IOActive®

Research-fueled Security Services

\ WHITE PAPER \

Arm IDA and Cross Check: Reversing the 787's Core Network

Ruben Santamarta
Principal Security Consultant

August 2019

Abstract

In 2008, the Dreamliner was presented as the world's first e-Enabled commercial airplane.[1] Boeing certainly introduced an impressive new set of functionalities, enabling the vast majority of the components to be highly integrated with and connected to regular systems, such as onboard maintenance, data-load, and the Crew Information System.

In order to achieve this degree of integration, the 787's different aircraft data domains were conceived to operate in a common network, without being physically isolated.

In this paper, IOActive has documented our detailed attack paths and component vulnerabilities to describe the first plausible, detailed public attack paths to effectively reach the avionics network on a commercial airplane from either non-critical domains, such as Passenger Information and Entertainment Services, or even external networks.

We believe as strongly in safety as we do in security. We provide these detailed findings herein so that all stakeholders, members of the security industry, and affected entities can form their own judgment as to the exploitability and impact of these confirmed software vulnerabilities.

[1] https://www.boeing.com/commercial/aeromagazine/articles/qtr_01_09/article_05_1.html

Contents

Notices

Trademarks

IOActive and the IOActive logo are trademarks and/or registered trademarks of IOActive, Inc. in the United States and other countries. All other product names, logos, and brands are the property of their respective owners and are used for identification purposes only. Use of these names, logos, and brands does not imply endorsement.

No Endorsement

The mention of a company or product herein does not imply any endorsement by IOActive of that company or product. Nor does it imply any endorsement by the company or product manufacturer of IOActive.

No Commercial Relationship

The mention of a company or product herein does not imply any commercial relationship exists, has existed, or will exist between IOActive and that company or product manufacturer.

Publicly Available Material

All source material referenced in this publication was obtained from the Internet without restriction on use.

Fair Use

This paper is intended for educational purposes only. It contains copyrighted material, the use of which has not always been specifically authorized by the copyright owner. We are making such material available in our efforts to advance understanding of cyber safety and security in the aviation industry. Only small portions of the original works are being used and those could not be used easily to duplicate the original work. This material is distributed without profit to those who have expressed a prior interest in receiving the included information for the purposes of criticism, comment, news reporting, teaching, scholarship, education, and research. We believe this constitutes a fair use of any such copyrighted material as provided for in section 107 of the Copyright Act of 1976.

Reader's Note

This is a complex, highly technical topic coming from the intersection of two highly technical, complex fields. In order to make this work more accessible to readers who primarily spend their time working in only either aviation or cybersecurity, we have chosen to explain concepts and technologies that the average practitioner in one field would already know intimately, but that the average practitioner in the other field would be unlikely to know. Our hope is this over-communication of somewhat basic concepts in each field helps make this paper more accessible to both aviation and cybersecurity practitioners.

Introduction

In 2008, the Dreamliner was presented as the world's first e-Enabled commercial airplane.[2] Boeing certainly introduced an impressive new set of functionalities, enabling the vast majority of the components to be highly integrated with and connected to regular systems, such as onboard maintenance, data-load, and the Crew Information System (CIS).

In order to achieve this degree of integration, the 787's different aircraft data domains were conceived to operate in a common network, without being physically isolated. In practical terms, this meant there were physical and potential logical network paths between components with varying levels of criticality, for example, the In-Flight Entertainment System and Avionics.

In view of this novel design, the FAA issued a Special Conditions for the Boeing 787, as the applicable airworthiness regulations did not contain adequate or appropriate safety standards for such a scenario.

> *"The proposed architecture of the 787 is different from that of existing production (and retrofitted) airplanes. It allows new kinds of passenger connectivity to previously isolated data networks connected to systems that perform functions required for the safe operation of the airplane. Because of this new passenger connectivity, the proposed data network design and integration may result in security vulnerabilities from intentional or unintentional corruption of data and systems critical to the safety and maintenance of the airplane"*[3]

Although Special Conditions are a common regulatory artifact, this situation obviously raised some concerns within the security industry, which faded out once the Dreamliner certified.

In September 2018, a publicly accessible Boeing server was identified using a simple Google search, exposing multiple files.

Careful analysis of the contents revealed the server was actually leaking a repository of files that contained portions of the firmware running on the Crew Information System/Maintenance System (CIS/MS) and Onboard Networking System (ONS) for the Boeing 787 and 737 models respectively. This included documents, binaries, and configuration files. Additionally, a Linux-based Virtual Machine used to allow engineers to access part of the Boeing's network access was also available.

The research presented in this paper is based on the analysis of information from public sources, collected documents, and the reverse engineering work performed on the 787's

[2] https://www.boeing.com/commercial/aeromagazine/articles/qtr_01_09/article_05_1.html

[3] https://www.federalregister.gov/documents/2008/01/02/E7-25467/special-conditions-boeing-model-787-8-airplane-systems-and-data-networks-security-isolation-or

CIS/MS firmware, which has been developed by Honeywell, based on a regular (non-avionics, non-certified, and non-ARINC-653-compliant) VxWorks 6.2 RTOS (x86) running on a Commercial Off The Shelf (COTS) CPU board (Pentium M). Essentially, it is a common embedded device that is both physically and logically connected to the 787's avionics network.

Live testing of the issues described herein has been not performed on a real aircraft.

The work we were able to do to confirm these vulnerabilities was necessarily limited. Normally, we would follow our reverse engineering, architecture review, and technical threat modeling with a focused penetration testing effort. We were unable to do so in this case, as we had no means to access a 787 aircraft or a 787 lab environment in a safe and responsible manner to perform the typical final verification of the exploitability of our findings. Performing any testing without a safe environment was never considered.

In this paper, IOActive has documented our detailed attack paths and component vulnerabilities to describe the first plausible, detailed public attack paths to effectively reach the avionics network on a commercial airplane from either non-critical domains, such as Passenger Information and Entertainment Services, or even external networks.

Upon conclusion of the analysis, Boeing and Honeywell confirmed that these vulnerabilities are present in the 787's Core Network codebase; however, the official response IOActive received from Boeing was that they do not consider our reported findings to be exploitable vulnerabilities, as they could not reproduce these flaws.

In addition, Boeing stated that they have mitigations in place that prevent the vulnerabilities from being exploited; however, they were unwilling to share those details with IOActive. IOActive found this to be deeply disappointing, since this prevents us from independently validating the exploitability of the identified vulnerabilities. Without a 787, a 787 lab environment, or an explanation of the controls, IOActive is unable to confirm Boeing's claims of compensating controls or mitigations for these software vulnerabilities.

As a result, we hope that a determined, highly capable third party can safely confirm that these vulnerabilities are not exploitable due to mitigation controls not visible to us during this analysis. We are confident owners and operators of these aircraft would welcome such independent validation and verification.

We believe as strongly in safety as we do in security. We provide these detailed findings herein so that all stakeholders, members of the security industry, and affected entities can form their own judgment as to the exploitability and impact of these confirmed software vulnerabilities.

Background

In a January 2008 Wired published an article[4] titled “FAA: Boeing’s New 787 Maybe Vulnerable to Hacker Attack.”

The author of the article, Kim Zetter, recorded Boeing’s response as, “Boeing spokeswoman Lori Gunter said the wording of the FAA document is misleading, and that the plane's networks don't completely connect. Gunter wouldn't go into detail about how Boeing is tackling the issue but says it is employing a combination of solutions that involves some physical separation of the networks, known as ‘air gaps,’ and software firewalls. Gunter also mentioned other technical solutions, which she said are proprietary and didn't want to discuss in public.”

