including the one-off error messages referenced above. Even when receiving data, the BC sends a message asking for that data. No compliant RT transmits unless the BC asks it to. By placing our trust in the BC, we can confidently say that if a message is initiated and the bus controller is not transmitting then the message must be illegitimate. This leaves the task of determining when the bus controller is transmitting.

MIL-STD-1553, unlike other serial data buses, uses transceiver coupling stub connections to receive and transmit data. These transceivers make it so that a transmitting message is observed at a completely different voltage than a receiving message. Now by monitoring the magnitude of the voltage, we have an extremely simple and accurate detection mechanism for an attacker sending illegitimate messages.

Our final contribution is a mechanism for detecting message collisions and unexpected message structures, and by extension an attacker spoofing an RT's data or status words. Error flags built into standardized MIL-STD-1553 transceivers allow us to easily detect these spoofing and collision attacks. In combination with our ability to verify that all messages are initiated by the Bus Controller, these errors flags mean that to remain undetected an attacker must take over the BC to initiate any message and must take over a specific RT to respond as that RT within a message. Building an effective IDS around standardized protocol flags and simple voltage checks allows for greater adoption amongst MIL-STD-1553 maintainers, while still accurately detecting more attacks than state of the art detection systems.

We summarize our contributions as follows:

- An intrusion detection and prevention system for MIL-STD-1553 which can detect an attacker transmitting a command word with 100% accuracy.
- A collision detection system which utilizes standard MIL-STD-1553 errors to detect any attacker interrupting messages on the bus or spoofing another active RT.

6.1 System and Adversary Model

In this section we define our MIL-STD-1553 bus and our corresponding assumptions about how our detection system and the adversary's device are connected to that bus. Additionally we define the capabilities of the adversary. Fig. 6.1 illustrates our system and adversary model, where the red RT is corrupted or an implant and our IDS is an addition to the system.

6.1.1 System Model

Our system model is a MIL-STD-1553 serial data bus utilizing a single BC, dual bus system with up to 30 RTs connected to both. On the physical layer we assume each device is connected to the bus line using transformer coupling stub connections. The alternative, direct coupling, is banned from US Airforce and Army aircraft due to a max length of only 12 inches, no DC isolation, and a shorting of isolation resistors causing entire bus failure, among other reasons [20]. These groups also recommend not using dynamic bus control for sensitive systems, so our system model will assume a singular, static, BC. The RTs vary from system to system, but examples include engine controllers, navigation, communication systems, and sensors. Each RT is capable of

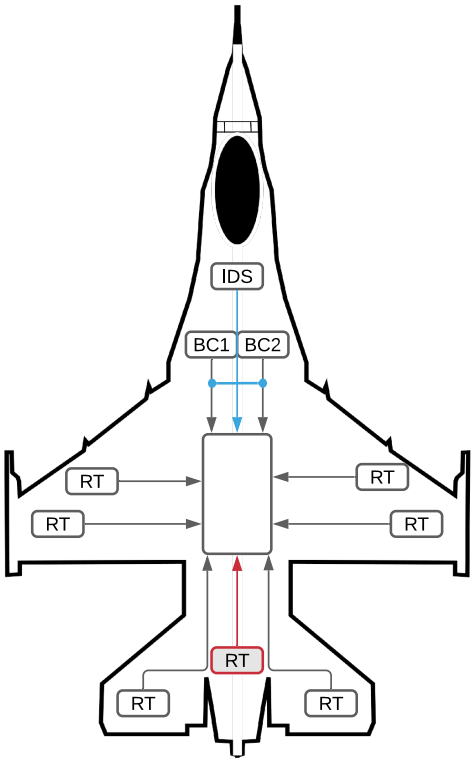


Figure 6.1: Our system and adversary model depicting how our IDS is connected to the bus. The red outlined RT is taken over via a remote attacker. BC2 represents a backup BC common to these networks.

receiving and decoding commands from the BC. We assume the physical bus has not been damaged or spliced to include in line modifiers. While it is possible for advanced RTs to detect a plethora of system errors, and illegal messages, we do not assume our system contains any of these RTs. Our system's RTs follow the baseline standard, meaning they ignore any message with invalid manchester encoding, more than 16 data bits in a word, an invalid sync, or an invalid odd parity. We assume these errors are rare on an attacker free system. We make no assumptions about how frequently or in what order messages are transmitted.

Our detection system is connected to both channels of the data bus, and can read MIL-STD-1553 messages. Additionally it is connected to the BC stub connection, enabling a boolean indicator for when the BC is transmitting. The technical details of this indicator are described in Section 6.2. We connect a high speed analog to digital converter (ADC) to the BC stub and main bus line to monitor voltages across the stub and bus. We assume our detection system is secure from wireless threats and

can indicate if uninstalled from a system.

6.1.2 Adversary Model

An attacker is successful if they are capable of transmitting or cancelling a message/word on the MIL-STD-1553 bus without being detected. An attacker targeting a MIL-STD-1553 system is likely seeking to cause system errors at strategic times for a military advantage. These errors could manifest in navigation systems misleading a pilot, engaging and firing weapon systems around allies, or even turning computers off to completely disable the system.

For the purposes of this paper we assume the BC is a trusted entity and cannot be compromised by the attacker. Our security system bolsters this assumption by making it more difficult to pivot from a foothold to the BC. We further examine this assumption in Section 5.6. Most security systems become useless when the BC, which starts every messages and tells every RT what to do, is corrupted. We also assume all transmissions are done through normal transceiver operations, and the adversary has no specialized hardware on the victim system. Implants, supply chain attacks, and remote attacks are all possible, but bit flipping that results in a valid Manchester encoding is out of the scope of this paper. The attacker has full knowledge of all of the devices on the system, and communicates by queuing command words, responding to the BC with status and data words, or interrupting messages at any point in the message. We assume the adversary has already gained access to the serial data bus through some exploit. Passive attackers are outside of the scope of this paper.

6.2 Intrusion Detection for MIL-STD-1553

When designing an intrusion detection system it is important to breakdown what data we can look for anomalies in. At a high level a MIL-STD-1553 IDS has access to the physical layer signals (voltage), the timing between messages, and the contents of the messages themselves. Let us break down each of these techniques and what attacks against MIL-STD-1553 they address.

Voltage analysis on MIL-STD-1553 is no different from what we outline in Chapter 4. Researchers have shown that by analyzing the characteristics of a particular RT's signal, it is possible to identify which device is transmitting [88]. By identifying which device is transmitting, it becomes clear which device is performing the attack. This technique inherently detects malicious devices implanted onto the network, as they have a completely new fingerprint, but it requires an additional intrusion detection

system to detect legitimate devices transmitting malicious messages. It follows that voltage fingerprinting is unable to detect a supply chain attack corrupting any RT.

Message timing relies on the assumption that when a message appears is predictable. MIL-STD-1553 theoretically uses a repeating set of messages with a dedicated time frame for one-off messages. In practice multiple implementations exist, though all implementations have some predictable periodic messages. This technique detects attackers transmitting new messages or messages out of order, but only for a subset of predictable messages. One-off user initiated messages are not covered, as this detection system has no way of determining when one-off messages should be sent. Additionally, if an attacker corrupts an existing device then this technique is unable to detect the corrupted device transmitting any periodic messages that it would normally transmit.

The final technique is analyzing the data words going over the system and determining when an attacker's message puts the overall vehicle into an anomalous state. For example, given a certain altitude, heading, and speed is a specific maneuver or sensor output anomalous? This detection technique is capable of detecting corrupted RTs responding with malicious data words. However, it is still unable to determine when a one-off message is valid.

These techniques are tried and tested, with analogous research in the mature CAN bus and car hacking space. However, the MIL-STD-1553 Bus has two differences which are key for selecting the appropriate technique. Firstly, MIL-STD-1553 has one-off messages which enable weapons systems, and mode codes which can turn off any RT with a single message. These mode codes are legitimate messages, required by the MIL-STD-1553 specification. But clearly these messages are great tools for an attacker. Any detection system for MIL-STD-1553 must detect illegitimate mode codes and one-of messages with 100% accuracy or risk catastrophic attacks going undetected. These messages, by definition, can appear at any time. The unpredictability of when these one-off messages occur disqualifies timing-based detection techniques. Analyzing the state of the system to determine if these messages should appear risks not detecting the attack 100% of the time and not differentiating between built in maintenance errors and attacks. This ambiguity, along with practical concerns over being able to enrich MIL-STD-1553 data, disqualifies analyzing data words. The only technique left is voltage based signal fingerprinting. This is where the second difference of MIL-STD-1553, the bus controller, becomes important.

All messages in MIL-STD-1553, including user initiated weapon system messages and mode codes, start with a command word and are initiated by the bus controller. Voltage fingerprinting allows us to deduce which device is transmitting. By combining

these two concepts we can determine if an observed command word was transmitted by the bus controller. The bus controller could always be corrupted, but we have full certainty that if a command word was transmitted by a device other than the bus controller then it must be an attack, as we know that only the bus controller sends command words. This technique allows us to detect any non-BC attackers initiating any new messages on the bus, including the aforementioned one-off mode codes.

6.2.1 Fingerprinting the Bus Controller

In the previous section we decided to focus on voltage fingerprinting, as it allows us to abuse the fact that the bus controller begins every message. However, every message beginning with a command word started by the BC means that we do not need to fully fingerprint the system to detect spoofed command words. Full fingerprinting would allow us to know which device is spoofing a command word, but this capability comes with the cost of high complexity and retraining difficulties [41]. Risking false positives and creating additional attack windows runs counter to our goal of an intrusion prevention system and knowing which device is executing the attack is inconsequential for preventing each attack. As for spoofed data words and status words, we argue in the next subsection that spoofing another RT while that RT is still online inevitably raises an error flag in our receiver and thus full fingerprinting is not necessary for detecting spoofed words. We propose only fingerprinting the BC. With this we maintain our ability to detect all messages initiated on the bus without over-complicating the detection system.

A typical fingerprinting solution would monitor the main bus line and train on labeled data to learn which RT is which. Instead, we can once again use the MIL-STD-1553 standard to our advantage. The BC is connected to the bus via a transformer coupling stub connection, as shown in Figure 6.2. Monitoring this connection reveals that a transmitting device operates at a significantly higher voltage than a receiving device. Applying this logic to our BC, a high voltage indicates that the BC is transmitting and a low voltage indicates the BC is receiving data. It then follows that if our IDS monitors for a command word sync starting a message, then we can check the voltage of the command word sync to determine if our trusted BC is transmitting that message or an attacker is. This solution requires no training and no high fidelity waveform analyzers. BC fingerprinting only requires a connection to the stub connecting the BC and bus line and the ability to read the magnitude of the voltage. The proximity of the attacker to the BC on the main bus line has

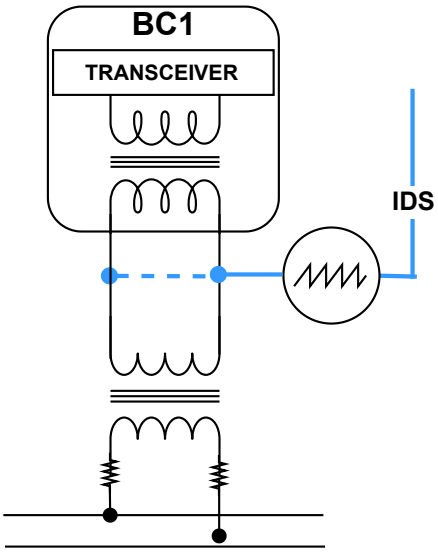


Figure 6.2: A transformer coupling stub connection with our IDS connected between two transformers. The isolation transformer after the transceiver and the coupling transformer and isolation resisters connected to the bus line ensure that the voltage at our IDS connection is significantly more when the BC is transmitting than when it is receiving.

no consequence on this technique due to the main voltage drop-off being behind the transformer.

However, we still need to account for an attacker intentionally circumventing this system by operating the bus at a higher voltage than normal. Transmitting the attacker message at a higher voltage than normal makes it appear, from the perspective of our BC stub connection, as if the bus controller is transmitting the message. Our IDS then fails to detect this attack. To address this technique, we must also monitor the voltage of the main bus line. We know only the bus controller is meant to be transmitting command words, meaning if the differential voltage of a command word is substantially different from the previous command word then our IDS throws an alert. If this solution were too restrictive it could risk false positives, but the magnitude of the difference in differential voltage between transmitting and receiving is approximately 8 volts peaks to peak. By permitting any fluctuations up to 4 volts peak to peak, we are safe from normal voltage fluctuations while detecting every attack message. The differential nature of reading voltage from a MIL-STD-1553 line helps ensure our system is not prone to environmental errors. Our IDS works without training, as we only need to alert on deviations from a single source. Any attempts to circumvent this technique by slowly raising the voltage over subsequent command word

messages results in alerts as the non-BC command words are transmitted. Raising the voltage significantly results in voltage deviation alerts. The combination of our two voltage detection systems leaves the attacker with no options for transmitting a command word undetected.

6.2.2 Collisions and Spoofed Status and Data Words

This section focuses on what happens when an attacker attempts to spoof an RT's response or intentionally collide with a word to invalidate it. Collisions might occur if an attacker creates a duplicate RT address or transmits at the same time. The collided with message will almost certainly be undecodable nonsense with Manchester encoding errors or sync waveform errors. As for spoofing, there is some nuance between spoofing status and data words. Spoofing data words is the case more likely to cause a collision as data is always sent immediately after an RT or BC has started transmitting a different word. This means to avoid a collision an attacker must send extra data words on the end. These extra words may be ignored or may be interpreted in a way that results in an exploit. Spoofing status words has three cases: an attacker can transmit before, during, or after an RT responds. Transmitting before the RT responds does not result in a collision [48], as the legitimate RT thinks the spoofed message is a command word because the sync waveforms are the same and a status word at that time would make no sense. The attacker is using the same address as the command word it is responding to, otherwise a BC error occurs from a mismatched command and status word address. At this point two things can happen. Either the legitimate RT sees an illegal command word and does not transmit further or the legitimate RT transmits a status word response to the attacker's message. Our system model assumes illegal command functionality is not enabled, however we believe it is a fair assumption that if illegal functionality is enabled then any security system can trivially observe an illegal message on the bus. If the legitimate RT responds as a status word, then our security system now thinks that a command word was sent, a status word responded, and now views the legitimate RTs status word as an illegal command word. This is detected by our BC fingerprinting security mechanism. We do not consider alerting on the legitimate RT a false positive considering that it is clear that one of the two messages sent right after each other is an attack. A defender could differentiate these cases by monitoring for fast status word responses. The attacker transmitting concurrently triggers a collision. The attacker transmitting after the RT is sending a command word as far as our security system is concerned, meaning our BC fingerprinting detection can detect the attack.

Algorithm 3 MIL-STD-1553 Alerting

```
1: //Execute on Command Word
2: while commandSync do
3:     bcVoltage ← getBcTapVoltage()
4:     mainVoltage ← getBusVoltage()
5:     //Check for Command Word not from BC
6:     if bcVoltage < 10 then alert()
7:     //Check for deviations from known voltage for BC
8:     if (trained − 2) > mainVoltage > (trained + 2) then alert()
9: //Collision Detection - After Message Recv
10: if wordCount||invalidWord||responseTimeOut then alert()
```

The above cases can be summarized as collisions, extra command words, and extra data words. Our BC fingerprinting mechanism handles any command words while the collisions and extra data words are handled by error flags built into standard MIL-STD-1553 transceivers. Through these flags we can detect any collision, as long as that collision does not nullify a message such that no encoding error is observed. We believe this is not plausible with commercial off the shelf MIL-STD-1553 transceivers. We describe the relevant errors below.

Invalid Word Errors indicate a sync field error, Manchester encoding error, parity error, and/or bit count error. This is useful for indicating collisions.

Word Count Errors are triggered by an attacker transmitting a number of data words inequal to the value set in the command word. They might wish to do this to exploit a parsing error and cause some malicious effect but this error flag makes it simple to detect.

Response Timeout Errors are registered whenever an RT does not respond or responds after a configurable amount of time (between 18.5 and 128 microseconds). This is useful for detecting an attacker holding the bus high such that no messages are processed.

In aggregate, invalid word errors ensure we detect a collision, word count errors detect extra data sent on the bus, and response timeout detects a denial of service from the bus being held high. Now an attacker needs to takeover a device to transmit as it and remain undetected. However, the attacker's target is not always vulnerable to supply chain or remote attacks. The attacker may need to pivot. We describe how our detection system restricts pivoting in our security analysis. A summary of our different detection techniques can be seen in Algorithm 3.

6.3 Security Analysis

In this security analysis we examine if an attacker can still achieve their goals while being fully aware of our intrusion detection system and how it works. The attacker's goal is injecting a message onto the bus or interrupting messages on the bus without being detected. Let us begin with the attacker trying to initiate a message on the bus.

Initiating a message requires the attacker to transmit a command word over the bus. Our detection system recognizes the initial command word sync waveform within the first 3 microseconds of the message. The attacker's message when received by our BC stub connection is at less than 10 differential volts, and so is alerted on as a non-BC initiated command word and detected. To bypass this detection system, the attacker can transmit at a higher voltage across the main bus, such that the transformed voltage going into the bus controller is greater than 10 differential volts. However, doing so drastically increases the voltage over the main bus line. The BC maintains a relatively constant differential voltage while transmitting and any deviation from that voltage results in an alert from our IDS. The nature of differential signalling makes this is a reliable metric, as any interference that would significantly alter the voltage of the bus will have little impact on the differential voltage. Our detection systems leave the attacker unable to transmit a command word undetected without taking over the bus controller.

If the attacker cannot initiate messages via a command word, then they can still affect the system by transmitting status or data words in response to the BC's command words. After a command word the the attacker responds before, during, or after the legitimate RT's response. As described in Section 6.2, the attacker generates extra command words (which are detected by the previously described detection system), generates a collision, or generates extra data words. The attacker cannot disable the legitimate RT to avoid collisions as the built in mechanism for doing this, mode codes, is detected by our command word spoofing detection. A collision raises an invalid word error due to: the sync or Manchester encoding being mangled, a parity error, or extra bits. Extra data words result in a word count error. These errors flags are generated on our receiver and thus the attacker has no way of preventing them.

The combination of these two security mechanisms, command word spoofing detection and collision and extra data detection, ensures that the attacker must take over the BC to transmit command words and must take over a specific RT to transmit status or data words as that RT. We do not detect malicious status or data words transmitted by a corrupted RT as long as they transmit in that corrupted RT's time

slot. The only remaining option for an attacker not in this position is pivoting from one system to another. Pivoting to the BC requires the attacker's RT to be addressed with an RT to BC message and pivoting from one RT to another requires an RT to RT message where the attacker is the transmitting RT. This provides a limitation on the attacker, as they must corrupt an RT that speaks to their intended pivot target. Furthermore, alerting on invalid word counts limits the attacker to a small payload for their any data exploit.