In 2016, a DHS-led team successfully accomplished a “remote, non-cooperative, penetration” [5] against a legacy Boeing 757 with the aircraft on the ground at the airport in Atlantic City, New Jersey within 48 hours of getting access to the aircraft. While some high-level results of this penetration testing were presented at the 2017 CyberSat Summit in Tysons Corner, Virginia, the technical details, including attack paths and proof of concept code, remain classified as of the publishing of this paper.

Boeing’s response to the results of this assessment was, “We firmly believe that the test did not identify any cyber vulnerabilities in the 757 or any other Boeing aircraft.”[6]

Unfortunately, since no technical details, proof of concept code, or detailed attack paths were shared, there was no opportunity for independent assessment of these conflicting claims.

Responsible Disclosure Process

Overall, the parties (Boeing and its Tier 1 suppliers) involved in the responsible disclosure process for this research project behaved in a highly responsive manner through the disclosure process. IOActive was quickly engaged with the appropriate point of contact in each organization and while IOActive was ultimately significantly disappointed in the level of reciprocity in information sharing at the end of the disclosure process, all parties remained highly responsive at least until the publishing of the paper. The communication was regular, detailed, and mostly constructive. This highly responsive communication stood in sharp contrast to another high-impact disclosure process that IOActive ran concurrently with this one.

[4] https://www.wired.com/2008/01/dreamliner-security/

[5] https://www.aviationtoday.com/2017/11/08/boeing-757-testing-shows-airplanes-vulnerable-hacking-dhs-says/

[6] https://www.ainonline.com/aviation-news/air-transport/2018-02-01/boeing-757-hacked-dhs-test

Everything IOActive observed indicated that the involved parties took this disclosure very seriously and allocated a level of resources to reviewing IOActive's disclosure that was consistent with an appropriate level of due care.

Disclosure Timeline

Initial disclosure of the public availability of the firmware was made on September 26, 2018 to Boeing. The primary point of contact at Boeing responded in under two hours and handed the contact off to the proper personnel. After establishing a secure communication channel using PGP, the details were provided to Boeing on September 27, 2018. Boeing indicated they were already aware of the issue and had moved to resolve it and thanked IOActive for the report. This was an outstanding example of responsive communication and resolution with the entire process taking approximately 24 hours. Throughout this initial process, the interaction was professional, collaborative and productive.

IOActive began work on assessing the firmware retrieved from the publicly accessible server in April 2019. After the initial assessment, the vulnerable firmware was determined to have been produced by Honeywell Aerospace. Initial contact was made to them on May 9, 2019. The initial response took 31 minutes. After the establishment of a secure communication channel, the detailed findings were shared. They quickly performed an initial review and, after conducting an initial assessment, notified Boeing of the reported vulnerabilities and shared their initial analysis.

At this time, IOActive started communicating both with Boeing as well as Honeywell. The parties communicated weekly or more frequently as needed. IOActive regularly provided additional information as requested, including documentation of the attack paths, additional findings, and other clarifying information. Boeing and Honeywell kept IOActive updated on the progress of their analysis and lab testing of the identified issues. Boeing and Honeywell performed this analysis in a very compressed timeframe for such a complex systems-of-systems as the 787.

Upon conclusion of their analysis in July 2019, Boeing and Honeywell confirmed that these vulnerabilities are present in the 787's Core Network codebase; however, the official response IOActive received from Boeing was that they do not consider our reported findings exploitable vulnerabilities, as they could not develop a working proof of concept for these flaws. In addition, Boeing stated that they have mitigations in place that prevent the vulnerabilities from being exploited.

Disagreement in Assessment of Exploitability

In view of the discrepancies in the exploitability assessment and the lack of access to a 787 or 787 lab environment, IOActive requested additional information from Boeing in order to facilitate a series of technical conversations that could be used to further investigate this matter. Boeing requested a formalized agreement be in place before such details could be shared with IOActive. Although an agreement was signed, Boeing did not provide IOActive with the minimum technical details that are usually shared between the involved actors in order to develop a viable vulnerability disclosure process, such as:

Versions

Boeing did not share with IOActive the version number of the CIS/MS firmware they were using in their testing, despite the fact that this information was requested several times. This is a crucial part in any responsible vulnerability disclosure, even more important when discrepancies in the results exist, as is the case here.

Testing Plan

During the vulnerability coordination process, IOActive did not have any visibility of the tests, methodologies, proof-of-concept code, exploitation techniques, or any technical details in general terms, that Boeing and their partners implemented during their internal evaluation of the vulnerabilities. To help address this situation, IOActive offered to assist Boeing in reproducing these vulnerabilities at their own controlled environment. Unfortunately, Boeing declined.

Mitigations

Boeing communicated to IOActive that there are certain built-in mitigations that, from their point of view, prevent these vulnerabilities from being successfully exploited. IOActive was unable to locate or validate the existence of such mitigations in the CIS/MS firmware version we analyzed. When asked, Boeing declined to answer whether these mitigations might have been added in a later version. A comprehensive technical explanation on the lack of mitigations in the CIS/MS firmware can be found in the 'Exploitability Assessment' section of this paper.

Boeing 787 Overview

In order to facilitate the reader's understanding of the security boundaries, attack scenarios, and vulnerabilities referenced in this paper, this section introduces part of the 787's architecture and systems that are relevant to this research.

Common Core System

As opposed to the federated avionics architectures, which make use of distributed avionics functions that are packaged as self-contained units, Integrated Modular Avionics (IMA)[7] architectures employ a high-integrity, partitioned environment that hosts multiple avionics functions of different criticalities on a shared computing platform. Boeing engineers went one step further and developed the Common Core System (CCS) for the 787, a further enhancement based on an open IMA avionics technology.

Essentially the CCS is a hardware/software platform that provides computing, communication, and input-output (I/O) services for implementing real-time embedded systems, known as hosted functions.

Multiple hosted functions share the platform resources within a virtual system environment enforced by partitioning mechanisms that are implemented as part of the platform design, relying on a VxWorks 653[8,9] OS.[10]

This virtual system partitioning environment guarantees that hosted functions are isolated from each other, so it supports highly critical applications but also lower levels of application integrity. For instance, DO-178B[11] Level-A software may coexist in the same physical shared resource with a Level-E application.

[7] https://www.aviationtoday.com/2007/02/01/integrated-modular-avionics-less-is-more/

[8] https://en.wikipedia.org/wiki/ARINC_653

[9] https://www.windriver.com/products/product-overviews/vxworks-653-product-overview-multi-core/

[10] https://www.windriver.com/customers/customer-success/aerospace-defense/boeing/

[11] https://en.wikipedia.org/wiki/DO-178B

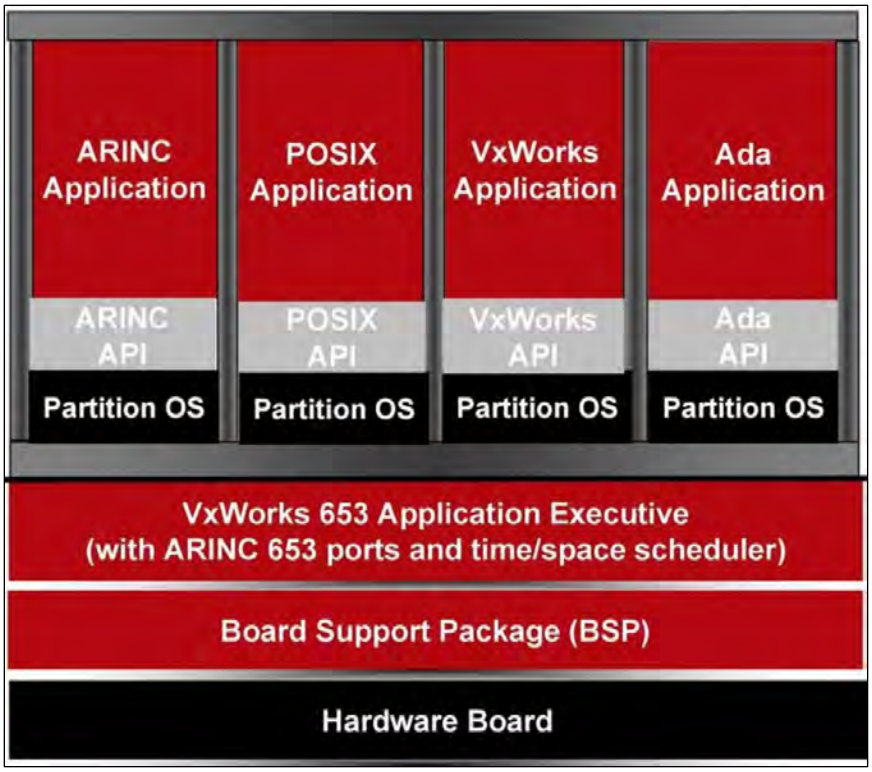


Figure 1. VxWorks 653 Architecture[12]

Ideally, the applications cannot interfere with each other, regardless of faults that may occur within the hosted functions or the platform resources, which are predetermined and communicated to the platform components via loadable configuration files usually in either XML or proprietary binary formats.

Within the CCS we can find the following major components:

- General Processing Modules (GPMs) to support functional processing needs
- Remote Data Concentrators (RDCs) to support system analog signals, analog discrete signals, and serial digital interfaces (CAN bus[13], A429[14], etc.)
- Avionics Full Duplex (A664-P7) Switched Ethernet[15] network for communication between platform elements

[12] http://www.artist-embedded.org/docs/Events/2007/IMA/Slides/ARTIST2_IMA_WindRiver_Wilson.pdf

[13] https://en.wikipedia.org/wiki/CAN_bus

[14] https://en.wikipedia.org/wiki/ARINC_429

[15] https://pdfs.semanticscholar.org/5db4/b539ed7bdec182448ac8d7219db12a8bbc12.pdf

These elements can be packaged as Line Replaceable Units (LRUs)[16] or in module or card form, which then can be grouped within cabinets or integrated LRUs. As a result, the CCS is made up of:

- Two (2) Common Computing Resource (CCR) cabinets
- The Common Data Network (CDN)
- 21 RDCs

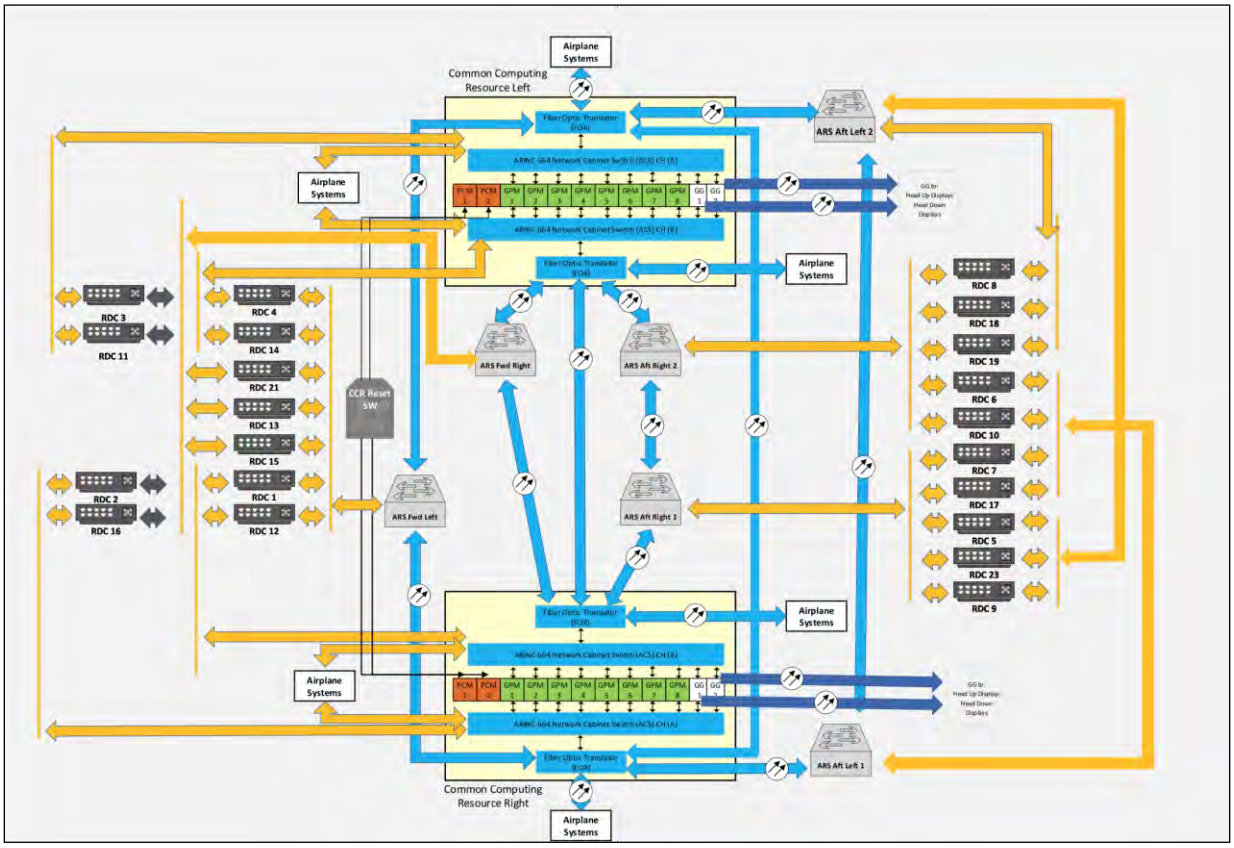


Figure 2. CCS Architecture

[16] https://en.wikipedia.org/wiki/Line-replaceable_unit

Common Computing Resource Cabinets

Each CCR cabinet has:

- Two (2) Power Conditioning Modules (PCMs)
- Eight (8) General Processing Modules (GPMs)
- Two (2) ARINC 664-P7 network Cabinet Switches (ACSs)
- Two (2) Fiber Optic Translator Modules (FOXs)
- Two (2) Graphic Generators (part of the Display and Alert Crew System)

Figure 3. Boeing 787 CCR Cabinet[17]

Each GPM is an independent computing platform that hosts airplane systems operational software and provides the hosted applications a partitioned environment based on the ARINC 653 standard. However, each GPM has the same hardware and core operating system.