In summary, our BC detection system flags on any spoofed command words and spoofed status and data words are detected through standardized MIL-STD-1553 error flags. With this we achieve the goals outlined at the start of the paper - a detection system which rapidly alerts on attacks and can differentiate between legitimate rare messages, and attacker generated ones. Having full attack coverage for command words is a significant improvement on existing work, as it ensures an attacker cannot disable an RT or spoof any message from the BC without being detected. Additionally, removing the ability to disable an RT undetected allows us to safely rely on standardized error flags to detect spoofing attacks.

6.4 Experimental Results

In this section we describe our testbench, experiments, and results. Our results demonstrate the practical security benefits of this research and how our IDS can be rapidly integrated into a MIL-STD-1553 system. Additionally, we provide a comparison between our IDS and existing MIL-STD-1553 work.

6.4.1 Testbench

Our test bed consists of two MIL-STD-1553 devices connected via transformer coupled stubs to the same bus. The first is an alta computer, which is effectively an FPGA with a MIL-STD-1553 transceiver. The second is a Data Device Corporation (DDC) MIL-STD-1553 chapter 10 compliant computer which is running our analysis software. We use each device's API to transmit MIL-STD-1553 messages, we collect them with our detection system, and then we analyze the data using Python. For monitoring voltages, we use a single oscilloscope to maintain consistency in the resolution of measurements.

6.4.2 Experiment

We can prove the efficacy of the detection system by having multiple MIL-STD-1553 devices transmit command words. If our IDS only alerts when a non-bus controller is transmitting, then we can confirm our ability to detect when the bus controller is transmitting. For this experiment we use the two computers in our test bench. We use different manufacturers to ensure our detection mechanism works on multiple MIL-STD-1553 compliant devices. First, we have the DDC computer take the role of the bus controller. We connect an oscilloscope to the DDC computer's stub connection to monitor the voltage, then have it transmit a set of test messages. The alta computer then replays those test messages as our RT. We compare the voltage observed during the DDC computer's transmission to the voltage observed during the alta computer's transmission. We then swap our setup such that the alta computer is the BC and the DDC computer is replaying messages as an RT. This experiment is successful if our IDS can distinguish between the BC and the RT transmitting identical messages.

Our next experiment is intentionally introducing collisions and protocol errors onto the bus to determine what errors occur. We do this by having an RT attempt to transmit after our other computer starts transmitting a message. Additionally, we add data words to the end of messages to simulate an attacker tagging onto an existing message to deliver an exploit. The experiment is a success if we can reliably detect collisions and extra words with standardized error flags. For this experiment a 3rd party working with the US Airforce designed and executed the attacks while our detection system was installed into their testbench. The MIL-STD-1553 data going across the bus is modeled off of real MIL-STD-1553 systems, with randomized addresses, subaddresses and data to remove any national security concerns. The randomization is a mapping from one value to another and is consistent for each RT.

6.4.3 Results

For our bus controller detection experiment, we find that we can successfully differentiate between when the bus controller is transmitting and receiving data. Figure 6.3 demonstrates the results of our experiment with the alta device acting as the bus controller. When transmitting data, our stub connection registers greater than 10 differential volts, while it registers less than 4 differential volts when receiving data. We observe the same results when the DDC device acts as the bus controller and the alta device transmits data. These results give us confidence in the accuracy of our

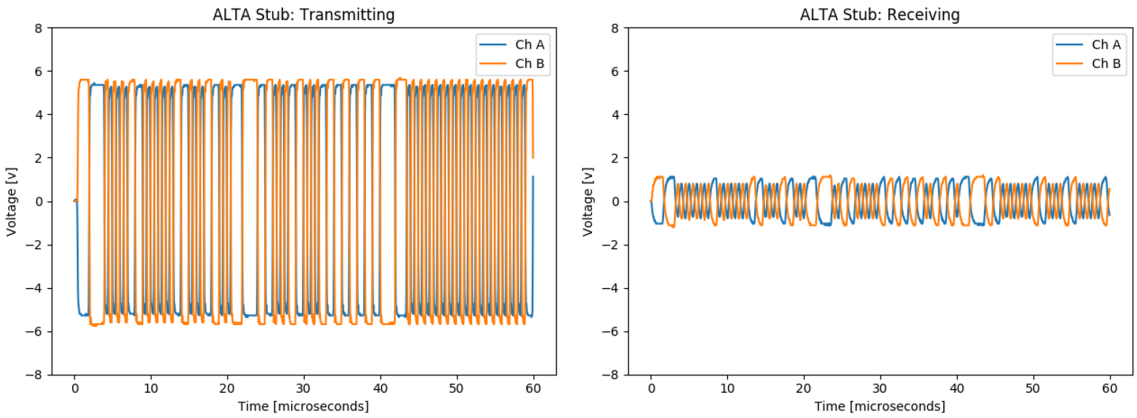


Figure 6.3: The results of our BC transmission detection experiment. These graphs show the voltage of each channel, depicting a clear divide in the magnitude of the peak to peak voltage observed by a transmitting and receiving device from the perspective of the bus controller stub.

detection system. Our IDS only needs the magnitude of the differential voltage on the BC stub connection and a way of monitoring for a command word sync waveform.

Our collision test is colliding with a message as it is being sent. Our IDS transceiver registers an invalid word error every time we have two devices transmit at once. Whenever a collision occurs, our standard transceiver logs the error on the next message and drops the message that was collided with. This form of error logging complicates incident response, as it is difficult to know what message the attacker decided to interrupt without using non-standard collection hardware. The error process also means an attacker continuously interrupting the bus can ensure that our transceiver never passes error information up to the IDS process. However, long periods of silence on an otherwise regular bus make this attack trivial to detect. For messages with data words appended to the end, our transceiver raises word count errors. This ensures an attacker cannot arbitrarily force an RT or BC to read extra data without being detected.

6.4.4 Comparison to Existing Work

There are a few main pieces of MIL-STD-1553 research: Orly [88], who combines RT fingerprinting and a model which predicts the order of MIL-STD-1553 messages, Onodueze, who does something similar but contrasts multiple machine learning techniques [63], and Generuex [29] who measures the response time of each RT to identify new devices on the bus. In terms of accuracy, Orly's solution and our IDS are the same for periodic messages, but their message sequence model does not apply to

aperiodic messages. Their RT fingerprinting solution has a false positive rate of 0.9%, making it difficult to use on a fighter jet. From a detection perspective, our command word spoofing detection detects more attacks than either solution by detecting all injected command words. The trade-off for this accuracy and attack coverage is that our IDS does not have the passive implant detection potential or the source attestation potential of Orly's research. However, better accuracy, full message coverage, and a simple implementation make our solution better suited to real world applications. Without full message coverage an attacker can transmit illegitimate mode codes or engage weapon systems with no risk of being detected. Additionally, our solution is not reliant on any particular implementation of MIL-STD-1553, making it applicable to any MIL-STD-1553 system rather than only MIL-STD-1553 systems from certain manufacturers. This is relevant for Orly's work which relies on a major/minor frame transmission scheme.

6.5 Preventing Illegitimate Command Words

An intrusion prevention system takes a detection system and adds the ability to cancel messages on the bus. In the case of serial data bus networks, a prevention system usually involves mangling the offending signal. Mangling a signal causes each RT to drop the targeted message [30]. In order for an intrusion prevention system to be effective it must be accurate and fast. An IPS must be accurate because preventing legitimate messages could cause safety issues, particularly on a moving aircraft. An IPS must be fast because it must process an attack, detect the attack, and then mangle the message. All of this must be done before the message is completed or else the system's computers will process the attacking message. The fastest message on MIL-STD-1553 is a mode code with no data which takes 20 microseconds.

Our BC fingerprinting technique is perfect for an intrusion prevention system. Our evaluation gives us confidence in the accuracy of our technique. In terms of speed, our IDS is capable of detecting an attack during the command word sync, which is the first 3 microseconds of a message. Our IPS is left with 17 microseconds to process this alert and create a collision which mangles the victim signal.

We are unable to provide prevention protections to collision based attacks, as an IPS would further mangle the message. This means that an attacker can perform a denial of service attack, but they cannot do so undetected. More targeted or nuanced attacks require transmitting a command word, which means that we can prevent these attacks. Preventing attacks has no long term consequences on the bus.

Another option for intrusion prevention in MIL-STD-1553 is using the shutdown mode code ourselves. This mode code would disable an RT until the entire system is reset or we use another mode code to restart the RT. While this approach could prevent future denial of service attacks, it requires a number of capabilities and assumptions to work. Our IDS would need a way to determine what device sent the attack, which typically requires full scale voltage fingerprinting. Otherwise, an attacker could spoof the address of one RT, send an obvious attack, and then our IPS would shutdown a completely legitimate RT. Additionally, the mode code approach assumes the attacker reads the bus and perfectly follows the protocol. We do not believe these are reasonable assumptions, particularly for implants and supply chain attack vectors. The consequences of being wrong are too risky, even for remote attackers.

6.6 Discussion

In this section we discuss: the impacts of denial of service attacks and how best to deal with them, the detection of malicious data words, and how to handle a corrupted bus controller.

6.6.1 Denial of Service Attacks

Denial of service attacks prevent all communication on the bus, either by holding the line high, or causing collisions with individual messages. This is exactly how our intrusion prevention system works. Because our IPS uses collisions to prevent messages on the bus, it is incapable of preventing collisions on the bus. These collisions are still detected, but colliding with a collision is pointless as the message is already dropped. The result is that a denial of service attack can hold the bus hostage.

To that point, it is important to understand what occurs when bus communication stops. If no bus communication is happening and no mechanical backups occur, then no changes to the system can be made. But the consequences of a dead bus are not defined in the standard. What happens, be it the vehicle turning off, an emergency protocol beginning, or the last state continuing until bus operation resumes, is implementation specific. That said, MIL-STD-1553 does have a redundant bus architecture which could allow us to gracefully recover from a denial of service attack, despite being unable to prevent it.

When MIL-STD-1553 was designed, the idea of a bus being dead was considered, but only for physical damage. While the existing redundant bus architecture is suitable for this, it is unsuitable for helping in the event of a denial of service attack or any

other cyber attack. A remote attack vector should affect both buses because each RT and BC has an interface to the main and redundant bus. However, small tweaks to how avionics computers handle their redundant interfaces can enable redundancy in the face of denial of service attacks.

The threat of these attacks could be mitigated by ensuring redundant interfaces do not process inputs until the bus controller indicates the redundant device is necessary. By not processing inputs, we disable the potential for both interfaces to be infected by the same attack. Instead, the attacker must replay their attack once the aircraft switches to the redundant device, at which point the aircraft may be out of range of the attack. If we assume the exploit allows us to process the attack message, we may even be able to pass along a unique attack payload to a temporary disallowlist for attacks not based on holding the line high.

This all assumes the attacker is unable to escalate privileges and is unable to take control of the processes that own each bus interface. We believe containerizing each process would enable our redundancy safeguards, while maintaining redundancy for the bus and resiliency for the process. Containerization adds another barrier to escalating privileges and escaping from one interface to the next.

6.6.2 Detecting Malicious Data Words

Detecting malicious data words relies on knowing when an RT sends malicious data. A contextual awareness for 'should the bomb bay doors be open?'. At a high level, this looks like domain specific rules, where a defender codifies the manual for the aircraft into an IDS. An interface control document may have this information, but these documents are often in difficult formats which make development extremely labor intensive. To make the problem worse, this information is unique to each system. Some existing work tries to get around the problem of proprietary data fields. Orly blindly analyzes data words for changes [88]. Pese pragmatically extracts common data fields from proprietary CAN buses [69]. Both of these approaches are helpful for detecting large deviations and identifying core system components, but they are insufficient for an attacker targeting less obvious RTs and state parameters. As external security researchers, our options for developing a security system are limited, but this is not to say our detection system is useless.

As we discussed in our security analysis, our command word spoofing detection and data word spoofing detection limits the scenarios where an attacker can send malicious data. These detection systems ensure an attacker trying to pivot must corrupt the RT that is asked to transmit a data word and have that RT speak to the

victim they are targeting. This security mechanism means the attacker needs either a BC to RT message to their corrupted RT, or an RT to RT message that allows their corrupted RT to transmit. Our ability to simply detect extra data words further restricts what payload their corrupted RT transmit as they attempt to pivot. Let us consider how our pivoting restrictions affect implant attackers, remote attackers, and supply chain attackers.

Implants do not start on a legitimate device, as such they cannot be addressed by the BC for an RT to RT message. An implant can never send an illegitimate data word with our IDS installed. For a remote attack vector, the attacker needs to have an exploit for a remote capable RT which also happens to speak directly to another important RT. Additionally, a remote attacker needs either: another exploit which lets them queue their attack for that RT to RT message or to get extremely lucky and happen to launch their remote attack just as the desired RT to RT message is transmitted. Supply chain attackers are affected by the same pivoting restriction. The RTs vulnerable to a supply chain attack may not be conducive to the attacker's desired attack. From this analysis we can conclude that our command word injection detection and data word spoofing detection strongly limits any attacker's ability to pivot on the bus.

After developing the techniques laid out in this chapter, malicious data word detection and physical modeling may still be desirable. But unless the defender has multiple purposes for this data analysis, such as predictive maintenance or incident response, then a different resource investment is probably advisable. It is far better to harden the subset of RTs capable of remote communication than model the entire aircraft to find anomalous data. Engineering resources spent hardening remote-capable resources are applicable to all aircraft and all future aircraft, while any data analysis is bespoke to a particular model of aircraft. This is not to say that security investments into unique systems are not worthwhile, but better to focus on the lower hanging fruit. This is especially true when focusing on that lower hanging fruit can prevent the difficult to detect attacks from happening.

6.6.3 A Corrupted Bus Controller

Our system assumes that the BC is not taken over by the attacker. In terms of realism, the BC is unlikely to host any typical remote communication functionality and so would not be the initial foothold. This means our pivoting detection is useful against remote attacks. As for supply chain attacks, one would hope the computer that is in control of a very expensive aircraft would be the most hardened part of

that aircraft, but it is difficult to generalize. That said, no existing MIL-STD-1553 research secures against a corrupted BC as they essentially control all of the data going over the bus. Timing based systems only work if the corrupted BC breaks from standard traffic patterns. Fingerprinting is ineffective since they are transmitting as that device. The BC dictates data to RTs which limits how effective any physical modeling based detection can work. Cryptography is ineffective since the corrupted device presumably has access to any secure keys to transmit data.

We believe the best way to handle this is to have our detection system on the BC. Our system is far simpler when implemented on the BC directly as the BC knows when it is or is not transmitting. Additionally, end point monitoring software on the BC itself would likely be effective at detecting any attackers on the system. Similarly to how internet of things research proposes monitoring CPU and memory usage to identify infected devices [92]. Avionics computers act regularly enough that it should be obvious when new activity is occurring on the device.

6.7 Conclusion

In this chapter we present a novel intrusion detection system tailor designed for MIL-STD-1553. Our detection system detects any illegitimate messages started on the bus, regardless of if the attacker is sending standard traffic, engaging weapon systems, or transmitting legitimate error messages at inappropriate times. We detect these attacks with 100% accuracy through a simple bus controller fingerprinting solution which is resistant to attacker activity or environmental factors. Additionally, we show that standardized MIL-STD-1553 receiver errors can be used to detect an attacker attempting to collide with or cancel traffic as it goes across the bus. The simplicity of these approaches allows any MIL-STD-1553 system maintainer to implement our IDS without complicated training or advanced capabilities. By designing our IDS with the MIL-STD-1553 protocol as the foundation, we create a more impactful detection system which can operate quickly enough and accurately enough to power an intrusion prevention system. Safe and easy to implement IPSes are desperately needed to protect the millions to billions of dollars represented by even a single MIL-STD-1553 system, much less the decades of aircraft and infrastructure relying on the protocol.

Chapter 7

An Evaluation Framework for Intrusion Prevention Systems on Serial Data Bus Networks

In Chapters 5 and 6 we propose intrusion detection systems for different serial data bus networks. The natural next step is an Intrusion Prevention System (IPS), which stops an attack after an IDS flags it. After all, if an IDS knows what an attack looks like then why allow the attack to complete? Industry and government maintainers of serial data bus networks want prevention capabilities to protect themselves from cyber attacks [81], [94], [53], but safely implementing an IPS is tricky. The IDS powering the IPS must meet certain criteria or the IPS risks transitioning from a security device to an attacker's accomplice. Even when an IPS is implemented safely, a clever attacker can still circumvent the IPS itself. The lack of nuance around calls for IPS technology leads to ambiguity in how exactly IPSes work and what their limitations are. IPS has become a catch-all term for a silver bullet solution to attacks on serial data bus networks. But grouping all intrusion prevention style capabilities into a single bucket oversimplifies what IPSes can do, leads to dangerous gaps in coverage, and results in unspoken assumptions about adversary capabilities.

The first thing to clarify is what exactly an IPS does, or at least, what an IPS is meant to do. IPS can either imply cancelling out an individual message or outright turning off the attacker's device. We separate these functions because their technological requirements and assumptions about attacker capabilities are completely different. Cancelling out an individual message is theoretically simple. If the IPS mangles a message, then the rest of the network drops it and the attack is stopped. But this implies the IPS, and by extension the IDS, can act fast enough to mangle a message before it is accepted by the rest of the network. Speed often comes at the

cost of complexity, which may mean an IPS is not suitable for some threat models because the IDS powering it is computationally limited. On the other hand, turning off an attacker's device has nothing to do with speed and everything to do with what the attacker can do. The assumption that an attacker will kindly listen to a shutdown message, accept a firmware upload, or follow some protocol-specific error process is often laughable for advanced attackers. But limited attackers do exist for many threat models. Of course, this all assumes that the attacker-disabling IPS knows where to aim. Spoofing one's source address is extremely common attack behavior. An attacker-disabling IPS requires a source attestation capability to avoid turning off whichever perfectly legitimate device the attacker decides to spoof. A defender can assume that an attacker is unable to spoof their origin, but this needs to be factored into how a defender views their system. The reality is that no IPS or IDS can do everything; all technical capabilities come at a cost. We require a way to evaluate the viability and efficacy of these tools such that a defender can allocate their resources according to their unique individual needs. A defender should be able to look at the breadth of separate IDS and IPS technologies, determine which combination secures against their threat model, and then evaluate how effective their implementation actually is at detecting and preventing attacks.

To enable a defender to do just that, we propose an evaluation framework that works on three axes: the IPS's viability for different attack vectors, the IDS's suitability to IPS for different attacks, and the IDS's efficacy at detecting those same attacks. We choose these axes as they represent the core of the IPS, the intersection of IDS and IPS, and the core of the IDS. If any of these axes fail, then the entire security system fails. We break our axes down by attack vector and by different attacks. This allows a defender to understand how an adversary can evade their security system. For example, an IDS may be fantastic at detecting spoofing attacks, but be completely blind to an attacker manipulating the voltage on the line. There is nothing inherently wrong with limitations, as long as they are known quantities. To this end, our evaluation framework provides a score for each attack vector and attack type, resulting in 8 scores across our three axes. These scores can be used by a defender to compare and contrast different security systems. While a single combined score can make for simple comparisons, we believe it obscures too much information for any practical use. But too strenuous of an evaluation system is equally impractical. To this end, we built an open-source, protocol-agnostic tool, which helps an evaluator test their IDS by randomly adding attacks to a dataset [74].