[17] https://bioage.typepad.com/.a/6a00d8341c4fbe53ef0162fbf813b6970d

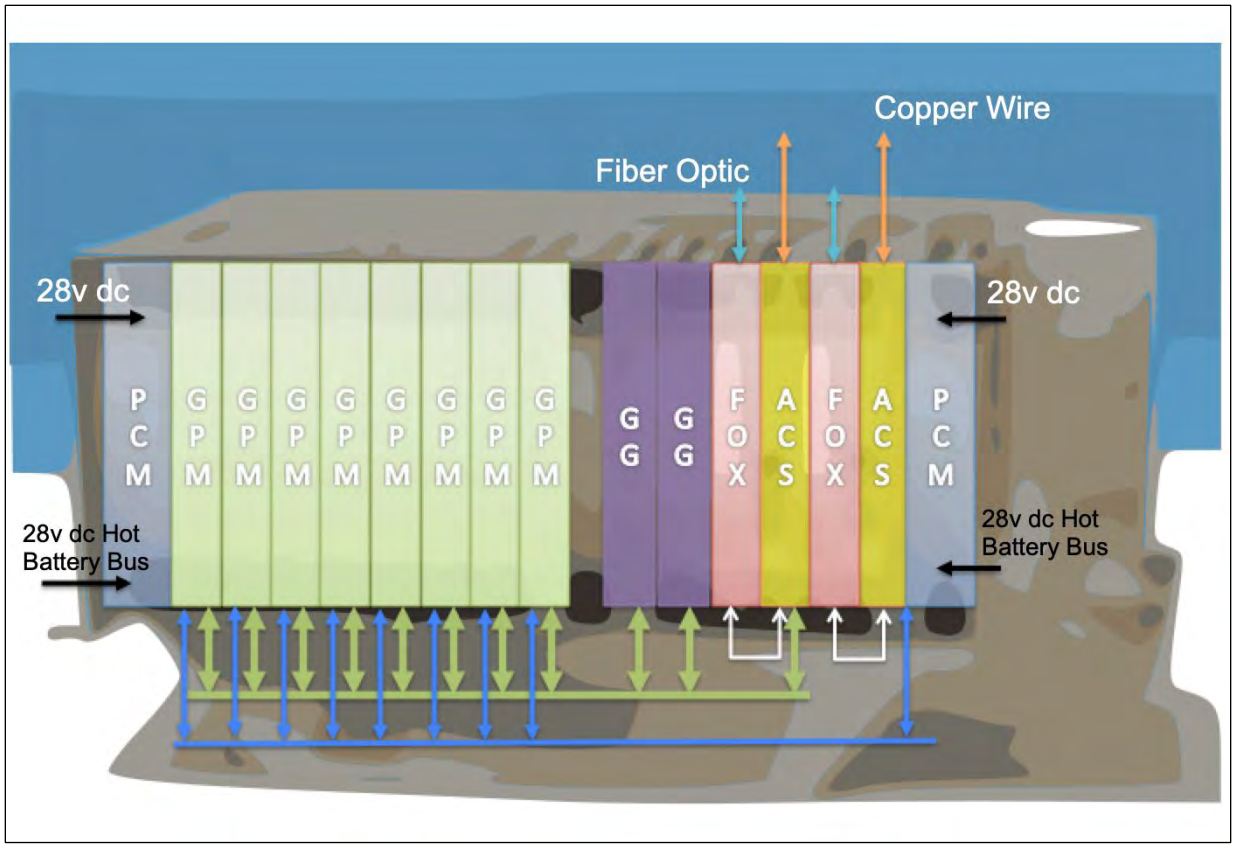


Figure 4. CCR Cabinet Schema

The GPMs in these CCR cabinets run the following hosted functions:

- Cabin Air Temperature Control System
- Equipment Cooling System
- Integrated Cooling System
- Remote Power Distribution System (RPDS)
- Power Distribution Panels (PDPs)
- Generator/Bus Power Control Units (GCU/BPCU)[18]
- Low Pressure System
- Power Electronics Cooling System
- Communication Management Function

[18] https://s3.amazonaws.com/public-inspection.federalregister.gov/2015-10066.pdf

- Switches - Flight Deck and Control Panels
- Circuit Breaker Indication and Control
- Electrical Power Distribution and Control
- Engine Fire Protection System
- Cargo Fire Protection System
- Fuel Quantity System
- Hydraulic System Control
- Wheel Well Fire Detection System
- Engine Anti-Ice Indications
- Cabin Air Compressor Inlet Ice Protection System
- Window Heat System
- Display Crew Alerting System
- Landing Gear Indication and Control
- Lighting Systems
- Thrust Management Function
- Flight Management Function
- Water and Waste Systems
- Airplane Conditioning Monitoring Function
- Central Maintenance Computing System
- Nitrogen Generation System
- Door Indication and Control

Common Data Network

The CDN is a high-integrity digital data network. It uses both fiber optic cable and copper wire and moves system information between the various airplane systems connected to it, either directly or through ACSs, FOXs, or RDCs.

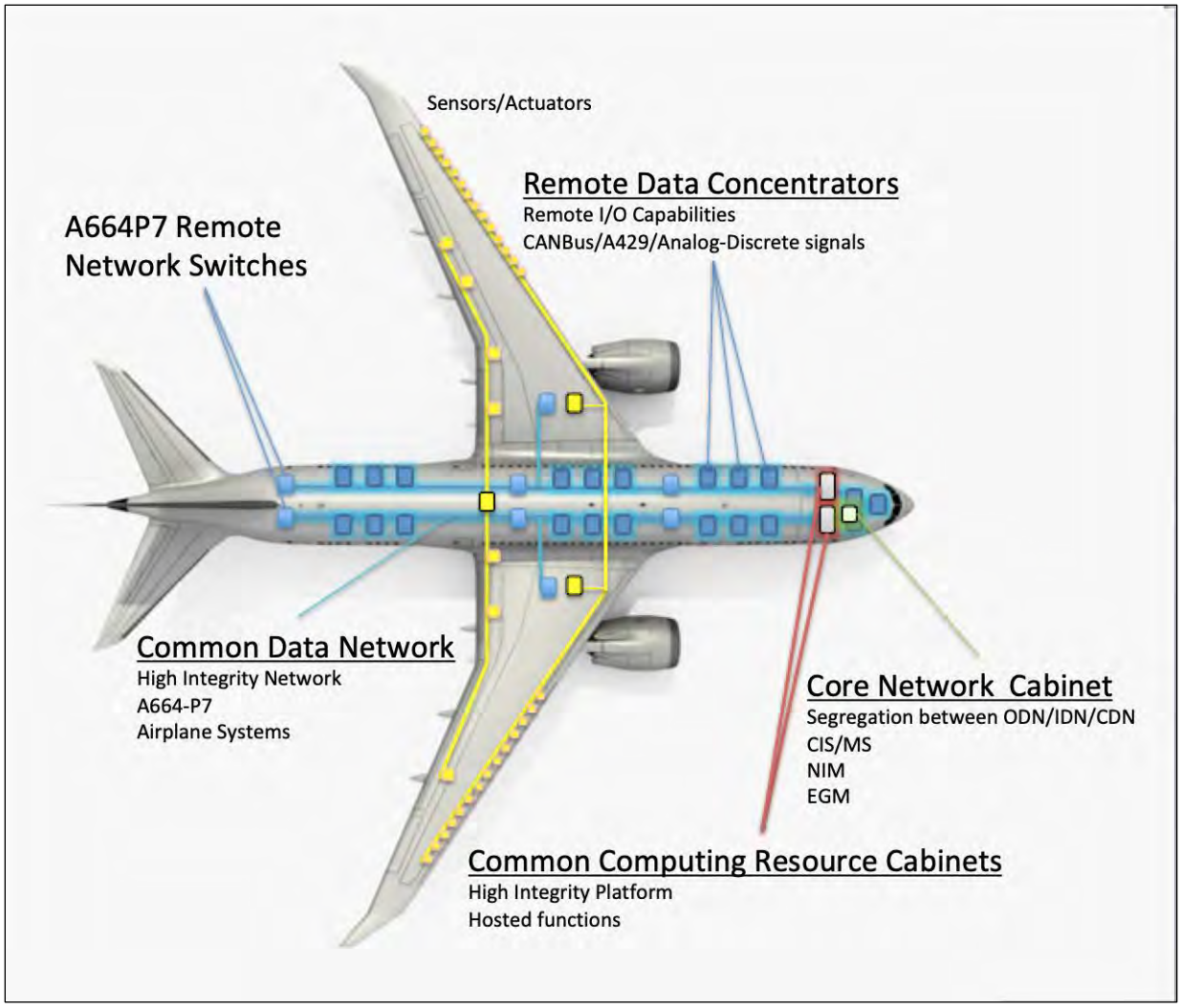


Figure 5. 787 Network Architecture

As in a common A664-P7 architecture, it is comprised of the network end system hosted in each connecting end node and multiple network switches. The network has been designed in a dual-channel switched star topology with each end node having a redundant, full duplex point-to-point connection with two independent communication pathways (A and B channels).

The CDN is the backbone of communication for the Boeing 787.

Remote Data Concentrators

There are 21 RDCs in the CCS.

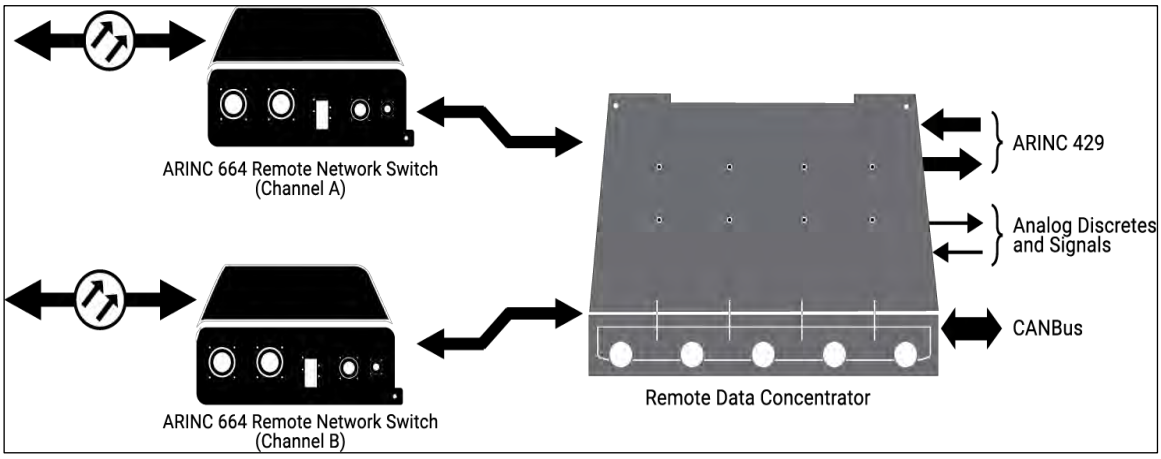


Figure 6. Remote Data Concentrators

These RDCs provide the interface between airplane systems that do not have the ability to support ARINC 664 in the CDN.

The RDCs convert these signals to ARINC 664 data and vice versa, thus effectively acting as a gateway for a variety of analog devices, such as sensors or valves, ARINC 429 buses, and CAN subnets.

From an A664-P7 perspective, these RDCs map:

- Analog signals to parameters
- A429 to communication ports
- CAN bus to both parameters and communication ports

Crew Information System – Maintenance System

The CIS provides the flight crews, airlines, mechanics, engineers, and maintenance personnel with access to data for flight operations and maintenance functions. The CIS essentially implements the interface between the 'outside world' and the CCS, thus enabling the 'system of systems' design that is associated with an e-Enabled aircraft.

The main component of the CIS is the Core Network Cabinet (CNC), which encompasses the following elements:

- Network Interface Module (NIM)
- Ethernet Gateway Module (EGM)
- Controller Server Module (CSM)
- Crew Information System/Maintenance System File Server Module (CIS/MS)
- Additional File Server Modules (FSM) (Optional)

Figure 7 depicts the architecture of the CIS, showing the three networks involved:

- ODN – Open Data Network
  From a security perspective, the ODN is the less trusted of these three networks as it provides the interface to connect a plethora of potentially hostile devices, including the In-Flight entertainment system, SATCOM system and airport terminal wireless network. The ODN routing and access control is implemented by the EGM.
- IDN – Isolated Data Network
  This network provides connectivity to a series of secure devices with routing and access control being implemented by the CIS/MS
- CDN – Common Data Network
  As previously explained, the Common Data Network is the backbone network of the 787, to which the airplane systems are connected. The NIM determines which traffic coming from the ODN/IDN should be allowed into the CDN.

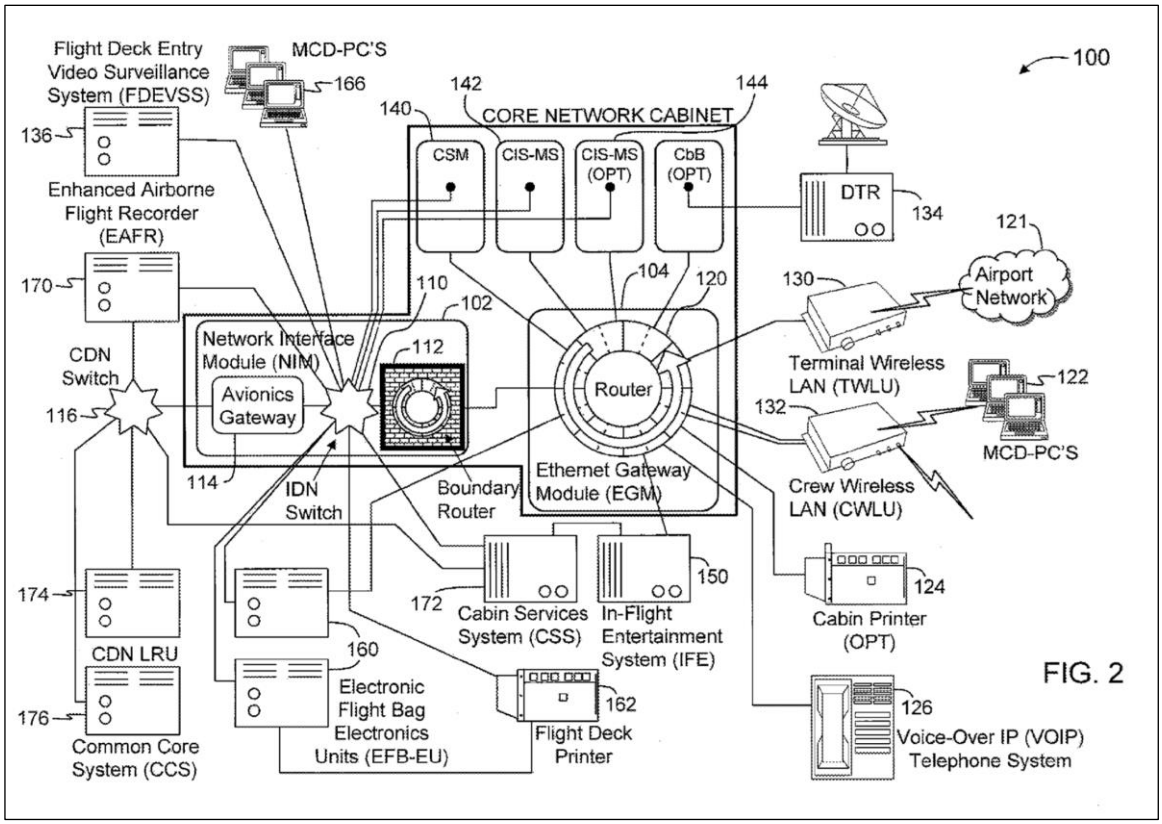


Figure 7. CIS Architecture [19]

At this point it is crucial to highlight that, according to this design, there is no physical isolation between the ODN and CDN. This effectively means that if the security controls implemented in the EGM, NIM or CIS/MS fail, there is a physical path to reach the CDN (Aircraft Control Domain) from untrusted aircraft data domains (Passenger Information and Entertainment Domain). Therefore, it seems reasonable that this specific part of the design played a fundamental role in the decision to issue the aforementioned Special Conditions by the FAA.

> *"[…] Therefore, special conditions are imposed to ensure that security, integrity, and availability of the aircraft systems and data networks are not compromised by certain wired or wireless electronic connections between airplane data buses and networks."*[20]

The research presented in the next section elaborates the scenarios and vulnerabilities that can enable such an attack.