In this chapter we provide case studies for the CAN and MIL-STD-1553 serial data bus protocols. These demonstrate that after using our evaluation framework an evaluator will know: what attack vectors their IPS is secure against, what aspects of their IDS can safely power their IPS, and how well their IDS can detect different attacks. We believe that our evaluation framework has the potential to turn IPS for serial data bus networks from a desirable but terrifying concept, to a real tool that can be safely used in the right scenarios. We summarize our contributions as follows:

- An evaluation framework which scores IDS/IPS tools based on their efficacy and viability for different attack vectors and attack types.
- A new categorization strategy for prevention systems.
- A protocol-agnostic and open-source attack emulation tool built to simulate attacks on serial data bus network datasets.

7.1 Existing Research

The most direct comparison to this work is other evaluations of existing IDS technologies. While there are papers that compare different solutions, these largely examine them as standalone solutions rather than integratable components for a larger security goal [101], [3], [48]. We are unaware of any comparisons or analyses of IPS technologies. Outside of serial data bus networks, network IPS is a cleaner problem with core routing infrastructure acting as an adversary-in-the-middle device. The distinctions we draw between stopping an attack and stopping attackers from speaking are still relevant, though the safety considerations built into vehicle networks often provide more potential IPS mechanisms [11] [75].

For the purpose of our evaluation framework, we are assuming that a defender has a large list of detection systems and a large list of prevention systems. When deciding on what security system to deploy to a serial data bus network, a defender plugs the prevention system into the detection system and hopes it works. Our evaluation framework allows a defender to know what scenarios their IDS/IPS combination works in and how well it works. This section presents a list of prevention and detection systems that can be matched up.

7.1.1 Existing Prevention Systems

IPS technology on serial data bus networks is poorly researched in comparison to IDS technology. Likely because once one IPS solution works for a protocol, there is a limited amount of novel research that can be done without integrating a complicated IDS. The complicated collection setup required to get IDS systems to operate before a message is finished transmitting results in few integrated IDS/IPS systems.

Let us first discuss prevention systems which are connected to the serial data bus network as if they were an additional device, which we will refer to as a Line Replacable Unit (LRU). One of the first popular IPS proposals is CANStomper [30], which intentionally flips a bit in the last 11 bits of the CAN bus to trigger a CAN error frame, causing the victim message to be ignored by the rest of the bus. CANStomper triggers this prevention system using an allowlist-based detection system which alerts whenever a new ID appears on the bus. The complexity of the IDS aside, this work demonstrates that intrusion prevention on individual messages is possible with cheap FPGAs.

On the other end of the IPS spectrum is the idea of targeting not individual attack messages, but the attacker's device. Takada [90] does this in a similar method to CANStomper but instead of monitoring the line for an attack and interrupting it, this IPS copies the attacker message, transmits at the same time as it, then changes one bit to cause an error. The CAN bus then automatically replays both messages, resulting in another error. This loop repeats until the attacker's message changes, or the attacker reaches a 'bus-off' state which disables their transmitter. The benefit of this approach is that it prevents the attacker from sending more attacks, but it does introduce a limitation on the attacker. CAN devices have a mechanism to force themselves out of the bus-off state. Takada assumes the attacker did not think to account for this state or is somehow restricted such that they cannot exit the bus-off state.

The previous solutions are designed around the idea of a single device connected to the bus, as most intrusion detection research follows this model. The other approach is in-line prevention systems. In-Line systems intercept messages as they are communicated and drop them if they are anomalous [54], [40]. While this approach can be extremely effective, it comes with higher integration costs as one of these computers needs to be put in-line for every LRU. Given the increasing numbers of LRUs in vehicles [24], this approach does not scale well. In general, in-line detection systems are a poorly researched area. For future research we suggest focusing less on connections between the LRU and the bus and more on in-line connections on

the bus. In-line devices communicating with each other provide the potential for a novel security guarantee, even in situations where a security system can be a software upgrade to each LRU.

7.1.2 Existing Detection Systems

We observe four distinguishable types of detection systems for serial data bus protocols. These are timing-based, voltage-based, data-based, and protocol-based. We believe cryptography-based solutions [33], [19], [70], [58] are outside the scope of IDS/IPS, in part because IDSes still provide a useful layer of security even with a fully cryptographically authenticated bus.

Timing-based systems generally rely on the timing between messages [86] [12], acting on the assumption that an attacker spoofing a legitimate device will lead to an anomaly in the time interval between messages. We find that straight interval approaches like those used by Song [86] are often too limited to messages with tight, consistent transmission patterns. This means that an attacker can bypass timing intervals by selecting messages with varied timing intervals. This research can be expanded through greater contextual awareness for why timing intervals change. Higher level timing research like Cho's clock skew analysis [12] and Genereux's response time analysis [29] provide the detection system with fingerprinting capabilities. However, the greater complexity of the analysis makes it spoofable by an attacker, as shown by Sagong emulating clock skew [77]. It is plausible that an advanced attacker is equally capable of emulating Genereux's response time by predicting what message will be requested and preemptively starting to respond. We include state machine based solutions in the timing category as they try to predict the time slots a message could appear in [103], [88], [9], [68]. Orly [88] demonstrates this by predicting the order of MIL-STD-1553 messages and alerting on any deviations.

Data-based systems assume some level of correlation between the engineering values transmitted by different LRUs. In aggregate these correlations create a model of the system. If an attacker transmits a message that somehow breaks the invariance of the overall system, then an anomaly is detected [98], [72]. This approach is good at detecting unexpected deviations in engineering values, but can be defeated by an attacker who corrupts an important LRU, or an attacker slowly creating small deviations.

Voltage-based systems fingerprint each LRU and search for unseen fingerprints that indicate a new device is transmitting on the system [14], [13], [41], [27], [5], [76], [88]. These systems are good at detecting when one device pretends to be another, but have

weak security properties when an existing device is taken over by an attacker. The main problem with voltage based approaches is that they require consistent retraining to account for environmental impacts on vehicle circuity. This retraining provides an easy attack window for persistent attackers, like implants. Corrupted ECUs can completely bypass voltage fingerprinting as long as they only transmit messages their corrupted device would transmit.

Protocol-based detection systems are usually limited in scope, being focused entirely on built in error processes and attacks specific to a protocol, i.e., CAN and MIL-STD-1553. These are often proposed in the countermeasures sections of papers proposing some new attack [11], [55], [44]. The nature of these attacks makes them difficult to detect with any of the aforementioned generalist techniques. As a result, protocol-specific detection systems are an inevitable requirement of any security system. The general tact of these systems can be created before attacks papers are released by researching any protocol-specific assumptions, error states, or networking logic. Often flagging on built in error codes can do most of the work, but this assumes a system is relatively error free. A common theme with protocol-based detection systems is that the more errors naturally occurring on the system, the more difficult it is to separate the attack signal from the noise.

In terms of detecting attacks - timing and voltage are generally good at spoofing attacks but bad at detecting an attacker who has taken over an existing ECU and is transmitting messages the victim ECU would normally send at the right time. There is no reason to believe only advanced attackers can transmit in the appropriate time intervals, making voltage and timing based detection systems have weak security guarantees against corrupted LRUs. Data-based detection systems have strong security guarantees against attackers sending malicious data across the bus, but the threshold for what qualifies as malicious is variable and may need tuning from vehicle to vehicle to avoid false positives. The security guarantee of data-based systems against corrupted LRUs is significantly weaker if the attacker is in control from the moment the vehicle is started, as then there are no significant deviations and the corrupted LRU is the source of truth for some data. This allows an attacker to transmit whatever they wish as long as they only transmit malicious messages that are not a function of other bus traffic. More research is necessary to address this persistent corrupted attacker. Protocol based detection systems offer no rigid security guarantee.

7.2 Adversary Model

In this chapter we define two adversaries, both with the goal of transmitting, modifying, or cancelling messages on a serial data bus network. The first is a strong attacker. A strong attacker is in full control of their attack vector and everything on it; including the firmware and software running on the device, as well as which messages the attacker chooses to process. We assume stronger attackers have persistence on their attack vector and cannot be reliably erased. In contrast, weak attackers have limited control over their attack vector, and are restricted to simply transmitting and receiving messages, with no control beyond which messages are sent when.

The difference between these attackers is best illustrated through different attack vectors. Take an implant, a new LRU added to the network capable of transmitting messages. The adversary designed this implant and as such we can assume it is running custom hardware or modified firmware such that it only follows the protocol when it is advantageous for the attacker. But what about other attack vectors? A remote attacker's capabilities vary dramatically based on their exploit. A weak remote attacker may only be able to send a single message payload, while a strong remote attacker may take complete control of that LRU with some level of persistence. The same is true for supply chain attacks, though we consider this attack vector more through the lens of adaptability. A weak supply chain attack could be a time bomb attack, triggered after some condition is met, which does one specific payload, while a strong attacker could perpetually control the vehicle, adjusting their attacks on the fly based on what the rest of the network says.

Through strong and weak attackers we can properly evaluate how effective and resilient an IPS is. For example, if the IPS requires transmitting a complete message on the bus to disable an attacker, then the IPS would work on a weak attacker but not a strong one. This informs how we evaluate prevention systems, ensuring that the chosen IPS is effective against whichever attacker the defender expects to face.

7.3 Types of Prevention Systems

We put intrusion prevention systems into two categories: cancelling individual attack messages and preventing an attacker from sending further messages. We refer to these as Message Intrusion Prevention System (MIPS) and LRU Intrusion Prevention System (LIPS) respectively. In this section we evaluate the pros, cons, and feasibility of MIPS and LIPS.

7.3.1 MIPS

MIPS works by cancelling out a message as it transmits across the bus. It does this by exploiting a common error process on serial data bus protocols where a message with invalid encoding is dropped by the controller before it is processed as a 'message' by the victim LRU. MIPS intentionally mangles the signal of an attack by briefly driving the voltage of the bus in the opposite direction. The main benefit of this approach is that MIPs technologies are not concerned with where the attack is from. By avoiding the source attestation problem, MIPS works even if the attack vector is spoofing another device. MIPS has two primary limitations.

The first limitation is based on timing. In order for MIPS to work, it has to mangle a message before it finishes transmitting. If the attack is only detectable in the last few bits of a message, then MIPS has limited time to execute. MIPS inherently favors detection techniques which make it clear that an attack is occurring near the beginning of the message, this way time-to-detect is less of an issue. We will expand on these detection techniques in Section 7.5.

The second limitation is that MIPS only stops a single attack, without addressing the root cause of how that attack was transmitted. The fact that an individual attack was prevented is undeniably beneficial, but at the end of the day MIPS's technique is akin to a denial of service attack. If an attacker rapidly transmits attacks on the bus, then MIPS is simply performing a denial of service attack on the bus. We believe this is better than whatever the attacker's payload would have done, but only if the defender knows what the system does when it stops receiving valid messages. Most protocols do not specify this behavior and instead leave it up to the manufacturers, making it vary from system to system. The knowledge of what happens when the system stops receiving messages makes the effect of the MIPS denial of service attack predictable, which is always better than an unpredictable attack. If the denial of service attack is not a flood of messages, but instead an attacker mangling all messages on the bus, then MIPS is ineffective as it cannot correct mangled signals.

Based on these limitations, we make two conclusions. MIPS is best against attackers with limited attack windows - otherwise MIPS is akin to a denial of service attack, albeit an intentional one, and MIPS requires early detection metrics to be reliable at stopping attacks. Requiring early detection metrics usually implies that data based detection techniques are sub-optimal for prevention, as the data payload is often near the end of a serial data bus packet.

7.3.2 LIPS

LRU intrusion prevention is stopping the attacker from executing any future attacks after the first attack. LIPS is the ideal silver bullet solution for protecting serial data bus networks. Unsurprisingly, LRU prevention systems only work in a limited set of scenarios and can be avoided entirely by some attackers.

LIPS relies on the idea that there is some way of disabling the transmitter of another system. That there exists a message or sequence of events that causes the victim to turn off or be disconnected from the system. In our experience, a built in off-switch exists in many serial data bus protocols. How exactly it works varies from serial data bus protocol to serial data bus protocol. We elaborate on different protocol off-switches in Section 7.8. We can generally split LIPS techniques into four categories: firmware uploads, protocol-specific 'turn off' messages, device disconnection, and error manipulation.

Firmware uploads as an IPS technique are based on the idea that an attack can be halted by uploading known good firmware to a victim LRU. The technology to perform this firmware upload on the fly, literally, was demonstrated by the U2-Dragonfly [95]. It is unlikely most systems will be able to commence a firmware upload without stopping the vehicle first, making firmware uploads a slow prevention system, but it is worth noting as a viable option.

Protocol-specific turn off messages are exactly that - a message in the serial data bus protocol that tells an LRU to disable itself. These are more likely to occur in bus controller based protocols, where the bus controller has complete control over the flow of traffic on the bus. In Section 7.8 we discuss examples of this technique in the MIL-STD-1553 protocol.

Disconnection is the idea of physically disconnecting LRUs from the network. We are unaware of cases where this is built into the protocol, though in-line prevention systems [40] have integrated it into existing systems. Adding the capability to disconnect LRUs from a system is dangerous, as it means an attacker can now also disconnect LRUs from the bus.

The final technique is error manipulation. Many serial data bus systems have a built in error process to ensure the safety of the vehicle should a single LRU malfunction. By intentionally producing errors in a target LRU, the LIPS can make it believe it is in an erroneous state, such that it disables itself. This LIPS technique is often the result of many MIPS triggers in a row. But the idea of error states and targeted messages raise an important question for LIPS, will an attacker actually listen when it is told to turn off?

LIPS as a technology fundamentally requires the attacker to have limited control over their attack vector or to be ignorant of the LIPS. This does not mean that LIPS is impractical, it raises the minimum technical competency of the attacker. Much like how a traditional malware author must now bypass stack canaries and ASLR, the serial data bus attacker must now gain further control over their victim. Specifically, they must gain control over the 'controller', the chip which receives messages from the bus and enforces the protocol. If they can selectively disable their receiver or ensure the controller ignores certain messages, such as firmware messages, then they can bypass the LIPS. But control over the controller can require rewriting the firmware of the controller. An attacker gaining enough control over a victim to transmit a message does not necessarily have the ability to rewrite the firmware of the controller.

7.4 Attack Types

To create an effective IDS and IPS evaluation framework, we need attacks to evaluate the security system against. But selecting a subset of attacks will inevitably lead to gaps in our evaluation. Instead, we need to base our evaluation on higher level ways of interacting with the bus. Let us examine what actions are possible on a serial data bus network, and which of those we would consider an attack.

The first choice an LRU makes is whether to speak when the bus is free or when the bus is busy. If the bus is busy, then they are speaking while another message is being transmitted on the bus. At this point two outcomes can occur: either our LRU's transmission collides with the active message, mangling it such that the message is considered invalid and cancelled, or our LRU's transmission collides with the active message such that the message is still valid and the message is manipulated by our LRU speaking. Said differently, if the bus is busy then an LRU can either cancel out an existing message or manipulate it to be different. A legitimate LRU would never intentionally collide with another message, meaning that both of these options are potential attack paths.

Now let us assume the bus is free, meaning that our LRU can transmit a normal message. When transmitting on the bus, the first thing an LRU decides is what type of message it is going to send. An LRU can either send a message as itself or it can send a message as someone else, i.e., the engine can send a message as the engine or as the transmission. We refer to an LRU pretending to be someone else as spoofing. This is not a choice for most LRUs, but imagine an implant or a corrupted LRU. An implant is a new device connected to the network, it cannot help but pretend to be

something else. A corrupted LRU is the aforementioned scenario where our engine can choose to pretend to be the transmission. But let us say that an LRU is claiming to be the device it actually is. Now it is transmitting a message on the bus and all that is left to determine is if the message itself is malicious. More specifically, we need to identify if the data contained within the message is malicious. If it is malicious, then we refer to it as corrupt, otherwise our LRU is transmitting a normal message on the bus.

Figure 7.1 presents the options an actor can take when transmitting as a tree. On the top branch of this tree we can see a clear normal path, where an LRU transmits when the bus is free, transmits as themselves, and transmits normal data. All other paths are attacks which we refer to as corruption, spoofing, manipulation, and cancelling. It is the role of an intrusion detection system to differentiate the normal messages from the attacks.

Our four attacks are representative of how an attacker must interact with the serial data bus to execute their attack. These four attacks apply to both the strong and weak adversaries depicted in our adversary model. Strong adversaries have greater control over attack execution but the attacks are the same. Evaluating detection systems against high level attacker actions allows us to robustly test the security guarantees of IDSes. If an IDS is alerting on lower level attacks, then an attacker can still change their behavior to execute an attack on the system. But if an IDS is basing their detection on how an attacker must interact with the system, then even a strong attacker cannot adapt to avoid detection.

Notably our attacks are all based on what an attacker must do on a message by message basis. There is likely a way to consider attacks in a much broader context based on their consequences rather than their prerequisites to action, but the message by message context fits better for an evaluation system which includes IPS. For the purposes of an IDS/IPS evaluation framework we define our four attacks as follows.

Spoofing is when one device/LRU pretends to be a different device/LRU. To spoof another LRU, an attacker must transmit a new message onto the bus. High level detection for spoofing usually involves the artifacts from an attacker transmitting deviating from the artifacts of a legitimate device transmitting.

Corruption is when an attacker takes over another LRU and sends malicious data. Corrupting an ECU implies transmitting as it, meaning messages become malicious without any deviation from normal transmission patterns. The typical assumption for corrupted attackers is that their transmission artifacts are indistinguishable from the victim LRU. As a result, detection is based on the behavior of the overall system.

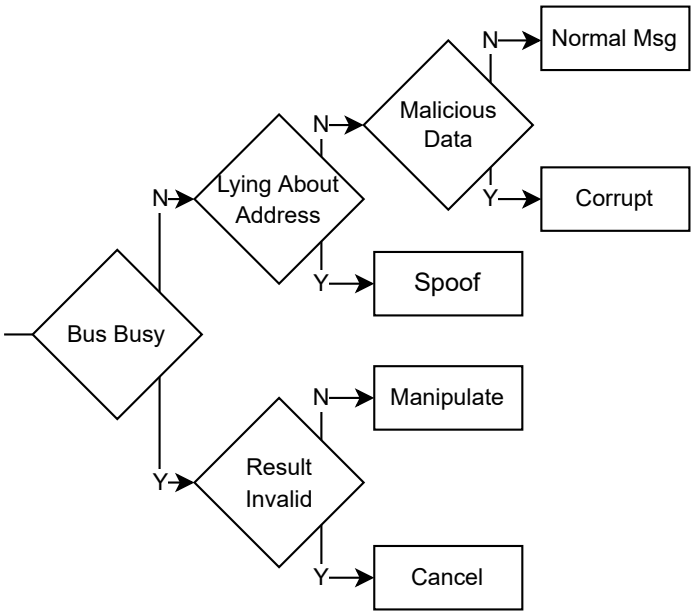


Figure 7.1: A tree of the options one can take when wanting to interact with a serial data bus network. Spoof, Manipulate, Cancel, and Corrupt are attack paths.

Manipulation is when an attacker modifies bits as they are transmitted on the line. To manipulate bits an attacker must override the physical layer signal. It is not a corruption attack, as they do not actually control the LRU, but they are capable of changing what the LRU says. Because of this, detection is usually based around the protocol level side effects of flipping bits [11], or voltage-based detection systems [14].

Cancelling is when an attacker interrupts some number of messages on the bus with the goal of stopping those messages from appearing on the system. Cancel attacks often make good prevention systems when used by a defender, as the principle of a cancel attack is the same principal used by MIPS. This attack is easy for a defender with standard receiver hardware to miss, as a standard receiver might drop any victim messages. Detection for this attack is normally protocol-specific, timing-based, or voltage-based [44], [86], [11]. Detection is: protocol-specific by the consequences of the protocol for cancelling a message, timing-based by the absence of a message, and voltage-based by the typical physical layer artifacts of mangling a signal. A MIPS could theoretically fix a cancel attack victim, but we are unaware of any research which does so.

7.4.1 Subsets of Attacks

Depending on the nature of the IDS one is evaluating, our four attacks may be too broad to be useful. Perhaps an IDS is designed to detect substantial increases in bus traffic, but it is not designed to detect every instance of spoofing. Alternatively, the threat model could not classify all types of spoofing attacks as attacks. An evaluator can selectively expand the list of attacks based on what they find valuable. We provide a list of subattacks below as examples.

Flooding is a subset of spoofing where an attacker transmits a large number of messages onto the bus, such that LRUs are unable to find a slot to transmit normal bus traffic. Flooding is effectively a denial of service attack done via a large number of spoofing attacks.

Modification is a subset of corruption and spoofing attacks and is when an attacker modifies the data field of a message to influence the system. This subset illustrates how to grade detection systems which are based on the payload of attacks, such as Wasicek [98] and Quinonez's [72] detection systems. Not every message an attacker sends is an attack, but if the payload exceeds a certain threshold or risk metric then it is alerted on.

Reflexive is a subset of corruption and spoofing attacks and is when an attacker replays a message on the bus. Reflexive attacks assume that any messages an attacker sends are inherently malicious and worth alerting on.

7.5 Detection for the Purpose of Prevention

All prevention systems require the existence of a detection system to alert the prevention system to fire, but not all detection systems are suitable for IPS. For a detection system to be effective for serial data bus intrusion prevention it must detect attacks early and be highly accurate. While accuracy is important for any prevention system, the importance varies between MIPS and LIPS, as LIPS has lasting consequences while a single MIPS misfire may have no noticeable effect on the vehicle. In this section we break down what existing detection mechanisms are best suited to message or LRU based prevention systems.

7.5.1 MIPS Detection Systems

MIPS requires an attack to be detected by the end of the message. Otherwise, the MIPS has no time to mangle the attacker's message and prevent it from being

processed by other LRUs. Phrased differently, after the detection system triggers, after it determines a message is an anomaly, and after the IPS fires, there must be some percentage of the message that has not been transmitted. Some part of the message must be available for the MIPS to interrupt. We refer to the measure of how much of a message is taken by the IDS/IPS as speed.

There are three main components of speed: the minimum number of bits required for the IDS to make a decision, the processing time of the IDS, and the time to signal the IPS. The minimum number of bits is based on what data the chosen detection system needs, while the processing time is implementation specific. It can be tempting to dismiss any implementation specific metrics, but we must remember that almost all existing detection systems were not designed with intrusion prevention in mind. Very few detection systems operate quickly enough to work for MIPS, and this will continue until performance is a meaningful metric detection systems are judged by.

In terms of the minimum number of bits, this varies for different detection techniques. A voltage fingerprinting technique requires a few bits of a single device transmitting. For completely new devices that may be all that is required, but for corrupted existing devices the IDS also needs to observe the unique ID of the message. That unique ID is compared against the voltage fingerprint to see if they match. A timing-based technique requires the unique ID to be transmitted so that it knows what message or LRU to calculate times for. A data-based technique requires the data field to be transmitted. Generally the unique ID of the message is sent as a header, with the data being later in the message. The implication being that data-based techniques require faster IDS processing than voltage or timing-based solutions to be viable. From a processing time and complexity point of view, timing-based systems are likely the fastest on average due to their inherent simplicity and the low number of data points required per decision.

Protocol-specific detection systems, like those which alert on built in error flags, can often make good prevention systems because detection is a simple condition check. However, preventing errors risks legitimate errors being erased from the bus. A method for reducing false positives for protocol-specific IDSes is using multiple detection systems to monitor for two compatible alerts. However, having multiple IDSes does come at the cost of increased overall detection time.

7.5.2 LIPS Detection Systems

The most important factor for LIPS is verifying that the LIPS activating will disable at best the attacker's device and at worst no devices, with no scenario where a legitimate

device is disabled by the security system. LIPS techniques usually require source attestation capabilities to verify who actually sent the attack. Source attestation can be provided in a number of ways [12], [14], but it is primarily done with voltage fingerprinting.

Unlike a MIPS, a LIPS has no specific speed requirements. A LIPS aims to prevent future attacks from happening, so as long as the LIPS disables the attacker before it can transmit multiple messages the LIPS succeeds. How many messages an attacker can transmit before the consequences render LIPS pointless is difficult to quantify and varies from protocol to protocol. Generally, the only speed goal for LIPS is to have no latency. That is, by the time a message is received and processed the next one should still be transmitting on the bus. Otherwise, if an attack occurs hours into bus traffic, then the attack may not be processed immediately by the security system.

7.6 Evaluation Framework

So far this chapter has broken down different types of prevention systems and the types of detection systems which are suitable to prevention. While we can talk through these problems on a case-by-case basis, we lack a meaningful way to compare and contrast different systems. A defender needs to be able to select an IDS and an IPS and know how well the entire system will work, as well as how well the system will work compared to other IDS, IPS combinations. To solve these problems, we propose an evaluation framework which measures the viability and efficacy of prevention and detection systems as they are faced with different threat models. A standardized evaluation framework highlights what scenarios the security system is designed to function in, while equally shining a light on any limitations or assumptions designed into the system. The result is a necessary clarity, which allows defenders to understand the attacks and attack vectors their system defends against.

Our evaluation framework asks three questions:

- How viable is an IPS for our threat model?
- How suitable is our IDS to powering an IPS?
- How effective is our detection system against spoofing, corruption, manipulation, and cancelling attacks?

These questions cover the three pillars of a security system: the prevention system, the detection system, and the integration of the two. The prevention system is graded

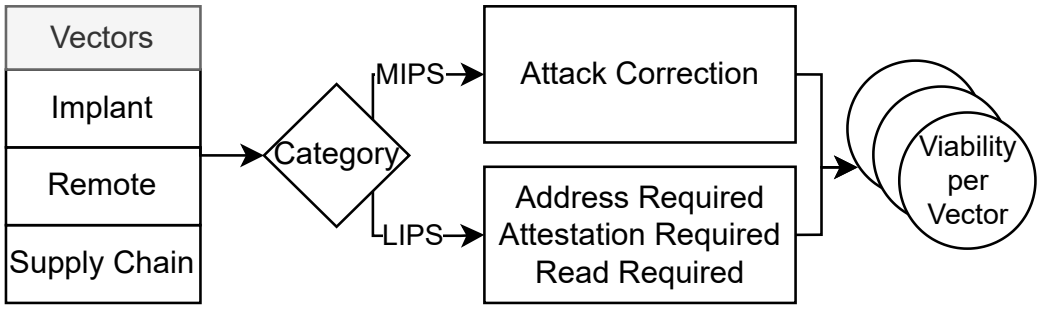


Figure 7.2: A depiction of our evaluation framework for IPS validity. The three attack vectors are evaluated against either MIPS or LIPS criteria, producing a valid/invalid result for each attack vector.

against attack vectors to allow the defender to see what scenarios the prevention system functions in. Figure 7.2 depicts the attack vectors and criteria used to determine the viability of the IPS. Both the detection system and the IDS/IPS integration are graded against the four attacks described in Section 7.4 to identify any gaps in the detection system. In addition to the standard gaps in coverage an IDS may face, our evaluation framework is focused on identifying any gaps which result in the detection system being unsuitable to powering an IPS. Figure 7.3 depicts our grading process, as well as the criteria used to evaluate IPS suitability and IDS efficacy. In Figure 7.3 each attack is added to a trace from a serial data bus network, which then feeds the evaluation framework and IDS. The evaluation framework outputs our criteria which are then are fed into the appropriate suitability and efficacy buckets to produce a score for each attack.

An important note for our evaluation framework is that we assume an accurate IPS is better than allowing the attacker's message to be delivered. The rationale behind this is that the ramification of an attacker's payload, and more importantly the length of the attack, are difficult to rapidly predict compared to the known effects of an IPS triggering, assuming the vehicle maintains some level of functionality. A defender with the appropriate cyber threat intelligence may be able to make this determination that but it is outside the scope of this chapter. The result is that our framework draws no distinction between attacks regardless of length or scale.

7.6.1 Prevention Evaluation - Viability

For evaluating prevention systems, we are less concerned with scoring the efficiency of the system and more concerned with the viability of the system. That is, what scenarios the IPS actually functions in. The reason for this focus is that in the overall

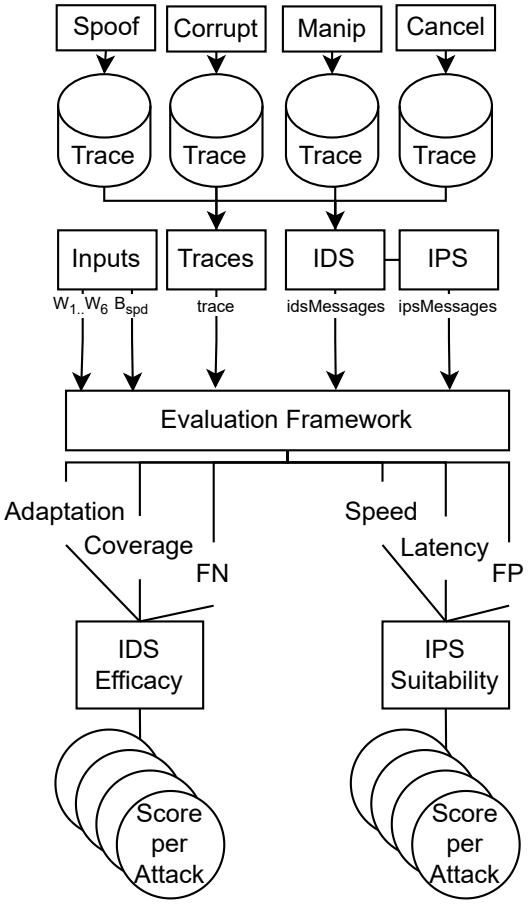


Figure 7.3: A depiction of how our evaluation framework is tested for IDS efficacy and IPS suitability, where the evaluator provides weights and the speed of the bus. Attacks are added to each trace, which are then sent into the evaluation framework and the IDS to produce a score for each attack. Algorithm 4 and Algorithm 5 describe how the six criteria are calculated.

IDS/IPS system the IDS makes all of the security decisions and the IPS does whatever it is told. An IPS could always be faster or more efficient, but we assume that if an IPS is being evaluated by this framework then it functions as a prevention system. The security challenge is knowing if the IPS can be circumvented by an attacker in certain situations. To solve this challenge a defender needs to understand what assumptions are baked into their IPS. To that end, we do not produce a score for our IPS evaluation. Instead, we report either true or false for if the IPS is viable for each of our attack vectors. These attacks vectors (implants, remote attackers, and supply chain attackers) represent: a new device with custom hardware, an unchanged existing device, and an existing device with modified hardware respectively. We list our evaluation criteria below. If any of our criteria are unviable, then the entire IPS

is unviable against that attack vector. This evaluation strategy makes explicit how the assumptions of an IPS match up against the different threat models a defender may have. Notably our criteria are split between MIPS and LIPS, as these prevention systems have different requirements for success.

Attack Correction (MIPS) is whether or not the attacker can unmangle the prevention signal. The pretense behind this is that an attacker could learn how a MIPS mangles a message, then counteract it. This capability requires additional hardware and advanced capabilities, but it would invalidate some implementations of MIPS if attack correction is in the defender's threat model. A MIPS is viable if it can counteract attack correction or attack correction is not in the defender's threat model.

Address Required (LIPS) is the question of whether or not the LIPS needs to be able to talk to a specific address. In other words, if the LIPS functions by telling source address 2 to turn off, does it work against an attacker without an address? This criterion is most likely to affect implant attackers, as they do not need to claim an address on the bus. A LIPS is viable if the IPS does not rely on an address to function.

Attestation Required (LIPS) is the determination of whether the LIPS needs a way to verify which LRU sent an attack. This criterion is not relevant to MIPS, as MIPS is unconcerned with targeting the attacker. A LIPS is viable if it: does not require attestation to function, has attestation capabilities, or the attacker is unable to spoof traffic.

Read Required (LIPS) is the determination of whether the attacker needs to be able to receive data for the LIPS to function. This criterion is not relevant to MIPS, as MIPS invalidates attacker messages without interacting with the attacker. A LIPS is viable if it functions despite an attacker selectively reading the bus or the threat model assumes the attacker must read the bus.

7.6.2 Detection Evaluation - Prevention Suitability

The criteria below focus on how suitable a chosen IDS is to powering a chosen IPS - be it MIPS or LIPS. After all, an IDS may be very good at detecting attacks but be too slow or too prone to false positives to be viable as an IPS. This is similar to our previous concept of IPS viability, but while IPS viability is capabilities based and centered around threat models, IPS suitability is based around attacks and the measurable differences in the way an IDS handles each of our four attacks. To this end, IPS suitability produces 4 scores which indicate how safe and viable an IDS/IPS combo is for spoofing, corruption, manipulation, and cancelling attacks. One way

to use this metric is to selectively enable IPS for different attack types, this helps a defender ensure that they only risk running the IPS if they are confident in the speed and accuracy of the IDS.

Speed (MIPS) is representative of how much of a message is transmitted before the IDS makes a determination and the IPS fires. Speed is used to determine whether or not there is time to mangle a message and prevent an attack. We calculate speed as the percentage of the message taken by the IPS plus the percent of the message taken by the IDS divided by the total time taken to transmit the message. If this number is greater than one, then the message is complete and MIPS is impossible, so it receives a score of 0. Anything less than or equal to 1 receives a score of 100, as it does not matter how fast a security system is as long as message prevention is possible.

Latency (LIPS) is the LIPS side of the coin for speed. If the IDS is too slow then the entire security system could start to build a backlog of messages. The consequence is that our IDS will start to register attacks well after they are transmitted. A delayed LIPS could still disable an attacker, but there is a significant distinction between a LIPS which stops an attacker after one attack, and a LIPS which disables an attacker several minutes or hours after the first attack.

Our scoring for latency reflects this difference by calculating how long the detection system takes relative to the time it takes to transmit a complete message. The score is the time taken by the detection system, divided by the transmission time multiplied by one plus the percentage of dead bus time. For example, if we assume a message takes 500 microseconds to transmit with an average bus utilization of 70% and our detection system takes 750 microseconds, then we get 1.15. This indicates that after we turn the vehicle off we would expect the IDS to keep processing messages for 15% of the time it collected data. Any number greater than 1 receives a score of 0. Any score less than 1 is a 100 as the total latency is irrelevant as long as the bus does not fall behind.

False Positives are whenever the IDS alerts on a benign message. A prevention system essentially acts as a DoS attack when presented with a false positive. It follows that installing an IPS with high false positives makes the system worse off than one without a security system. At least a system with no IPS has a chance that it will never be targeted by an attacker. Our evaluation framework aggressively weights false positives to discourage an IPS from negatively impacting a system. The score is one minus the percentage of false positives multiplied by ten million. For example, a 0.00001% false positive rate would receive a score of 0. We came to the number ten million based on our own serial data bus datasets. In those datasets we expect at

least one million messages per hour of traffic. If a greater than single digit number of false positives occurs in an hour, then it is likely to result in alert fatigue at best and safety risks at worst. The safety problem is more pronounced for a LIPS than for a MIPS, as a MIPS firing on accident is recoverable while a LIPS firing on accident often requires a system reboot.

7.6.3 Detection Evaluation - Detection Efficacy

The following criteria are used to test the efficacy of a chosen IDS. We do this against our four attacks of spoofing, corruption, manipulation, and cancelling, as an IDS may specialize in one kind of attack and not cover others, or the IDS may incidentally detect types of attacks it was not necessarily designed for. IDS efficacy is separate from IPS suitability in that IPS suitability focuses on speed and the risks of alerting on benign messages, while IDS efficacy is centered on how effective the IDS is at detecting attacks without concern for timescales or any negative consequences. Our criteria are focused on how an IDS handles changing attacks, any gaps in coverage, and any attacks completely missed by the IDS. An evaluator may use this metric to understand what attacks they cannot detect, while gaining an understanding of how an attacker might circumvent their IDS.

Adaptability is a metric indicating the flexibility of the detection system as an attacker modifies their attack. If the attacker can change either how they deliver their attack, or the payload of it, and remain undetected, then the detection system is too rigid for competent attackers. Increased adaptability often comes with the risk of decreased accuracy. We rank adaptability on a 4 point scale.

- 25 - Static Rule, detects a specific payload
- 50 - Dynamic Rule, based on bus behavior but relies on particular attack targets
- 75 - Independent of attacker behavior with exceptions
- 100 - Independent of attacker behavior

Coverage indicates the percentage of traffic the intrusion detection system is capable of evaluating. For example, imagine an IDS which detects spoofing attacks by looking for anomalies in timing intervals between periodic messages. In this case coverage would be the percentage of traffic in which periodic messages occur. Coverage is differentiated from adaptation by being concerned with the types of messages an attack uses. In contrast, adaptation is concerned with how the attack is delivered. For

example, aperiodic messages are types of messages and so affect coverage, while an attacker changing their transmission scheme or attack payload is considered adaptation. For categorical evaluations of an IDS, listing all message IDs not covered by that IDS is a useful exercise as it indicates where static rules or future detection systems may be useful.

False Negatives occur whenever an IDS fails to alert on an attacker message. This criterion is a measure of how effective the IDS is. We expect that many detection systems have high false negatives for some attack types, as different detection techniques are better for different kinds of attacks. The false negatives score is calculated as one minus the percentage of false negatives.