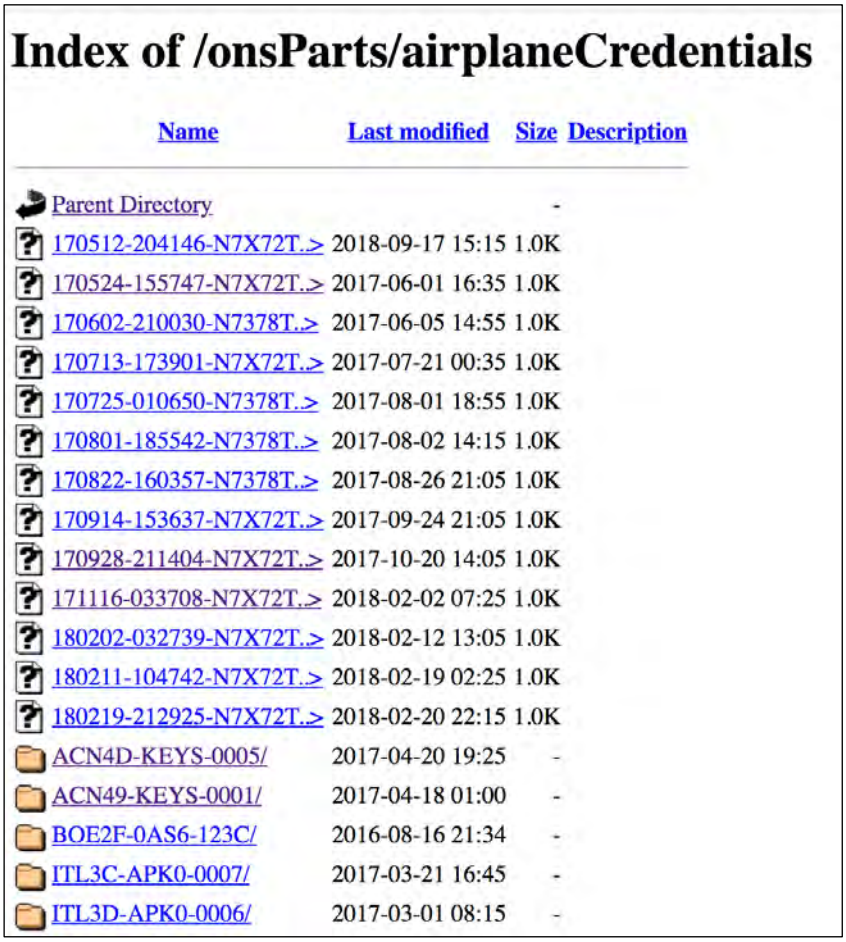
[19] https://patents.google.com/patent/WO2007117285A2/en

[20] https://www.federalregister.gov/documents/2008/01/02/E7-25467/special-conditions-boeing-model-787-8-airplane-systems-and-data-networks-security-isolation-or

Reverse Engineering the 787 Core Network

Introduction

In September 2018, a publicly available Boeing server containing a large number of Loadable Software Aircraft Parts (LSAPs) was discovered. It was found by using a specific Google search query, and did not require any kind of credentials.



Index of /onsParts/airplaneCredentials

Name	Last modified	Size	Description
Parent Directory		-	
170512-204146-N7X72T..>	2018-09-17 15:15	1.0K	
170524-155747-N7X72T..>	2017-06-01 16:35	1.0K	
170602-210030-N7378T..>	2017-06-05 14:55	1.0K	
170713-173901-N7X72T..>	2017-07-21 00:35	1.0K	
170725-010650-N7378T..>	2017-08-01 18:55	1.0K	
170801-185542-N7378T..>	2017-08-02 14:15	1.0K	
170822-160357-N7378T..>	2017-08-26 21:05	1.0K	
170914-153637-N7X72T..>	2017-09-24 21:05	1.0K	
170928-211404-N7X72T..>	2017-10-20 14:05	1.0K	
171116-033708-N7X72T..>	2018-02-02 07:25	1.0K	
180202-032739-N7X72T..>	2018-02-12 13:05	1.0K	
180211-104742-N7X72T..>	2018-02-19 02:25	1.0K	
180219-212925-N7X72T..>	2018-02-20 22:15	1.0K	
ACN4D-KEYS-0005/	2017-04-20 19:25	-	
ACN49-KEYS-0001/	2017-04-18 01:00	-	
BOE2F-0AS6-123C/	2016-08-16 21:34	-	
ITL3C-APK0-0007/	2017-03-21 16:45	-	
ITL3D-APK0-0006/	2017-03-01 08:15	-	

Figure 8. Directory Listing on the Exposed Server

The analysis of the files revealed the following contents:

- A Boeing Virtual Machine containing the cryptographic materials needed to establish a VPN connection to a specific Boeing network
- Firmware for the 787's Core Network Cabinet
- Firmware for the 737's ONS

The scope for this research is limited to the 787's CIS/MS firmware, which has been developed by Honeywell.

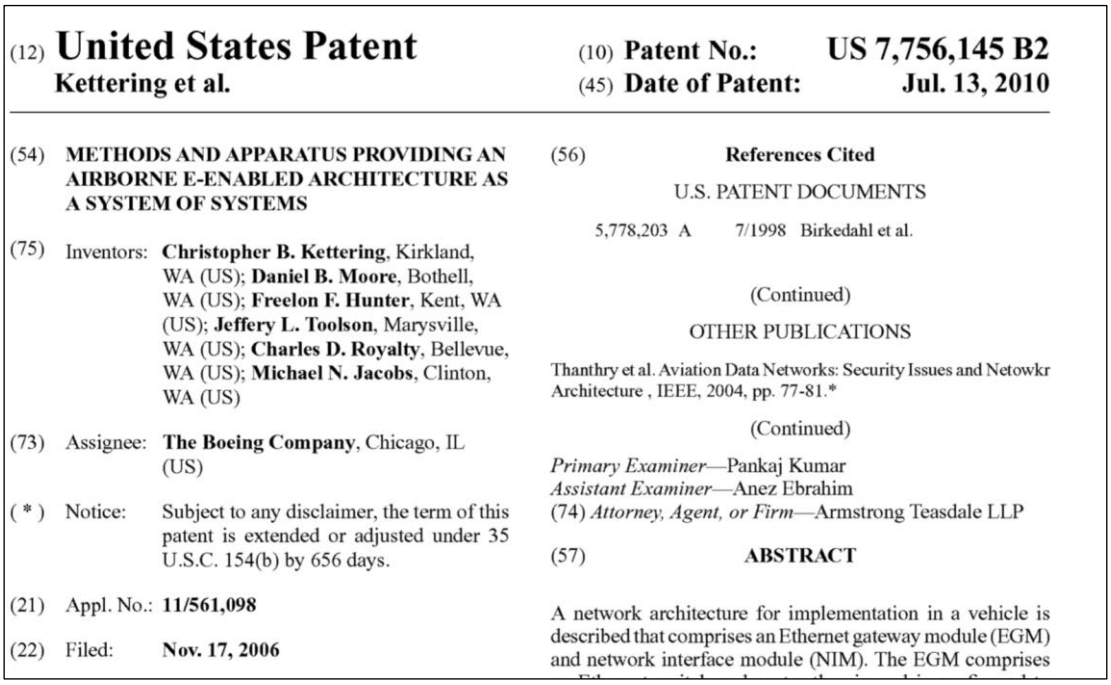
Methodology

Without physical access to the actual devices, the approach involved static analysis of the firmware via reverse engineering and a configuration review. As in previous research, the methodology essentially consisted of the following stages:

1. Information Gathering
    1.1 Documents, multimedia material, presentations, papers, press releases, patents, books, etc.
2. Reverse Engineering
    2.1 Identify the elements, components, and functionalities described in the patents
    2.2 Identify attack vectors
    2.3 Prioritize attack areas
    2.4 Find a minimum set of vulnerabilities required to demonstrate each of the attack scenarios described in 2.2
    2.5 Assess the exploitability and post-exploitation scenarios, which included reviewing the machine code for the presence of compiler-level mitigations.
    2.6 Evaluate the overall security posture of the in-scope elements taking information and knowledge gained in the previous phases into account

US Patent 7756145 B2

The Boeing patent[21] for the technology behind the CIS/MS is publicly available and was used as a reference to understand the system.