7.6.4 Calculating a Score

After the evaluation process the evaluator is left with eight scores, along with three pass/fails. The three pass/fails are based on IPS viability for each attack vector. If any of the IPS viability metrics fail, then the IPS is deemed unviable for that attack vector. A summary of the previously mentioned criteria calculation methods can be seen in Algorithms 1 and 2. These algorithms require the input of the bus speed, weights, the original trace, the trigger time for the IPS and IDS, and the IDS's verdict on the message. IPS suitability creates 4 scores, one each for spoofing, corruption, manipulation, and cancelling attacks. This score is computed by using Algorithm 4, and calculating IPS suitability. If the desired outcome is a MIPS, then speed and false positives are averaged together to compute a score. Otherwise latency and false positives are averaged. The components of each of our scores are equally weighted. IDS efficacy works the same way as IPS suitability with 4 scores across our four attacks. The process to calculate the components of the scores is described in Algorithm 5. Once again the score is equal to the average of all three components, weighted equally.

7.6.5 Weights

Evaluator defined weights are our mechanism for accounting for different threat models. As we stated in our introduction, all security tools inherently come at some cost and the evaluator is the best positioned to adjust the weights of the evaluation framework to reflect how they see that cost. By default, the evaluation framework assumes that all of the criteria are equally important and thus equally weighted. However, there are scenarios where some criteria are more important than others. Let us imagine an example where the most important thing to a defender is stopping known attacks. The

Algorithm 4 IPS Suitability Score Calculation

```
1: T_transmitting, FP, i ← 0
2: fastEnough ← True
3: //Iterate through Trace/IDS Lists
4: while i < trace.len do
5:     //Measure Speed and FPs for IDS
6:     if idsMessages[i].result = True then
7:         //Calculate IDS (T_d) and IPS (T_p) Time
8:         T_d ← idsMessages[i].time − trace[i].time
9:         T_p ← ipsTimes[i] − T_d
10:        //Calculate total msg transmit time (T_t)
11:        T_t ← B_spd * trace[i].numBits
12:        //Get % of msg taken by IDS/IPS
13:        Speed ← (T_d + T_p)/T_t
14:        if Speed > 1 then fastEnough ← False
15:        //Check for False Positives
16:        if trace[i].attack = False then
17:            FP ← FP + 1
18:    //Track total transmitting time
19:    T_transmitting ← T_transmitting + T_t
20:    i ← i + 1
21: //Fail if IDS/IPS longer than message
22: if fastEnough = True then
23:    S_S ← 100
24: else
25:    S_S ← 0
26: //Calculate Bus Utilization (B_util)
27: B_util ← T_transmitting/totalTraceTime
28: //Calculate how far IDS falls behind
29: Latency ← T_d * (1 + (1 − B_util))/avgTransmitTime
30: //Fail if Latency is > one message
31: if Latency ≤ 1 then
32:    S_L ← 100
33: else
34:    S_L ← 0
35: //Calculate False Positives Score
36: S_FP ← 1 − (FP/trace.len) * 10000000
37: if Category = MIPS then
38:    S_Suitability ← (w_1 * S_S) + (w_3 * S_FP)/2
39: else
40:    S_Suitability ← (w_2 * S_L) + (w_3 * S_FP)/2
```

Algorithm 5 IDS Efficacy Score Calculation

```
1: S_A ← See Step Function
2: IdSet, alertedSet ← []
3: FN, i ← 0
4: //Get unique IDs of Trace and Attacks
5: while i < trace.len do
6:     IdSet.add(trace[i].id)
7:     if idsMessages[i].result = True then
8:         alertedSet.add(idsMessages[i].id)
9:         //Check for False Negatives
10:    else if trace[i].attack = True then
11:        FN ← FN + 1
12:    i ← i + 1
13: //Calculate Coverage of alerted IDs (S_C)
14: S_C ← alertedSet.len/idSet.len
15: //Calculate Percent False Negatives (S_FN)
16: S_FN ← 1 − (FN/trace.len)
17: //Calculate Final Efficacy Score
18: S_Efficacy ← (w_4 * S_A) + (w_5 * S_C) + (w_6 * S_FN)/3
```

defender has excellent threat intelligence and knows roughly what an attack is going to look like. In this scenario, false negatives are the most important criterion for IDS efficacy and adaptation is inconsequential if false negatives are low. As for coverage, it is only important for the attacks the evaluator expects to see (which are presumably the ones in their testing dataset). For IPS suitability, stopping attacks requires MIPS. Between speed and false positives, speed is the most important criterion for ensuring that known attacks are prevented, especially if preventing a few benign messages does not impact the safety of the vehicle. After applying weights, the evaluator has a framework that works for their specific situation.

7.7 Attack Testing

Our evaluation framework makes the assumption that the evaluator has the capacity to test spoofing, corruption, manipulation and cancelling attacks against an IDS. To enable this, we provide an open source tool which randomly adds our attacks to datasets [74]. Our tool is generic and can support any protocol for our 4 attacks. Emulating these attacks in software ensures that we can easily test all of our attack types without complicated hardware setups. While spoofing attacks are simple enough to support in a hardware in the loop setup, corruption attacks require loading

'malicious' software onto an LRU and manipulation and cancelling attacks require custom hardware to drive the bus. Attempting to emulate these attacks outside of software often leads to the attacker using an implant and then performing a spoofing attack by the nature of using an implant. This testing setup results in extremely limited attack testing. By testing the same attack payload with a corruption attack, spoofing attack, and manipulation attack, the defender can see gaps in coverage for each attack.

The entire tool works by processing each message in a dataset and generating a random number; the chosen attack is executed whenever the random number is a certain value. We refer to this scenario as an an attack trigger. The rest of this section describes our process for emulating each kind of attack. Notably our injected attacks do not modify the rest of the dataset to add the ramifications of the attack, as this is irrelevant to testing a prevention system. An IDS/IPS needs to be able to detect attacks based on a single message to effectively prevent attacks.

Spoofing Attacks are done using the reflexive method. On an attack trigger, our tool copies the victim message and changes the timestamp of the copy to be a random time in the future. Lower and upper bound parameters control how far away the copied message is. A greater range is useful for testing interval based solutions, particularly for discovering aperiodic messages which can have multiple timing intervals.

Corruption Attacks require the user to point to the location of the data field, data being the engineering values aspect of the message. On an attack trigger, our tool randomizes the data of the victim message in-place. The result is that the timestamp of the victim remains the same and only the data is modified. The randomization of the data field represents a less subtle version of the attack than is technically possible. However, our tool is not equipped to model and simulate the entire system and so cannot recreate the downstream effects of an attack.

Manipulation Attacks are done by flipping a random bit in a message after an attack trigger, with a parameter for determining how many bits in a message are flipped. Manipulation attacks are frequently detected using voltage based detection systems, which are incompatible with this method of software based attack recreation. To emulate this technique the tool needs to be adjusted to output the appropriate encoded signal (i.e Manchester encoding) based on a bitstring, then create valid bit flips somewhere in the signal.

Cancelling Attacks are done by randomly removing messages from the dataset. Notably some protocols automatically replay a cancelled message and require consecutive cancellation attacks to eventually silence the replay. To support further denial of

service type attacks, our tool also supports silence and flooding attacks. For silence attacks, timestamps after an attack trigger are increased by a random static integer to simulate silence on the bus. For flooding attacks, the victim message is copied and duplicated a random number of times with a small randomized timestamp increase for each message.

7.8 Case Studies

This section provides case studies for two different protocols: CAN and MIL-STD-1553. These protocols are both serial data bus protocols but with vastly different architectures. Testing multiple architectures gives us confidence in the broad applicability of our framework to multiple serial data bus network protocols. For each protocol we choose an IPS technology and an IDS technology then evaluate them against our framework, filling in protocol-specific information and limitations as we go along. We discuss how to test each step in detail rather than re-implement four technologies ourselves. Our framework assumes the defender has a threat model in mind. For these case studies we assume all attack vectors and attacks are viable. We assume remote attackers are weak attackers, while implants and supply chains default to being strong attackers. Table 7.1 provides a list of scores for other detection and prevention systems. Some assumptions were made to produce scores for detection systems with vastly different datasets and missing paper data. We provide a detailed explanation for how these scores are produced later in this section.

	IPS Viability			IPS Suitability				IDS Efficacy			
IDS/IPS (M/L)	Implant	Remote	Supply	Spoof	Corrupt	Manip	Cancel	Spoof	Corrupt	Manip	Cancel
[86] / [90] (L)	-	✓	-	100	0	0	100	98	0	0	98
[12] / [30] (M)	✓	✓	✓	100	50	0	100	89.7	88.7	0	89.7
[72] / [30] (M)	✓	✓	✓	0	0	0	0	0	91.3	0	0
[41] / [30] (L)	-	✓	-	100	0	100	100	99.99	0	?	?
(Seq) [88] /1553(L)	-	✓	-	100	0	0	100	99.8	0	0	100
(Fp) [88] /1553(L)	-	✓	-	50	0	0	0	99.7	0	0	0
[29] / [40] (M)	-	✓	-	50	0	0	0	91.7	0	0	0

Table 7.1: Our evaluation framework applied to IDS/IPS combinations. We assume a strong implant and supply attacker, and weak remote attacker. A '?' indicates the system was not designed for those attacks but would likely detect them.

7.8.1 CAN

The easiest mechanism for doing MIPS on CAN is actually the one that enables LIPS. Interrupting a message generates an error frame which informs the rest of the bus to ignore the victim message. Doing this repeatedly puts the victim LRU into the bus-off state where its transceiver is turned off [11]. This means that if the MIPS triggers enough times on a CAN bus, then it effectively becomes a LIPS. A side effect of this error process is that manipulation attacks are exceedingly difficult on CAN. Manipulating any individual bit results in the message being cancelled unless the attacker commits to driving the bus and overpowering any error frames so the rest of the bus thinks the manipulated message is a legitimate message.

For our IPS we choose CANstomper [30], which flips one of the final 11 bits of a message to get the rest of the bus to drop it. Let us go through IPS validity. First we identify that CANstomper is a MIPS, as it cancels messages as it goes across the bus, with the option to be a LIPS across multiple attacks. We will evaluate the validity for both. For MIPS we assume attack correction is not possible for implants, remote attackers, or supply chain attackers, as a 1 to 0 transition is simple on the CAN bus, and so CANstomper can reasonably randomize the strength of their attack. LIPS is more complicated, but CANstomper is well suited to it. Because it cancels a message as it goes on the bus, there is no need to know an address and no need to determine which device is transmitting (attestation required). This is true for all of our attack vectors. Read required is more complicated because the LIPS relies on the the attacker reading the bus, realizing it made an error, and incrementing an error counter such that it eventually turns off. Notably CAN devices have a way to reset their own error counters, meaning the attacker would also need to choose to not do that. From this we can conclude that LIPS is effective against weak attackers, but not strong attackers. In our imagined threat model this means our IPS is invalid against implants and supply chain attacks, but valid against weak remote attacks.

For our detection system we select Cho's clock skew fingerprinting solution [12]. Let us start with the suitability of clock skew fingerprinting to becoming a prevention system. We evaluate this for spoofing, corruption, manipulation, and cancelling attacks. Cho's solution was built to detect what we refer to as spoofing, corruption, and cancelling attacks. Manipulation attacks are not detectable, so we will assume a score of zero.

The first criterion for IPS suitability is speed. Based on their implementation we have no reason to assume speed varies based on the kind of attack. Speed is not covered in the paper, but we know from the algorithm presented in the paper that the

IDS has enough data to analyze the CAN packet after 29 bits (116 microseconds). We know CANstomper can trigger in the last 11 bits of a CAN packet, meaning if Cho's IDS takes less than 340 microseconds then it passes this test. Assuming competent hardware, we can reasonably conclude a score of 100 for speed. False positives are at 0% for spoofing attacks, 0.055% for corruption attacks, and 0% for cancelling attacks. This leads to a score of 100, 0, and 100 for these attacks respectively. The IPS suitability scores for our 4 attacks are 100, 50, 0, 100.

Now we evaluate adaptation, coverage, and false negatives to calculate the efficacy of Cho's IDS. Adaptation we give a 75 for all attacks. Interval based solutions are not concerned with the payload of the attacker, making them resilient to specific attacks. However, clock skew can be spoofed, making it weak to certain attackers [77]. Coverage is also the same for each of our attacks, as the IDS's algorithm detects each attack in the same way. For coverage we see that Cho's work does not cover aperiodic messages. Based on our datasets roughly 6% of CAN IDs are not periodic, meaning coverage is 94%. False negatives vary from attack to attack. Spoofing and cancelling attacks have 0 false negatives (100), while corruption attacks have 3% (97). The total score is the average of the three criteria, resulting in: 89.7, 88.7, 0, 89.7.

7.8.2 MIL-STD-1553 and Combining IDSes

MIL-STD-1553 makes an interesting case study because it uses a bus controller to dictate all communication, and because it includes a built in LIPS techniques. A message called a 'mode code' can inform any LRU to disable itself; the premise being that if an LRU is malfunctioning it can be told to stop transmitting until either told to resume, or the system restarts. This makes LIPS easy for the MIL-STD-1553 bus, assuming the defender has some mechanism for source attestation. The shutdown mode code uses a 5 bit address field to target the victim LRU, but an attacker capable of spoofing this address field can trivially cause a LIPS to disable any LRU they choose. Additionally, LIPS only works if the attacker is taking over a legitimate device (e.g., a supply chain attack, or remote attack). Otherwise the attacker has no address that can be targeted by a mode code. In terms of counter-acting LIPS, there are mode codes which re-enable LRUs, but once the attacker is affected by the shutdown mode code their only option is resetting themselves or waiting for the vehicle to reset. From this description we can conclude the following for LIPS validity: addressing is required, attestation is required, and an attacker reading the bus is required. This implies our LIPS only works against weak attackers of remote and supply chain attacks, and does not work at all against implants due to the addressing issue. Crucially our choice

of IDS hinges on the need for source attestation, otherwise a spoofing attack could weaponize our LIPS.

Based on the need for source attestation we choose Orly's IDS synthesis of voltage fingerprinting and message sequence detection [88]. It works by first determining if an LRU matches a known fingerprint, then determining the likelihood that the observed message ID would appear in that time slot. It is designed to function against spoofing and cancelling attacks. With corruption attacks not considered, as the detection system does not look at the data words in MIL-STD-1553. Manipulation attacks are possibly detected by the voltage fingerprinting, but the paper does not run this experiment. Spoofing and cancelling detection is handled by the message sequence IDS, and voltage fingerprinting detects only spoofing.

In terms of IPS suitability, Orly does not specify the latency of either detection system or the system as a whole. Given that neither technique relies on data words and only requires the first 16 bits of a message to function, we can likely assume both techniques would be fast enough to not extend into the next message. For false positives the score changes for each attack. Spoofing and cancelling attacks receive a 100, as no false positives occur for the sequence-based detection system. The same is not true for the voltage fingerprinting IDS. At best the false positive rate is 0.4%. Even if only 200,000 messages appeared in an hour of traffic, we would expect 800 false positives an hour. An unacceptable number for the prevention system of a fighter aircraft. This means that we have a score of 100, 0, 0, 100 for sequence-based detection and a score of 50, 0, 0, 0 for voltage based detection. From this we can conclude that sequence-based detection is suitable to IPS, while the voltage fingerprinting solution is not.

IDS efficacy for our two detection systems perfectly demonstrates why one might want to have an IDS exist but not be a prevention system. The sequence-based IDS scores a 100 on adaption for spoofing attacks as the attacker cannot change how they transmit a message to avoid detection. Coverage is reduced as aperiodic messages are not included in the sequence-based detection. While we do not know how many aperiodic messages occur in a MIL-STD-1553 dataset, we do know the message powering our LIPS is considered aperiodic. If an attacker can circumvent the entire IDS while executing a highly impactful (and simple) attack, then the IDS has serious issues. The final metric for spoofing is false negatives, which scores a 100 as no attacks in their dataset are missed. Cancelling attacks score a 100 for adaptation as transmitting an aperiodic message is irrelevant to cancelling messages. Coverage is 100% for cancelling attacks as the next message is delayed. False negatives are also

reported at 0, resulting in a score of 100, (100 - the percent of aperiodic messages), 100 for spoofing attacks, and 100,100,100 for cancelling attacks. Assuming 0.5% of aperiodic messages we get scores of 99.8,0,0,100 for spoofing, corruption, manipulation, and cancelling attacks respectively.

But now let us consider the voltage fingerprinting solution, their 'RT Authentication'. The detection is based on the electrical fingerprint of one device being different from another. Assuming decent hardware, an attacker cannot change their attack to avoid detection. The fingerprinting receives a 100 for spoofing attack adaptation. Coverage is also 100 as it works on any spoofing attack, not just periodic messages. False negative rates vary depending on the spoofing device, but at lowest are .9%, giving a score of 99.1. Resulting in a score of 99.7,0,0,0 for each of our attacks.

Comparing the scores of these two detection systems can reveal issues in either the testing dataset, or the IDS efficacy. We have zero false negatives for sequence-based detection, but not full coverage for one of the detection systems. Indicating that our dataset has the appropriate attacks, but the sequence-based IDS does not process them. For the sake of becoming a prevention system this is good, as Orly indicates that aperiodic messages would result in false positives, which would then make the IPS suitability score of the sequence-based IDS low for spoofing attacks. But for the sake of actually detecting attacks, ignoring aperiodic messages is a disastrous hole in coverage. Orly smartly pairs the fingerprinting technique with the periodic detection technique to ensure they cover all spoofing attacks. The pairing of these techniques ensures that the overall system does not lose anything by having only the fingerprinting solution consider aperiodic messages. This flicking on and off of functionality, both within and between attack categories, maximizes what each IDS is good at without increasing the overall noise of the system. Scores for an individual IDS highlight what functionality is missing. Scores for multiple IDSes demonstrate how to configure each IDS to maximize the accuracy of the overall system and safely prevent attacks when one can.

The last example that needs covering is weights. Based on how our LIPS works, it is clear that some rare messages are particularly dangerous. It follows that coverage is the most important criterion for understanding how good an IDS is for MIL-STD-1553. The exact numbers are a bit vague as we lack Orly's implementation and datasets, but we can apply a higher weight to coverage such that missing aperiodic messages result in an extremely low overall score. It is important to note that evaluators do not necessarily need to stick completely to our criteria. If not detecting LRU disabling messages or not detecting weapon system messages immediately invalidates an IDS,

then giving a score of 0 unless those conditions are met is perfectly fine. Better to do simple conditional checks for system specific issues than risk the 0.01% in a 99.99% coverage score being a protocol-specific attack that every attacker would try.

7.8.3 Scoring Details

This section presents the criteria scores for the work referenced in Table 7.1. All of these scores should be taken with a healthy degree of skepticism as we were unable to perform dynamic testing, each paper used unpublished attack datasets, the accuracy is based on their own testing, and we had to extrapolate attack performance, speed, and coverage based on details in the paper. We generally favored more generous interpretations of existing research when we lacked concrete details. We would expect fewer 100s on an independently created attack data set applied to each IDS. A summary of scores is in Table 7.2.

Song [86] is evaluated with a dataset with no aperiodic messages, and only tests transmitting fast messages. We assume canceling attacks are incidentally detected by the interval deviation caused by a message not appearing. Song does not specify the algorithm watches for messages which appear too slowly, but its a small enough leap for us to consider canceling attacks as having the same security guarantees as spoofing attacks. In terms of scoring, the basis of relying on intervals makes Song's work similar to Cho's [12], but without the additional fingerprinting and corruption attacks. Adaptation is 100 instead of 75 as pure intervals cannot be faked like Cho's clock skews. Our definition of corruption attacks is different from Cho's masquerade attacks, but the principle is similar enough that we put it in the corruption category. We define corruption as a more advanced version of masquerading attacks where only one attacker is necessary. We would expect Cho to out perform Song in terms of accuracy with a different dataset, as the fingerprinting provided by Cho is a boon for detecting spoofing attacks.

Quinonez [72] only covers corruption attacks as it is specifically watching for deviations in the data field, without any concern for where the data is coming from. We know this because they detect the flood of sensor data by the deviations in the data rather than the sudden influx of inputs. In terms of scores, adaptability being 75 is in reference to it not detecting data fields with small fluctuations. Coverage is technically limited to the 13 sensors they base their invariance model on, though this is seemingly the only sensors on their small vehicle. We would expect less coverage on an actual automobile with more computers.

		IPS Suitability			IDS Efficacy		
IDS/IPS (MIPS/LIPS)	Attack	Speed	Latency	FP	Adapt	Coverage	FN
Song et. al [86]/ [90] (LIPS)	Spoof	-	100	100	100	94	100
Song et. al [86]/ [90] (LIPS)	Cancel	-	100	100	100	94	100
Cho and Shin [12]/ [30] (MIPS)	Spoof	100	-	100	75	94	100
Cho and Shin [12]/ [30] (MIPS)	Corrupt	100	-	0	75	94	97
Cho and Shin [12]/ [30] (MIPS)	Cancel	n.a	-	n.a	75	94	100
Quinonez et. al [72]/ [30] (MIPS)	Corrupt	0	-	0	75	100	99
Kneib and Huth [41]/ [30] (LIPS)	Spoof	0	100	100	100	100	99.99
Kneib and Huth [41]/ [30] (LIPS)	Manip	0	100	?	100	100	?
Kneib and Huth [41]/ [30] (LIPS)	Cancel	0	100	?	100	100	?
Orly (Sequence) [88]/LIPS	Spoof	-	100	100	100	99.5	100
Orly (Sequence) [88]/LIPS	Cancel	-	100	100	100	100	100
Orly (Fingerprint) [88]/LIPS	Spoof	-	100	0	100	100	99.1
Orly (Fingerprint) [88]/LIPS	Manip	-	100	?	100	100	?
Generuex [29]/ [40] (MIPS)	Spoof	0	-	100	75	100	100

Table 7.2: Our evaluation framework applied to IDS/IPS combinations, where this table breaks down the criteria for IPS suitability and IDS efficacy for different attacks.

Kneib and Huth [41] detect spoofing attacks via electronic fingerprinting, and have explicit limitations that show they cannot detect corruption attacks. A latency score of 100 may be somewhat misleading as the system uses a suspicion counter to detect devices which were not present during training. This means it may take tens of messages to actually generate an alert. It is unclear how high the false positives would be without this suspicion system. We believe this research approach would also detect manipulation and cancelling attacks. Manipulation attacks generally cause spikes in voltages as the bus is overridden, this should cause a fingerprint mismatch. Cancelling attacks can be done via a bit flip in CAN and result in a bus-off attack that is referenced in the chapter. That said, neither of these attacks are explicitly tested, so we are unaware of how well their IDS performs.

Generux [29] has an adaptability of 75 because the response time of messages is a high level detection metric, but it is also fake-able by any attacker capable of pre-processing their attack. In terms of speed we found detection time was greater than the length of a message.

Table 7.2 does not include Wasicek [98] and Choi [14] among others from Section 7.1, as they did not provide enough data on the performance of their IDS, their false positives, their false negatives, or their data set to make meaningful score estimations. This is not to say these detection systems do not work, merely that our detection system expects certain inputs and their papers do not happen to provide them.

7.9 Discussion

7.9.1 Tradeoffs Between Criteria

Detection system criteria are often tradeoffs with each other. Let us take false positives as an example. As we see it, one can reduce false positives in three ways: improve the detection mechanism, layer detection mechanisms, or more finely target the scope of the detection mechanism. All of these decisions come with tradeoffs. Algorithm improvements increase complexity and represent an opportunity cost, layering increases computational complexity and lowers speed, reducing scope lowers coverage. A defender must constantly balance the need for greater detection capabilities with their available resources and their ability to filter through results. We highlight false positives in particular because they reduce an operators trust in the system they are using. Overly broad security solutions risk replicating the complexities of secure operations centers, where teams of skilled analysts search through alerts to separate the signal from the

noise. This solution is anathema to the quick reaction times necessary to actually do something about attacks on vehicles and weapons systems.

7.9.2 Addressing Relative Scores

The inherent problem with computing scores based on attacks is that if someone else uses different attack datasets then scores cannot be meaningfully compared. Our attack tool helps solve this problem, but only to the extent of making it easier to create attack datasets. The ideal scenario for researchers is a universally agreed upon core dataset, which the attack tool inserts attacks into. As researchers discover stealthier attacks they can contribute to the attack tool such that all future detection systems are tested rigorously. We believe this would result in more intentionally designed IDSes, but would not necessarily lead to a meaningful universal score. Having full knowledge of a test makes it easy to cheat the test for a higher score, intentionally or not.

7.9.3 Comparing Scores

In our case studies we use individual scores to identify potential areas of improvement. Each criterion indicates where resources need to be focused. The other use case is comparing scores generated with the same dataset but from multiple security systems. Scores should only be compared within the same attack category. The goal is to maximize scores across attack categories by taking the highest scoring component from each IDS. Weights ensure that the defender's priorities are incorporated into the score, meaning we can trust aggregate scores to indicate which system is better for the defender. The only complication is how the criteria are calculated. For example, a 1% difference in coverage does not mean that IDSes have the same coverage except for that 1%. If some subset of messages are more important or more likely to be attacked then the system with lesser coverage may be better. The magnitude of the difference is helpful for maximizing detection with available resources, but more analysis into how each criterion is calculated is needed to draw meaningful conclusions from a scoring difference. These conclusions could be used to synthesize IDSes or create focused analytics for known gaps.

7.9.4 Use By System Owners and Red Teams

It is our opinion that the evaluation framework is better used as a relative score by maintainers and evaluators trying to select detection and prevention systems.

Particularly in the increasingly frequent scenario of a system owner trying to determine if a security product actually does what they need. Of course, a maintainer is not interested in the exact language academics use, but it is important for them to understand what attacks and attack vectors they defend against. Take the example of an interval-based detection approach [86]. This approach is designed to detect spoofing attacks and cancelling attacks. Spoofing detection is great for implants, all they can do is spoof, but what about remote and supply chain attackers? If the attacker controls a victim device and transmits the messages it would normally transmit in the correct time slot, but with malicious data, then they detect nothing. After using our evaluation framework the maintainer knows the IDS scores highly against spoofing attacks, but is largely useless against corruption, and manipulation attacks. Now they know to procure another IDS solution which covers these attacks. Our attack tool helps ensure these attacks are randomized and high level enough that an IDS developer cannot simply guess static rules to detect the attacks a red team might think of.

7.9.5 Extending the Evaluation Framework

While we believe our evaluation framework is sufficiently high level to accommodate new attacks and new IDS/IPS techniques, an evaluator may wish to extend the framework to provide more details for their threat model. As the evaluation framework currently exists there are two adjustment knobs available to the evaluator - weights and sub-attacks. If the evaluator finds one criteria is more important to them than another, they can reflect that by increasing the weight. If the evaluator finds a specific type of attack they are concerned with, then they can evaluate for exactly that attack rather than the higher level attacks we describe. The result is another score, which is easily comparable for an evaluator testing multiple security systems, and which is easily interpretable for third parties by sharing the specifics of the attack dataset. The last place to extend the framework is adding to the criterion list.

Adding a criteria only requires adding a new weight parameter as long as some basic rules are followed. For IDS efficacy and IPS suitability The new criteria needs to produce a score from -100 to 100, and be testable based on an attack dataset. The criteria does not necessarily need to be quantitative, as shown by adaptation, but qualitative scores are worse for setups which expect multiple evaluators who could interpret the data differently. Qualitative criterion are particularly effective when they encourage the evaluator to further examine a critical aspect of the IDS/IPS. For example, adaptation encourages the evaluator to ask how the system works and how

circumventable it is. The more distinguished the qualitative options are the better they are. Adding criteria to IPS validity is as simple as adding yes/no questions for either MIPS or LIPS.

7.10 Conclusion

In this chapter, we propose an evaluation framework for serial data bus networks to help researchers and defenders understand the capabilities and limitations of their detection and prevention systems. Our framework can be used to securely and efficiently develop new systems, or to evaluate existing technologies against specific threat models. We encourage the safe and intentional adoption of IPS technology by defining a more nuanced view of IPS. We examine IPS technology through the lens of attack vectors and create new categories of IPS based on whether the defender wants to prevent individual messages or disable LRUs. Additionally, we evaluate IDS technologies based on the high level attack behavior they can detect, and their suitability to powering an IPS. These insights allow a defender to know when an attacker can circumvent their IPS, what attacks their IDS does not cover, and what aspects of their IDS can be safely be connected to an IPS. The security and safety of serial data bus networks is crucial, and with the knowledge of our framework defenders can more confidently achieve it.

Chapter 8

Related Work

This chapter further examines the serial data bus research this thesis is built upon and compares it to our work. We split this research into the categories of attacking and defending serial data bus networks. In the attack section we break down authentication issues, protocol-based attacks, and attack vectors. In the defense section we examine cryptographic security solutions, timing analysis, data analysis, voltage analysis, and protocol specific defense mechanisms.

8.1 Attacking Serial Data Bus Networks

Most offensive security research for serial data bus networks is focused on the CAN bus. This focus is likely because consumer automobiles are far more accessible than industrial systems, critical infrastructure, or weapons systems. That said, any offensive CAN bus research done on a consumer automobile works just as well on these more inaccessible systems.

8.1.1 Authentication Issues and Initial Work

Initial CAN research is focused on what can be done by sending messages on a CAN bus. These found attacks which include: "disabling the brakes, selectively braking individual wheels on demand, stopping the engine" and a number of other attacks [42]. This work also bypassed many of the intended safeguards in CAN communication, such as some segmentation in the CAN bus. Notably early attack work required extensive reverse engineering for controlled attacks. The proprietary nature of CAN thus meant that an attacker needed to have access to the same make and model of vehicle to target it. However, vehicles are weak to fuzzing, meaning that any access to the CAN bus can lead to electronically disabling the vehicle. Over the next several years this

work continued, demonstrating similar vulnerabilities across different vehicles [18], [10]. Some security safeguards were put into consumer automobiles. One manufacturer had ECUs ignore certain messages on the bus if the vehicle was going over 5 MPH. However these techniques were bypassed, usually by flooding the bus and creating race conditions between attacker traffic and legitimate traffic [52], [31]. The work in this thesis largely avoids bypassing authentication mechanisms, as it begins with the assumption that we are connected to the bus or taking over an existing device. We assume that we can obtain any secret key by corrupting a device.

8.1.2 Protocol Attacks

So far we have described attacks which simply use the lack of authentication to attack the CAN bus. But as detection systems are proposed there is a need to develop unique ways of interacting with the bus to avoid detection. Chapter 3 already heavily emphasizes the bus-off attack [11], which produces repeated errors to turn off ECU transceivers. Advances on that attack randomize it and disable the auto-replay feature on the attacking system to make it more difficult for a defender to detect [6]. CANNON [44] pauses the clock to inject bits at arbitrary times. This is used to trigger a bus-off attack without the synchronization difficulties of the original bus-off paper or the reliance on predicting what messages are going to appear. These advances make the attack more difficult to detect with CAN errors while our advances in Chapter 3 make it not produce any errors at all.

In recent years new attack research has moved away from consumer CAN and towards other protocols and standards. Burakova [8] draws attention to the benefit of J1939 standardizing the data field, namely removing the need for reverse engineering. They test this by disabling safety critical functionality of a J1939 system using the standard. Mukherjee [55] develops a new attack by claiming all 255 addresses in J1939, such that no legitimate system can speak on the network. Alternatively it claims the address of a single device. Many safety critical components are not configured to dynamically claim an address, resulting in a safety critical error through a single networking message. These attacks are often simple to detect through any monitoring of a CAN packet but difficult to detect with the kind of meta data analysis common to the field and included in this thesis.

Serial data bus offensive research is extremely limited outside of CAN. Hayden gave a broad overview of attacks on MIL-STD-1553, breaking down the effects on an aircraft [83]. Orly proposed attacks based on the MIL-STD-1553 standard. The consequences of these attacks, spoofing the computers connected to the serial data bus,

are similar to CAN attacks but focused on the MIL-STD-1553 command word [88]. Genereux affirms these MIL-STD-1553 attacks [29]. A summary of attacks can be seen in Louis et al's work [48]. These attacks inform Chapter 6 by ensuring that we understand cancelled out messages correctly, as the process for this can be vague in the protocol specifications.

Offensive ARINC and commercial avionic research is less straightforward. Instead of focusing on the serial data bus itself, they focus on the connection between in-flight infotainment networks, and how it connects to both the pilot's network, which provides air traffic control data, and the command network, which operates the aircraft. Researchers have been unable to bridge the gap to the command network, and have successfully read data from the pilot's network from the infotainment network [78], [4]. A key complicating factor is the one way communication of ARINC systems [57]. With direct physical access to the command network, spoofing sensor data should be possible. This would be essentially an intentional version of the 2019 Boeing737 max incidents [47].

Most satellite security research is focused on the uplink itself, rather than the serial data bus network [67]. Hack-a-Sat, a competition by the US Air Force to hack satellites devoted one part of the competition to serial data bus communication [71], making it clear this a viable attack vector to some actors. Satellite networks frequently use multiple data buses, including MIL-STD-1553, though there is limited research on spacewire security. Our work on individual busses should be easily transferable to a multi-bus system with the exception of security for the gateway devices.

8.1.3 Attack Vectors

Most early attack work relies on connecting to the CAN bus via the OBD-II diagnostic port. This lead to some skepticism in how possible car hacking really was, as physical access requires the victim to be targeted and does not scale very well. To further legitimize car hacking as a field, researchers started looking into remotely hacking vehicles. Miller used a cellular connected vehicle infotainment system to pivot to the internal bus [52]. Similar techniques have since been proposed in the rail and aviation industries [56], [78]. Alternatively Kovelman uses the Bluetooth features of a vehicle to remotely corrupt the vehicle [43]. Bluetooth introduces a range of privacy concerns, particularly in the rental car industry, though privacy is not a focus of this thesis. The growing desire to extract analytics from a vehicle in real time only makes these remote attacks more plausible [39], [91]. In the military space this is demonstrated by Abrams tanks having a centralized health monitoring system [79], and

the U2-Dragonfly accepting in-flight remote firmware updates [95]. The recent focus on fingerprinting in CAN literature shows that this remote attack vector is not being addressed. Chapters 5 and 6 demonstrate how to tailor the security system towards remote attacks. The last area of consideration is attacks using custom hardware. This includes using commercial-off-the-shelf FPGAs to drive the voltage of the CAN bus. Giannopoulos [30] does this to cancel messages or flip bits. The same technology is used as an intrusion prevention system. Our work assumes that that this protocol-breaking hardware is fully plausible considering the impact from hacking the industrial and weapon systems we focus on.

8.2 Defending Serial Data Bus Networks

We split defensive research into two categories, cryptography and intrusion detection. Cryptography tries to solves the authentication problem by adding authentication to the bus. Intrusion detection research assumes the attacker is already on the bus and attempts to identify anomalies in bus data.

8.2.1 Cryptography

Cryptography based solutions add a key exchange mechanism and modify every ECU to be compatible with said key exchange mechanism. Using these keys, each ECU authenticates messages over the bus. Typically these solutions follow the AUTOSAR specification for key exchange mechanisms [2]. For CAN in particular, cryptography raises two main problems: bus load must remain low and each CAN 2.0 message can only transmit 8 bytes of data at a time.

If the bus load ever reaches 100% then the performance of the vehicle may suffer, causing safety concerns. Typical guidance for CAN vehicles is to never exceed 70% bus load for regular traffic [97]. Considering every CAN packet only has 8 bytes of data, with half that as overhead in an extended CAN ID, exchanging secure keys can take a significant number of packets. The arbitration ID itself cannot be signed with a key, as an unpredictable ID violates the arbitration process, leading to higher priority messages not always being sent first. This means the authentication must be a separate packet from the original message.

8.2.1.1 Heartbeat Authentication

CANcrypt [70] gets around the arbitration ID problem by adding a device which generates secure keys, then having each ECU send a heartbeat signal signed with

the secure key to prevent spoofing. Each ECU is programmed with message IDs that only it sends. If an attacker attempts to spoof that ECU, then the victim ECU stops transmitting the heartbeat signal. This lack of heartbeat alerts the system that something is wrong. CANcrypt assumes each ECU sends unique messages, with a max of 15 ECUs connected to the bus. In comparison the max for a J1939 system is 30 ECUs. These maximums are based off of theoretical bus load and could be exceeded based on the system. The true limit is the number of unique devices claiming an address on the bus. J1939 uses 1 byte for source addresses, but only addresses 0-253 are available, making the limit 254 address claiming devices. Limiting the number of ECUs means this solution does not work for every vehicle. Luxury automobiles support up to 150 ECUs as of 2017 [25], and this number is only rising [100]. In terms of unique IDs, tying the ID to the source address to allow for multiple ECUs of the same type, along with logic for detecting new IDs, ensures an attacker cannot spoof an ECU without the heartbeat signal stopping. Chapter 3 demonstrates how ineffective this approach is, as there are plenty of ways to disable an ECU such that it cannot send the heartbeat message.

8.2.1.2 Authentication Packets

Alternatively, Groza and Murvay load a secure key onto each ECU as they are installed. Each ECU uses Diffie-Hellman to compute a randomized key on startup. Then each ECU sends an authentication message of 24 bytes during run time. If the authenticated message does not match the HMAC computed by the ECU, then it discards the messages and ignores it. This provides protection against any injected messages, but not denial of service attacks. The initial key exchange has the most overhead but only needs to be done once. On a standard 250Kbit/s CAN2.0 bus the overhead is 7.28ms, with an 192 bit token and 64 randomized bits. Groza later removes most of the overhead of initial computation time by hiding the key exchange in the CAN arbitration process [35], as discovered by Muller and Lothspeich [59].