(12) **United States Patent**
Kettering et al.

(10) **Patent No.: US 7,756,145 B2**
(45) **Date of Patent: Jul. 13, 2010**

(54) **METHODS AND APPARATUS PROVIDING AN AIRBORNE E-ENABLED ARCHITECTURE AS A SYSTEM OF SYSTEMS**

(75) Inventors: **Christopher B. Kettering**, Kirkland, WA (US); **Daniel B. Moore**, Bothell, WA (US); **Freelon F. Hunter**, Kent, WA (US); **Jeffery L. Toolson**, Marysville, WA (US); **Charles D. Royalty**, Bellevue, WA (US); **Michael N. Jacobs**, Clinton, WA (US)

(73) Assignee: **The Boeing Company**, Chicago, IL (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 656 days.

(21) Appl. No.: **11/561,098**

(22) Filed: **Nov. 17, 2006**

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,778,203 A 7/1998 Birkedahl et al.

(Continued)

OTHER PUBLICATIONS

Thanthry et al. Aviation Data Networks: Security Issues and Netowkr Architecture , IEEE, 2004, pp. 77-81.*

(Continued)

Primary Examiner—Pankaj Kumar
Assistant Examiner—Anez Ebrahim
(74) *Attorney, Agent, or Firm*—Armstrong Teasdale LLP

(57) **ABSTRACT**

A network architecture for implementation in a vehicle is described that comprises an Ethernet gateway module (EGM) and network interface module (NIM). The EGM comprises

Figure 9. Patents Provided Valuable Research Information

The following patents were also found comprehensively covering certain functionalities implemented in the CIS/MS/CNC.

- US8165930B2[22] – Crate Tool
- US8321083[23] – Aircraft Maintenance Laptop
- US8442751[24] – Onboard Electronic Distribution System
- US20160205724A1[25] – System and Method For Connecting Aircraft To Networks On Ground
- US7599194 - Methods and apparatus for a redundant board assembly

[21] https://patents.google.com/patent/WO2007117285A2/en
[22] https://patents.google.com/patent/US8165930B2
[23] https://patents.google.com/patent/US8321083
[24] https://patents.google.com/patent/US8552751
[25] https://patents.google.com/patent/US20160205724A1

Attack Surface of the Core Network Cabinet

The patent includes a schematic of the CNC, as well as how it is connected to both onboard and offboard networks and devices. This provides valuable information to start characterizing threat scenarios.

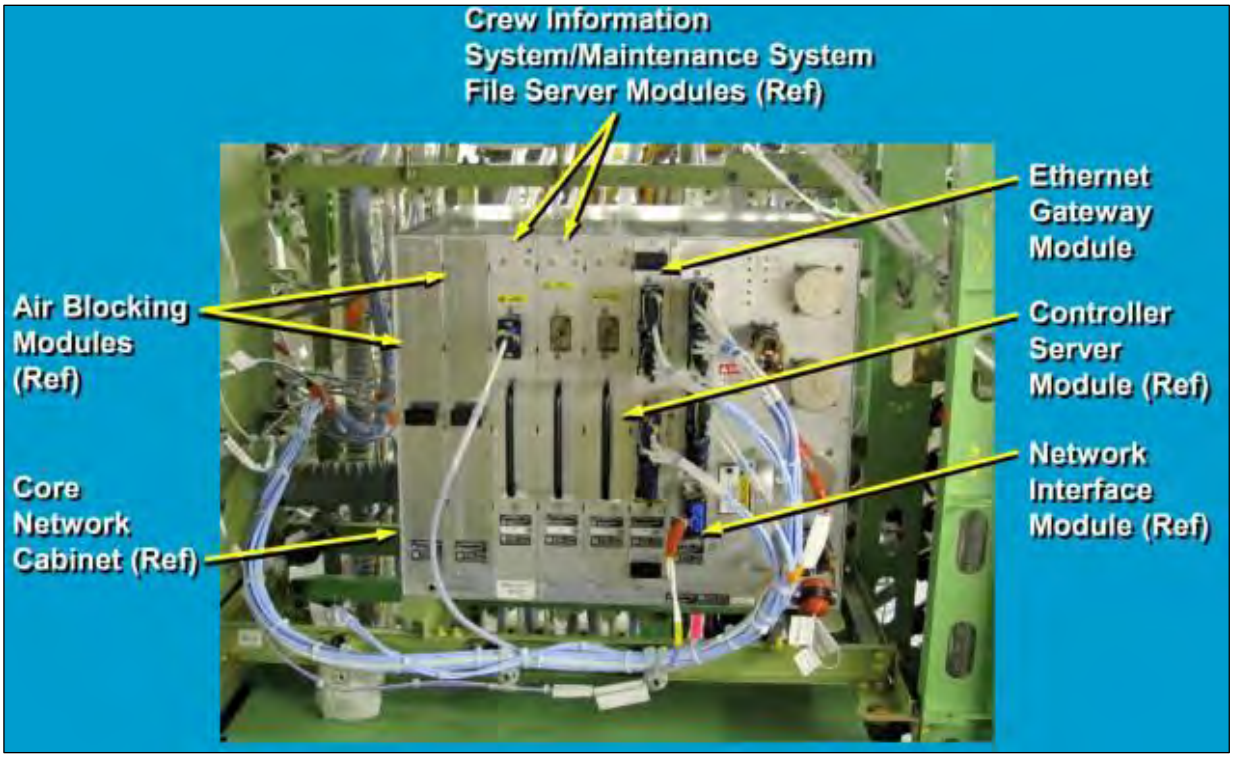


Figure 10. CNC

The patent clearly indicates that the NIM and EGM are where most packet filtering and network segregation functionalities are implemented.

> *"The architecture provides segregation between network devices in the IDN and CDN related to operation and navigation of the vehicle, and network devices in the ODN."*
>
> *US Patent 7756145 B2*

From a security perspective, the EGM, CIS/MS, and NIM, are preferred targets, as they represent the security controls that provide segregation between the ODN, IDN, and CDN.

Although multiple attack vectors are possible, Figure 11 depicts one of the worst-case scenarios: a threat coming from the ODN trying to reach the CDN. It is worth mentioning that once an attacker has compromised the CIS/MS it is not yet possible to reach every single system connected to the CDN but only those that have been defined in the NIM's

A664 End System. NIM's access to the CDN[26] network is limited by the rules that are loaded into the GE's NIM Network ASIC through a proprietary configuration file (`es_config.bin`), which is generated using a DO-178B Level-A tool: 'ESBIN'.

This situation will be elaborated upon later on.