For run time authentication, Groza offers an alternative approach to compress the 24 bytes to 2 CAN packets of overhead, but it is still 2 packets per message sent by the bus. For each ECU this provides a maximum computational overhead of 6.2ms for SHA256 and a maximum computational overhead of 120ms for SHA3-256 [58]. No message is meant to transmit faster than once per 10ms in J1939 [97], making this computational delay acceptable for SHA256. Assuming a standard 250Kbit/s CAN2.0 bus, the authentication packets take an additional 6.144ms to be transmitted. Emergency messages, and many engine related messages transmit every 10ms, meaning

the periodicity of these messages is impacted by the authentication message. The performance impact of a periodicity delay depends entirely on the pre-authentication bus load. If the bus load is such that emergency messages could not be reliably sent then the system is not suitable for Murvay and Groza's authentication approach [58]. Any busload or timing concerns are fixed by using a CAN-FD bus, as it uses a 64 byte data field, and transmits at 1Mbits/s. This demonstrates cryptographic approaches are easy solutions for modern and future systems, but difficult with existing systems.

8.2.1.3 Cryptography Conclusion

A number of other authentication schemes exist from voting [89], collective authentication [34], a secure symmetric server [38], etc.. The research problem is no longer securely generating and exchanging keys, barring a side channel mechanism to steal the keys. Instead, the focus is on removing bus overhead such that system performance is not affected. We are aware of no research which offers a crypto solution for CAN2.0 systems with no constraints on the number of ECUs connceted to the bus. This is a lesser problem for CAN-FD systems, which is the CAN implementation in most modern consumer systems [15]. Cryptography does not solve denial of service attacks, but it does successfully address message injection attacks and attackers manipulating existing messages.

The core problem with a cryptography-based solution is updating every existing ECU. This requires a software upgrade from a diverse set of manufacturers that must be maintained as the system is upgraded and new ECUs are added to the bus. Upgrading each ECU, with the complicated supply chains of vehicles, is an expensive process. It may be possible to automate this process through J1939 commands which dump ECU firmware, along with static code analysis, though applying this to multiple manufacturers may be difficult.

8.2.2 Intrusion Detection Systems

This section expands on the intrusion detection system overview presented in Chapter 7.

8.2.2.1 Timing-Based Detection Systems

A number of papers apply machine learning algorithms such as Deep Forest [9], Deep Neural Networks [68] [103], Markov Models [88], and Bloomfilters [32] to identify when a message is sent out of order. Song calculates the interval between two messages with the same ID and monitors to see if the message occurs at more than double

its previous speed [86]. Chapter 5 significantly improves on this work by adding an upperbound and defining intervals based on greater logic than double the known speed. Cho fingerprints ECUs using the clock skew of each ECU [12]. The main benefit of this approach over other timing-based work is the fingerprinting capabilities. ECUs in CAN are in charge of synchronizing themselves to the bus, thus any skew in an ECU's clock can be used to uniquely identify them. Beyond fingerprinting, clock skew allows for detecting attackers sending periodic messages. Cleverly, it also detects a corrupted ECU masquerading as another ECU on the bus. Lee [46] relies on sending requests on the bus, which requires an IDS capable of transmitting data, something which risks increasing the bus utilization such that legitimate messages fail to transmit on noisy buses. Requests are a known attack vector in J1939, which may cause other problems on the bus [55]. Several of these papers identify aperiodic messages as difficult to detect. Some authors propose that messages are not quite as periodic as indicated by these papers [44]. Avoiding some sub-set of messages provides a clear target for an attacker, as we demonstrate and detect against in Chapter 5.

8.2.2.2 Data-Based Detection Systems

Orly [88], and Groza [32] parse the data field without knowledge of data contents, or where data fields begin and end. They rely on the cyclic nature of the bus to assume the data field will change predictably. It is unclear how effective this blind understanding of the data field is against attacks. Small changes in the data field could result in safety consequences. How frequently values are confused with each other depends on how much each data field changes. If a data field does not change frequently and is positioned next to a state flag, then it could be considered one larger field. These occurrences are determined by the system manufacturer and outside the defender's control. The level of ambiguity offered by these data solutions is insufficient for accurately tracking the comprehensive state of the system.

In terms of research that currently uses an enriched data field, the closest is physical invariance models [72], and neural networks incorporating the OBD-II pinout [98] [36]. Each of these papers account for primary vehicle functions but are inherently limited in their data analysis. Quinonez [72] applies physical model restrictions to a small line following robot car, making it unclear how well this approach scales to the magnitude of data coming off a CAN bus. Wasicek et al. [98] references ten data points. Even factoring in recent reverse engineering work [69], only 32 data points are extracted. Hanselmann et al. [36] uses a neural network on individual IDs, but it lacks context for what the data means. This ambiguity, in combination with the general difficulty of

actionable alerts from neural networks makes this approach poorly suited for any real time decisions. In regards to detection, each of these approaches acts in isolation of message timing. We believe physical models are best suited to an integrated solution between data and timing, as this improves timing analysis while ensuring a fast attacker cannot bypass the physical model by intelligently flooding the bus. Chapter 5 implements this idea, while Chapter 6 avoids the data problem entirely by limiting the detection system to new messages. Avoiding the data problem is often better if corrupted devices are not an issue, as data-based approaches are prone to false positives if not tuned in such a way that limits their effectiveness against small data changes. Our approach of using CUSUM against the predicted data in Chapter 5 helps combat these issues for detecting corrupted devices.

8.2.2.3 Voltage-Based Detection Systems

Some work attempts to protect against LRU spoofing through voltage based fingerprinting [14], [88], [76], [13], [41]. The primary benefit of signal fingerprinting is detecting new devices put onto the bus, and ensuring a corrupted LRU can only transmit the messages sent by the victim LRU. These security benefits can be replicated with timing model, though as we have discussed timing models can struggle with certain types of messages. Kneib and Huth use a combination of "variations in supply voltage, variations in grounding, variations in resistors, termination and cables, and imperfections in bus topology causing reflections" to fingerprint CAN ECUs. They first take a signal and preprocess out three sets: a rising edge, a falling edge, and two dominant bits in a row. Feature selection takes the mean, standard deviation, variance, skewness, kurtosis, root mean square, maximum, and energy of the preprocessed sets. The features are then classified based on which ECU sent them, with a supervised learning phase to teach the classifier which ECU is which. Kneib and Huth identify 99.85% of attacks with 0 false positives (after filtering). They use the differential nature of the bus to avoid some of the environmental impacts of other voltage-based IDSes. These environmental factors result in changes to the voltage of the bus as temperatures change, this means that voltage fingerprinting criteria must be chosen carefully [41]. This filtering process can make it difficult to tell if the attack could modify their approach to bypass filering mechanism. Targeting the fingerprinting solution to specifically voltage manipulation as done in Chapter 5 or targeting to an individual device as done in Chapter 6 can reduce any environmental effects by lowering the sensitivity of the overall model.

Voltage fingerprinting techniques assume the attacker cannot replicate the signal of a given ECU on a silent bus. Admittedly, doing so would require increasingly complex hardware where the complexity is based on the complexity of the fingerprinting mechanism. That said, we have to assume an actor capable of gaining physical access to industrial systems or military aircraft also has the advanced hardware engineering skills to perform this attack. Bhatia's DUET attack [5] demonstrates that two attackers acting together can bypass voltage fingerprinting. This works by corrupting the voltage processed by the voltage intrusion detection system during training. We detect this highly advanced attacker on CAN system in Chapter 5 by taking advantage of the CAN error process forcing the bus into predictable states that the attacker needs to avoid.

8.2.2.4 CAN-Based Detection Systems

Detection systems which directly rely on the features of CAN are rare, and largely contained within CAN specific attack papers. These rely on CAN features to circumvent detection systems. The countermeasures for CAN attack papers either focus on CAN specific indicators of compromise from the attack or changing CAN's behavior to eliminate the attack. The bus-off attack paper [11] proposes detecting its attack by monitoring for a large number of error frames followed by the victim message replayed on the bus. They posit this requires a custom CAN controller, though we detect the attack with a standard controller by monitoring the REC register in Chapter 3.

CANNON's authors [44] take a similar tact to EREFs and propose changing the gating structure on CAN peripherals to prevent their exploit from working. Interestingly they also propose a power analysis system which watches for a sudden drop in power as the peripheral clock is disabled. Power analysis on legitimate ECUs is a potential future area of research. While it is not currently practical to implement at scale, research indicating that other attacks cause power fluctuations would make it a tempting option.

8.2.2.5 MIL-STD-1553-Based Detection Systems

There are two security papers focused on MIL-STD-1553, both focused on intrusion detection. Orly et al uses RT fingerprinting, determining if a signal observed over the bus is the same as previously seen over the bus. After passing RT fingerprinting, Orly uses several machine learning techniques to predict the sequence of messages across the bus, with a separate model for periodic and aperiodic messages [88].

For the aperiodic model Orly et al tries to predict what aperiodic message would come next based on aperiodic message history. By definition, the transmission of an aperiodic message does not influence the next aperiodic message sent. For example, a mode code which disables one computer, has no bearing on then firing a weapon system. That said, the periodic model presented in this paper is effective, particularly on a MIL-STD-1553 bus implementing the major/minor frame timing scheme. Our work in Chapter 6 addresses the periodic data issue as well as all aperiodic messages.

Genereux builds a timing model for periodic 1553 messages based on the timing delays caused by processing time and physical location [29]. Essentially the paper claims that each computer has a predictable processing time and by combining that processing time with the propagation delay caused by the location on the bus, they can determine if a legitimate RT was speaking. Genereux's implementation of this technique does not cover aperiodic messages. We believe a sophisticated adversary could pre-process messages based on the observed delay each time the victim RT transmits and knowledge of the positions of different LRUs on the bus. Chapter 6 avoids this problem by detecting on the spoof message itself rather than the meta-data surrounding it. This ensures that even if a legitimate device is taken down, the attack can still be detected by the voltage on the bus controller.

Chapter 9

Conclusion

The driving question behind this thesis asks how we can design more practical and more efficient intrusion detection systems for serial data bus networks. Detection systems which are accurate enough, fast enough and flexible enough to be implemented on the decades of vehicles already using serial data bus networks. The motivation behind this question is threefold. Accuracy problems in existing work, specifically high false positives, make it too difficult for a defender to draw meaningful conclusions from an IDS and make the IDS unusable if the intent is to perform near real time actions in response to an alert. Gaps in message coverage result in easy IDS bypasses for attackers with knowledge of the IDS and the victim system. Considering we are discussing attackers targeting expensive industrial equipment and weapons systems, this is all attackers. The final issue is the gap in protocol knowledge between the attacker and defender. While attack papers are taking advantage of protocol design to accomplish their goals, most defensive papers are using high level approaches agnostic to protocol design. This thesis answers our core question by developing methodologies, frameworks, and example detection systems which demonstrate that protocol-centric design can lead to greater accuracy, greater coverage, and simpler designs.

9.1 Overall Conclusions

The research presented in this thesis provides a clear throughline for answering its core question. For answering how to design more efficient and practical detection systems, practical enough to be implementable on the decades of systems which already exist, and efficient enough to be actionable and usable by defenders. To answer this question we focus on the nuance of serial data bus protocols. The motivating attacks of Chapter 3 emphasize what an attacker can do with knowledge of the protocol. The methodology for protocol-centric IDS development presented in Chapter 4, with

an example in Chapter 5 and Chapter 6, demonstrates how our approach leads to simpler and more accurate solutions which cover all messages on the bus, not just the easily predictable ones. Finally, our evaluation framework provides defenders the tools necessary to measure how good their IDS is on an attack-by-attack basis and to know what parts of their IDS can safely be used to power a prevention system. We see prevention as the ultimate end goal for serial data bus security and this thesis demonstrates how to architect a detection system around that goal.

In essence, we explore what an attacker can do with protocol knowledge, what a defender can do with protocol knowledge, and then how to evaluate detection and prevention systems. From this we can draw conclusions about how detection and prevention systems should be designed for serial data bus networks. The main issue is one of attacks and attack vectors. Most detection systems are spectacular at detecting one kind of attack or any attack from a particular attack vector, but they fail when confronted with a certain kind of message or when confronted with a corrupted device. Of course, no detection system can detect everything, no matter how monolithic. The more attack space a singular detection system tries to cover, the more likely it is to produce false positives or alerts that no user can understand. False positives and confusing alerts are the inherent risk of complexity and are where protocol knowledge can be beneficial. Protocol knowledge allows a defender to simplify aspects of their detection system. CANDID demonstrates this by using CAN error frames to reduce a voltage fingerprinting system to a bit-flip detection system, and our MIL-STD-1553 detection system uses built in error flags to detect collisions. The protocol will largely operate how it was designed, factoring that into a detection system is free. Counter intuitively, we find that this approach of designing with the protocol in mind from the onset can create improvements in accuracy and attack coverage, where normally the simplification of a detection system leads to the trade off of greater accuracy at the cost of attack coverage. We see this in particular with our MIL-STD-1553 detection system, where simple voltage monitoring on the bus controller completely replaces spoofing detection while expanding coverage to unpredictable user initiated messages.

This all is not to say that a completely generic solution would not have some merit. CANDID could have been a voltage fingerprinting system, data analysis system, and timing system integrated together. However, that increase in algorithmic and computational complexity is costly. CANDID has no need to fingerprint ECUs and CANDID already has spoofing detection from its timing and data analysis. This is where practicality comes in. The best detection system does everything and uses all of its overlapping techniques to accommodate for the strengths and weaknesses of

individual modules. But a real computer needs to run this detection system, and that computer needs to be inexpensive, light weight, and have a small enough form factor to be deployed to countless legacy systems which were likely not designed to have an extra computer slotted into them. Add in the speed requirements for an intrusion prevention system, and clearly the approach of stacking a plethora of detection systems on top of each other is not practical outside of forensic analysis. Designing detection systems using the protocol allows the defender to replace parts of our generic solution with more practical and efficient detection systems which are more capable of swiftly detecting and preventing attacks.

9.2 Chapter Contributions

In Chapter 3 we propose a stealthy CAN attack which uses bit-flips to disable transceivers without producing any visible errors, allowing an attacker to spoof the victim device without concern that artifacts of the victim device's transmissions will give them away. By examining a bit-flip attacker through a probabilistic lens, we determine that even with a 40% chance of flipping each individual bit, an attacker can execute our stealthy attack nearly 100% of the time. We expect our stealthy bit-flip attack, which comes from an attacker who would have better odds calling a coin flip than flipping a bit, will have two major influences on future academic work. The first is centered on bit-flip attacks. Bit-flip attacks do not need to be particularly sophisticated to bypass security systems while having a high impact on the system. Based on this conclusion, bit-flips should more frequently be factored into security system design as a reliable way to drop messages, a potential way to disable devices connected to the bus, and in advanced cases a way of altering messages as they are transmitted.

Our second influence is around CAN errors and built in error processes. The error process makes attacks simpler and more reliable. Specifically, a common issue for attackers is not knowing what data will be on the bus when an attack is executed. CAN errors provide a reliable bus state which an attacker can effectively loop forever. Chapter 3 contributes examples of how CAN errors can be used for attack and defense with our proposed blind sync methodology and Error Resistant Error Frames (EREFs) respectively.

The blind sync methodology allows a limited attacker to guarantee that they can send a valid message on the bus, without being able to read the bus. This methodology lowers the complexity of attack payloads such that flipping 2 bits is

enough to force the bus to the start of a new message. Our methodology specifically enables electromagnetic interference attackers, who previously were reliant on getting lucky and executing an attack during dead bus time. Our methodology also enables limited remote device exploits where the attacker is incapable of queuing messages. Future security systems should consider adding these attacks vectors to their threat model.

EREFs are a change to the CAN error process such that an interrupted error frame will retransmit, but the error counter will not increase until the error frame completes normally. Additionally, we propose 3 methods for differentiating legitimate error frames from attacker induced error frames. EREFs can be used directly to remove the threat of stealthy bit-flip attacks while our error detection methods can be used to allow an IDS to safely alert on CAN error frames. This allows a defender to protect against future exploits of the CAN error process. As future CAN systems are developed, EREFs can be safely integrated into the CAN controller. We believe intrusion detection systems must be easily integratable with existing systems, otherwise they are too impractical to be deployed broadly. But easy integration typically comes with the tradeoff of worse detection efficacy. To offset these tradeoffs, we examine IDS design and the benefits of protocol specific approaches to security.

Chapter 4 proposes a novel methodology for designing generic intrusion detection systems on serial data bus networks and for incorporating protocol specific information into a security device. Designing detection systems with the protocol reduces the shock of attack papers by enabling preemptive detection capabilities. The generic design approach allows defenders to prioritize security resources based on the attacks they think are likely and it provides defenders a road map to full attack coverage. Using our methodology for protocol-centric design, we develop our own more effective CAN IDS which tackles the problems of accuracy and attack coverage in CAN security.

Chapter 5 proposes CANDID, a novel CAN intrusion detection system. CANDID has greater accuracy and coverage than existing work, as well as a more advanced attacker capable of corrupting ECUs and flipping bits. Existing work which attempts to detect spoofing attacks finds that some percentage of messages have irregular frequencies. These messages either result in false positives or are ignored. CANDID augments its timing analysis with data analysis to understand why timing changes occur and why certain messages are only sent at specific times. This contextual awareness allows CANDID to extend message coverage to all messages, ensuring an attacker cannot cherry pick messages with irregular frequencies to bypass an IDS. On top of our timing analysis, CANDID uses CUSUM to alert on deviations in

correlated data fields. Doing so allows CANDID to detect a subtle attacker performing incremental attacks. When combined with our extended timing analysis, our IDS ensures than an attacker must corrupt a device to send its messages undetected, and even then the attacker can only send messages from their corrupted device that are not dependent on data from the rest of the system. The integration of timing and data analysis to better detect spoofing attacks and corrupted devices is a key contribution of this thesis. We expect more future research to integrate state information into existing detection techniques.

CANDID's bit-flip attack detection uses CAN error frames as a predictable bus state such that CANDID can assume it only needs to be able to detect 0 to 1 bit-flips. 0 to 1 bit-flips are far easier to detect on the CAN bus because of the physical architecture of the protocol. Assuming all physical layer manipulations result in CAN errors, our detection system is far simpler and more reliable than full voltage fingerprinting solutions. Additionally, our solution removes the attack window generated by voltage fingerprinting's consistent need for retraining. This strategy of using CAN error frames is directly from Chapter 3, where an attacker uses a CAN error frame to make an attack more reliable. We expect detection systems which require voltage manipulation detection but not fingerprinting to use this technique going forward.

Overall, CANDID demonstrates that our protocol-centric IDS methodology can achieve greater levels of detection, but CAN is a widely researched protocol. To prove out the effectiveness of our methodology we must demonstrate its effectiveness for protocols with very limited foundational research. MIL-STD-1553, the protocol powering fighter aircraft and other important pieces of national infrastructure, is arguably just as important as CAN, but it lacks the wealth of security research that CAN enjoys.

Chapter 6 proposes a bus controller fingerprinting solution for MIL-STD-1553 which detects 100% of illegitimate messages started on the bus, as well as any collisions. Our solution is applicable for all MIL-STD-1553 systems and is the first security research tailored to the protocol's architecture. Our IDS ensures we can detect an attacker turning off devices and enabling weapon systems, where previous security research only applies to periodic messages or detecting implanted devices. We expect this chapter to result in simpler, more protocol-centric, detection systems as more IDSes are developed around rare messages instead of relying on rules-based solutions to cover edge cases.

After developing multiple intrusion detection systems with a new methodology, we need a way to evaluate how good our detection systems are. But normal confusion

matrix metrics fail to capture the nuance of different attacker capabilities, how those capabilities interact with different detection criteria, and how the detection criteria affect what the IDS can safely be used for.

To solve these problems, Chapter 7 proposes: a new structure for attacker behavior, an evaluation framework for prevention and detection systems, and an open source tool to inject attacks into serial data bus network datasets. Examining attacker behavior on a message by message basis encourages high level detection schemes based on how an attacker has to interact with the system. This encourages greater coverage and adaptability. Additionally, message by message analysis is necessary for intrusion prevention systems to function. This encourages lower false positives and higher speed. Our evaluation framework allows a defender to know what attacks they are secure against and which attack vectors their prevention system works against. Prevention can be enabled for fast and accurate IDS modules and further detection resources can be allocated based on the defender's perception of likely threats and gaps in their existing systems. Our attack generation tool assists defenders with generating robust attack data such that they can test their IDS. In combination with the protocol-centric IDS development methodology outlined in Chapter 4, our framework allows a defender to safely and efficiently construct a comprehensive IDS and IPS.

9.3 Final Thoughts

This thesis takes great strides in solving the issues of false positives, message coverage, and protocol-specific attacks for serial data bus networks. For CAN, our integration of the data field reduces false positives by increasing the accuracy of timing analysis and eliminating the coverage issues of aperiodic messages. Additionally, our CAN error analysis and voltage detection scheme remove the threat of CAN error attacks. For MIL-STD-1553, our trust in the bus controller eliminates obvious protocol attacks while expanding coverage and simplifying the overall detection system. Our IDS development methodology and evaluation framework provide a solid foundation for designing security systems for legacy protocols. Now future defenders can short-circuit the decade of CAN research required to develop reasonably secure systems and design secure systems from day one of researching a protocol.

The core conclusion of this thesis is that a defender can build more comprehensive, practical, and lightweight detection and prevention systems for serial data bus networks by designing around the protocol. While building these security systems, a defender must evaluate their work on the basis of what attacks and attack vectors they can

reasonably detect and prevent in their own threat model. Doing otherwise risks an eternal cat and mouse game in academia between attack and defense papers, while industry implementations cover pieces but never the whole of the attack space. This thesis provides the tools, techniques, and methodologies a defender needs to produce practical and efficient detection systems for legacy serial data bus networks.

Bibliography

[1] Andrey Akinshin. Lowland multimodality detection. *aakinshin.net*, Nov 2020.

[2] Autosar. specification of secure onboard communication, 4.3.1 edition, 2017.

[3] Omid Avatefipour and Hafiz Malik. State-of-the-Art Survey on In-Vehicle Network Communication (CAN-Bus) Security and Vulnerabilities. *arXiv e-prints*, page arXiv:1802.01725, Feb 2018.

[4] P. Berthier, C. Bresteau, and J. Fernandez. On the security of aircraft communication networks. In *On the security of aircraft communication networks*, volume 10707, pages 266–269. Springer Verlag, 2018.

[5] Rohit Bhatia, Vireshwar Kumar, Khaled Serag, Z. Berkay Celik, Mathias Payer, and Dongyan Xu. Evading voltage-based intrusion detection on automotive can. In *NDSS*, 01 2021.

[6] Gedare Bloom. WeepingCAN: A Stealthy CAN Bus-off Attack. *NDSS*, February 2021.

[7] Robert Bosch et al. Can specification version 2.0. *Rober Bousch GmbH, Postfach*, 300240:72, 1991.

[8] Yelizaveta Burakova, Bill Hass, Leif Millar, and André Weimerskirch. Truck hacking: An experimental analysis of the SAE j1939 standard. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, 2016. USENIX Association.

[9] Matthew Butler. An Intrusion Detection System for Heavy Duty Vehicle Networks. In *ICCWS*, volume 12, pages 399–405, 2017.

[10] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack

surfaces. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association.

[11] K. Cho and K Shin. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1044–1055, New York, NY, USA, 2016. Association for Computing Machinery.

[12] K. Cho and K Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 911–927, Berkeley, CA, USA, 2016. USENIX Association.

[13] Kyong-Tak Cho and Kang Shin. Viden: Attacker identification on in-vehicle networks, 2017.

[14] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee. Voltageids: Low-level communication characteristics for automotive intrusion detection system. *IEEE Transactions on Information Forensics and Security*, 13(8):2114–2129, 2018.

[15] CiA. Can 2020: The future of can technology, Mar 2016.

[16] Cisco. `https://www.snort.org/`.

[17] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *1987 IEEE Symposium on Security and Privacy*, pages 184–184, 1987.

[18] Roderick Currie. Hacking the can bus: Basic manipulation of a modern automobile through can bus reverse engineering. *SANS Institute*, 2017.

[19] Jeremy Daily, David Nnaji, and Ben Ettlinger. Securing CAN Traffic on J1939 Networks, 2021.

[20] Data Device Corporation. Electrical and layout considerations for 1553 terminal design. https://www.milstd1553.com/wp-content/uploads/2012/12/MIL-STD-1553B.pdf, 2012.

[21] Jennifer Cheeseman Day and Andrew W. Hait. Number of truckers at all-time high. *US Census Bureau*, July 2019.

[22] D. De Santo, C.S. Malavenda, S.P. Romano, and C. Vecchio. Exploiting the mil-std-1553 avionic data bus with an active cyber device. *Computers & Security*, 100:102097, 2021.

[23] Elastic. `https://www.elastic.co/products/elasticsearch`.

[24] Automotive ecu: Journey from mechanical to electronics based control units, Jul 2017.

[25] Embitel. Automotive ecu: Journey from mechanical to electronics based control units, Jul 2017.

[26] Fortune Business Insights Pvt. Ltd. Marine vessel market size, share and industry analysis, by type (commercial vessel, passenger ship, lng/lpg carrier, and special purpose vessel), by system (marine engine system, sensor system, control system, electrical system, auxiliary system, and communication system), and regional forecast, 2019-2026.

[27] Mahsa Foruhandeh, Yanmao Man, Ryan Gerdes, Ming Li, and Thidapat Chantem. Simple: Single-frame based physical layer identification for intrusion detection and prevention on in-vehicle networks. *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, pages 229–244, 2019.

[28] Paula Fraga-Lamas, Tiago M Fernández-Caramés, and Luis Castedo. Towards the internet of smart trains: A review on industrial iot-connected railways, Jun 2017.

[29] 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*, 56(1):276–284, 2020.

[30] Hristos Giannopoulos, Alexander M Wyglinski, and Joseph Chapman. Securing vehicular controller area networks: An approach to active bus-level countermeasures. *IEEE Vehicular Technology Magazine*, 12(4):60–68, 2017.

[31] Andy Greenberg. Hackers remotely kill a jeep on the highway-with me in it, Jul 2015.

[32] Bogdan Groza and Pal Stefan Murvay. Efficient Intrusion Detection with Bloom Filtering in Controller Area Networks. *IEEE Transactions on Information Forensics and Security*, 14(4):1037–1051, 2019.

[33] Bogdan Groza, Stefan Murvay, Anthony van Herrewege, and Ingrid Verbauwhede. LiBrA-CAN: Lightweight broadcast authentication for controller area networks. *ACM Transactions on Embedded Computing Systems*, 16(3), 2017.

[34] Bogdan Groza, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. LiBrA-CAN: Lightweight broadcast authentication for controller area networks. *ACM Transactions on Embedded Computing Systems*, 16(3), 2017.

[35] Bogdan Groza, Lucian Popa, and Pal-Stefan Murvay. TRICKS—Time TRIggered Covert Key Sharing for Controller Area Networks. *IEEE Access*, 7:104294–104307, 2019.

[36] Markus Hanselmann, Thilo Strauss, Katharina Dormann, and Holger Ulmer. Canet: An unsupervised intrusion detection system for high dimensional CAN bus data. *CoRR*, abs/1906.02492, 2019.

[37] FRANK E. HARRELL and C. E. DAVIS. A new distribution-free quantile estimator. *Biometrika*, 69(3):635–640, 12 1982.

[38] Oliver Hartkopp and R MaCAN SCHILLING. Message authenticated can. In *Escar Conference, Berlin, Germany*, 2012.

[39] Chris Hartnoll. Rolls-Royce: Optimising jet engine maintenance with machine learning. *Harvard Business School*, 2018.

[40] Buscop mil-std-1553b cybersecurity. `https://www.idahoscientific.com/pdfs/ISCI-BusCop_Product_Sheet_082321.pdf`.

[41] Marcel Kneib and Christopher Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 787–800, New York, NY, USA, 2018. Association for Computing Machinery.

[42] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security

analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, May 2010.

[43] Alexei Kovelman. A remote attack on the bosch drivelog connector dongle, Mar 2017.

[44] Sekar Kulandaivel, Shalabh Jain, Jorge Guajardo, and Vyas Sekar. Cannon: Reliable and stealthy remote shutdown attacks via unaltered automotive microcontrollers. *2021 IEEE Symposium on Security and Privacy (SP)*, 2021.

[45] Can bus error handling. https://www.kvaser.com/about-can/the-can-protocol/can-error-handling/, Jan 2017.

[46] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In *Proceedings - 2017 15th Annual Conference on Privacy, Security and Trust, PST 2017*, Proceedings - 2017 15th Annual Conference on Privacy, Security and Trust, PST 2017, pages 57–66. Institute of Electrical and Electronics Engineers Inc., September 2018.

[47] Adam Levin and Ryan Beene. Sensors linked to boeing 737 crashes vulnerable to failure. *Bloomberg.com*, 2019.

[48] Karim Lounis, Ziad Mansour, Michael Wrana, Marwa A. Elsayed, Steven H. H. Ding, and Mohammed Zulkernine. A review and analysis of attack vectors on mil-std-1553 communication bus. *IEEE Transactions on Aerospace and Electronic Systems*, pages 1–1, 2022.

[49] George Marsh. Typhoon: Europe's finest. *Avionics Today*, June 2003.

[50] Kara McDermott. Navy's cybersecurity program office gears up for hackthemachine unmanned, Nov 2021.

[51] MiG-35. Mig-35 multi-role combat aircraft. *Defense Update*, 2007. `https://web.archive.org/web/20070314164702/http://www.defense-update.com/products/m/mig35.htm`.

[52] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015(S 91), 2015.

[53] Kyle Mizokami. Nsa to pentagon: Lock down your weapons before hackers get to them, Oct 2021.

[54] Thomas Morris, Rayford Vaughn, and Yoginder Dandass. A retrofit network intrusion detection system for modbus rtu and ascii industrial control systems. In *2012 45th Hawaii International Conference on System Sciences*, pages 2338–2345, 2012.

[55] Subhojeet Mukherjee, Hossein Shirazi, Indrakshi Ray, Jeremy Daily, and Rose Gamble. Practical DoS Attacks on Embedded Networks in Commercial Vehicles. In *International Conference on Information Systems Security*, volume 10063, pages 23–42, 2016.

[56] Ken Munro. Wi-fi as a weapon: how customer communication networks can be used to take over trains. In *InfraRail*, 2018.

[57] Ken Munro and Alex Lomas. 747-400 walk through from a hacker's perspective. defcon 28 - aerospace village (2020).

[58] P. Murvay and B. Groza. Security shortcomings and countermeasures for the SAE j1939 commercial vehicle bus protocol. *IEEE Transactions on Vehicular Technology*, 67(5):4325–4339, May 2018.

[59] Andreas Müller and Timo Lothspeich. Plug-and-secure communication for can, 2015.

[60] Neousys Technology. Poc-351vtc series.

[61] US Department of Defense. Department of defense interface standard, digital time division command/response multiplex data bus, 2018.

[62] Department of Transportation. Positive train control (ptc), Jun 2020.

[63] Francis Onodueze and Darsana Josyula. Anomaly detection on mil-std-1553 dataset using machine learning algorithms. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 592–598, 2020.

[64] Kris Osborn. Pentagon, industry pursue increased data bus security, Jul 2017.

[65] E. S. PAGE. CONTINUOUS INSPECTION SCHEMES. *Biometrika*, 41(1-2):100–115, 06 1954.

[66] D. S. Paraforos, V. Vassiliadis, D. Kortenbruck, K. Stamkopoulos, V. Ziogas, A. A. Sapounas, and H. W. Griepentrog. Automating the process of importing data into an fmis using information from tractor's CAN-bus communication. *Advances in Animal Biosciences*, 8(2):650–655, 2017.

[67] James Pavur, Daniel Moser, Vincent Lenders, and Ivan Martinovic. Secrets in the sky: on privacy and infrastructure security in dvb-s satellite broadband. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 277–284, 2019.

[68] Krzysztof Pawelec, Robert A Bridges, and Frank L Combs. Towards a CAN IDS based on a neural-network data field. arXiv, 2017.

[69] Mert D. Pesé, Troy Stacer, C. Andrés Campos, Eric Newberry, Dongyao Chen, and Kang G. Shin. LibreCan: Automated can message translator. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 2283–2300, 2019.

[70] O. Pfeiffer. *Implementing Scalable CAN Security with CANcrypt: Authentication and Encryption for CANopen, J1939 and Other Controller Area Network Or CAN FD Protocols*. Embedded Systems Academy Incorporated, 2017.

[71] Secretary of the Air Force Public Affairs. Us defense, air force departments invite hackers to re-imagine how space systems are secur, Aug 2020.

[72] Raul Quinonez, Jairo Giraldo, Luis Salazar, Erick Bauman, Alvaro Cardenas, and Zhiqiang Lin. SAVIOR: Securing autonomous vehicles with robust physical invariants. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 895–912. USENIX Association, August 2020.

[73] Vivek Ramachandran and Sukumar Nandi. Detecting arp spoofing: An active technique. In Sushil Jajodia and Chandan Mazumdar, editors, *Information Systems Security*, pages 239–250, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[74] Matthew Rogers. https://github.com/matthewRekos/serializedNightshade.

[75] Matthew Rogers and Kasper Rasmussen. Silently disabling ecus and enabling blind attacks on the can bus. In *ESCAR Europe*, 2022.

[76] Sang Uk Sagong. *Physics-based Security Analysis of Controller Area Network Protocols*. PhD thesis, University of Washington, 2019.

[77] Sang Uk Sagong, Xuhang Ying, Andrew Clark, Linda Bushnell, and Radha Poovendran. Cloaking the Clock: Emulating Clock Skew in Controller Area Networks. *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018*, pages 32–42, 2018.

[78] Ruben Santamarta. Last Call for SATCOM Security, 2018.

[79] Carol Scheina. Abrams demonstration proves concept for enterprise-level system health monitoring. *www.army.mil*, Sep 2020.

[80] Jayaprakash Selvaraj, David Ware, Gökçen Yılmaz Dayanikli, Ryan M. Gerdes, Neelam Prabhu Gaunkar, and Mani Mina. Electromagnetic induction attacks against embedded systems. *ASIACCS 2018 - Proceedings of the 2018 ACM Asia Conference on Computer and Communications Security*, pages 499–510, 2018.

[81] U S Senate. WEAPON SYSTEMS CYBERSECURITY DOD Just Beginning to Grapple with Scale of Vulnerabilities United States Government Accountability Office, 2018.

[82] Congressional Research Service. F-35 joint strike fighter (jsf) program (crs report no rl30563), April 2020.

[83] Patrick M. Hayden David K. Woolrich Katherine D. Sobolewski. Providing Cyber Situational Awareness on Defense Platform Networks. *The Cyber Defense Review*, 2(2):125–140, 2017.

[84] Society of Automotive Engineers. SAE J1939 Standards Collection. https://www.sae.org/publications/collections/content/j1939_dl/.

[85] Society of Automotive Engineers. J1939dab: J1939 digital annex - sae international. `https://www.sae.org/standards/content/j1939da_202007/`, 2020.

[86] H. M. Song, H. R. Kim, and H. K. Kim. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In *2016 International Conference on Information Networking (ICOIN)*, pages 63–68, 2016.

[87] Splunk. https://www.splunk.com/.

[88] Orly Stan, Adi Cohen, Yuval Elovici, and Asaf Shabtai. Intrusion Detection System for the MIL-STD-1553 Communication Bus. *IEEE Transactions on Aerospace and Electronic Systems*, 9251(c):1–17, 2019.

[89] Christopher Szilagyi and Philip Koopman. Low cost multicast authentication via validity voting in time-triggered embedded control networks, 01 2010.

[90] Masaru Takada, Yuki Osada, and Masakatu Morii. Counter attack against the bus-off attack on CAN. *Proceedings - 2019 14th Asia Joint Conference on Information Security, AsiaJCIS 2019*, pages 96–102, 2019.

[91] Adam Tanner. Data monitoring saves some people money on car insurance, but some will pay more. *Forbes*, October 2013.

[92] Stefano Tedeschi, Christos Emmanouilidis, Jörn Mehnen, and Rajkumar Roy. A design approach to IoT endpoint security for production machinery monitoring. *Sensors (Switzerland)*, 19(10), 2019.

[93] Joseph Trevithick. Lockheed made a three minute long cartoon just to explain f-35's alis, Jun 2019.

[94] Joseph Trevithick. Army hires company to develop cyber defenses for its strykers after the vehicles got hacked, Nov 2020.

[95] Joseph Trevithick. U-2 spy plane got new target recognition capabilities in first ever in flight software updates. *The Drive*, Oct 2020.

[96] US Federal Motor Carrier Safety Administration. Electronic logging devices and hours of service supporting documents. *Department of Transportation*, Dec 2015.

[97] Wilfried Voss. Guide to SAE j1939 - controller area network and j1939, 2018.

[98] Armin Wasicek, Mert D Pesé, André Weimerskirch, Yelizaveta Burakova, and Karan Singh. Context-aware intrusion detection in automotive control systems. In *Proc. 5th ESCAR USA Conf*, pages 21–22, 2017.

[99] *Waveshare*. 2-Channel Isolated CAN Expansion HAT for Raspberry Pi, Dual Chips Solution.

[100] Ally Winning. Number of automotive ecus continues to rise, May 2019.

[101] Clinton Young, Joseph Zambreno, Habeeb Olufowobi, and Gedare Bloom. Survey of automotive controller area network intrusion detection systems. *IEEE Design and Test*, 36(6):48–55, 2019.

[102] Duncan Young. Mil-std-1553's longevity is well deserved, Aug 2010.

[103] Aiguo Zhou, Zhenyu Li, and Yong Shen. Anomaly detection of CAN bus messages using a deep neural network for autonomous vehicles. *Applied Sciences (Switzerland)*, 9(15):1–12, 2019.