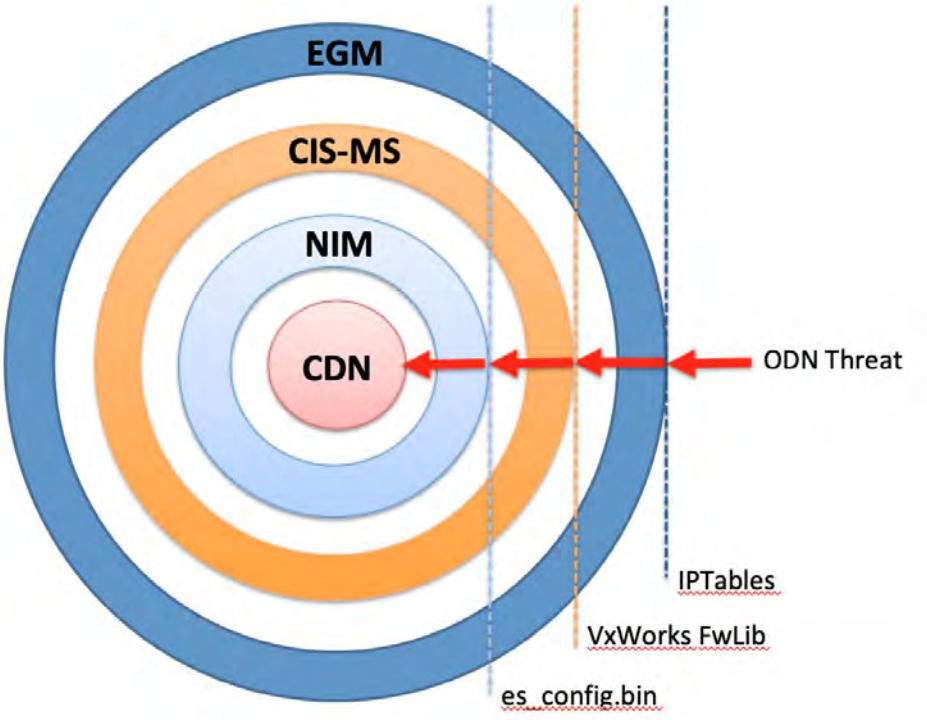


Figure 11. A Worst-case Scenario Affecting the CNC

In order to achieve this goal, the attacker needs to go through the security controls implemented in the EGM, the CIS/MS and finally the NIM's A664 End System.

This table summarizes the security controls that allow the aircraft domain segregation and the required actions from the attacker's perspective.

Table 1. Summary of Security Controls

Component	Action Required	Network Security Control	Exploit Required
EGM	Find a Firewall rule that allows access to a vulnerable CIS/MS service.	Linux IPTables [27]	No

[26] http://www.ieee802.org/1/files/public/docs2015/TSN-Schneele-AFDX-0515-v01.pdf

[27] https://netfilter.org/projects/iptables/index.html

Component	Action Required	Network Security Control	Exploit Required
CIS/MS	Remote code execution against a vulnerable network service (VxWorks' RTPs).	VxWorks Packet Filtering Library 'FwLib'	Yes
NIM's A664 End System	Leverage the configured virtual links to abuse supported functionalities (Data Loading, Maintenance, etc.).	'es_config.bin'	No

Figure 12 summarizes the generic steps an attacker needs to perform to reach the CDN from the ODN.

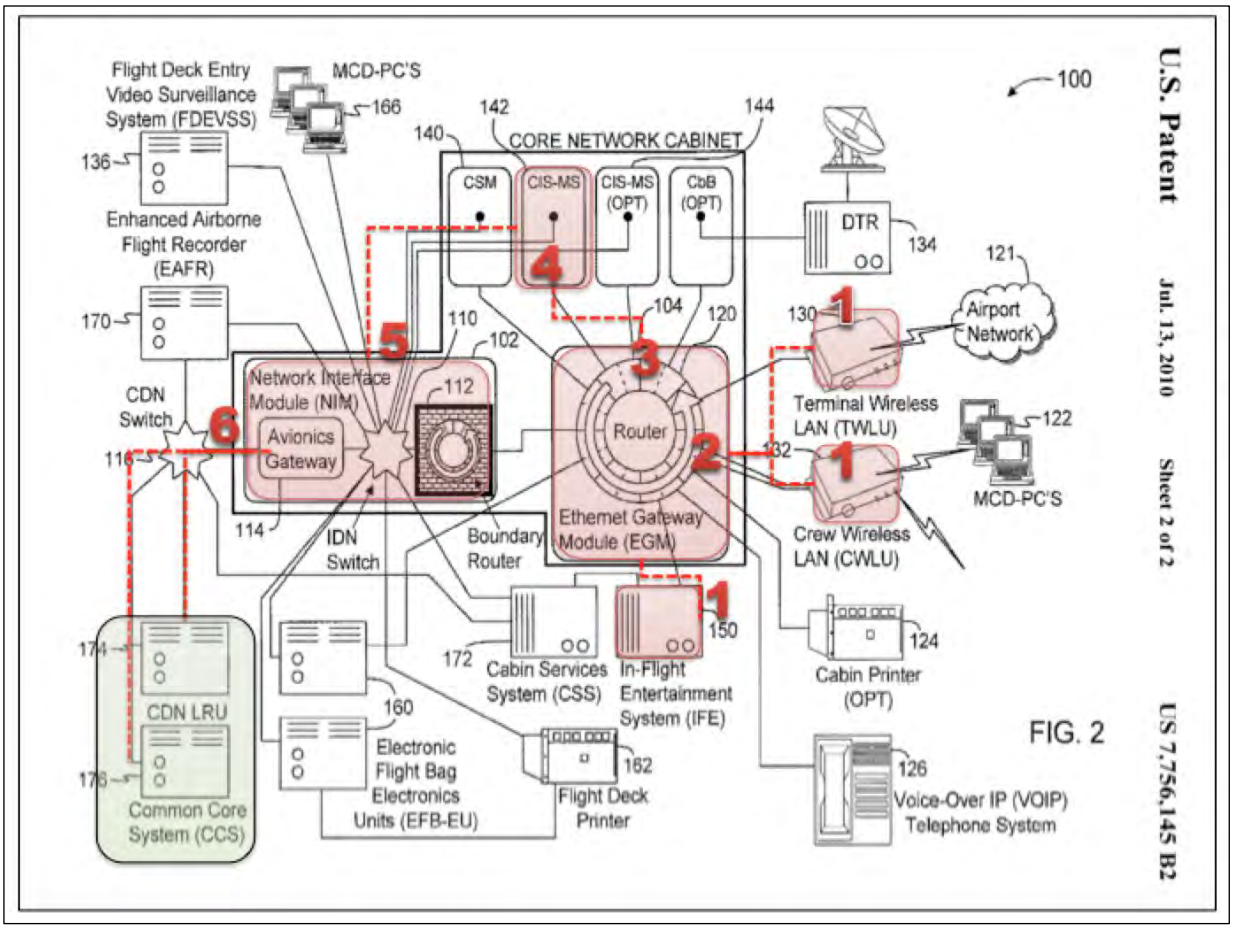


Figure 12. Basic Attack Surface

The following steps correspond to the red-colored labels in Figure 12:

1. These components are located in the ODN. They represent both onboard and offboard attack vectors. The In-Flight Entertainment System (IFE) can be considered a valid onboard attack vector, while the TWLU is used to interface with external networks such as the Gatelink822 infrastructure.

As a result the malicious actor needs to compromise only one of these components.

2. The EGM implements routing and security for the ODN. It is based on a series of Linux's IPTables rules that are located in the 'S24egmcfg' (DO-178B Level E).
3. The attacker does not need to compromise the EGM. This step requires finding a network path in the EGM firewall that allows the attacker to reach a vulnerable service in the CIS/MS.
4. In this step, the attacker exploits a vulnerability in one of the exposed CIS/MS (VxWorks 6.2) services (RTPs) to compromise the CIS/MS. This allows the attacker to take control of the core functionalities running in the CIS/MS, such as OBEDS, ODLF, MVPN, Firewall, etc.
5. The attacker now is able to go through the NIM and reach the CDN.
6. The attacker can communicate with those components connected to the CDN via established A664-P7 virtual links required for normal system operation according to the rules defined in `es_config.bin`. This includes Maintenance and Data Loading operations for safety critical units.

The upcoming sections will provide detailed information about the vulnerabilities and techniques that make this attack possible.

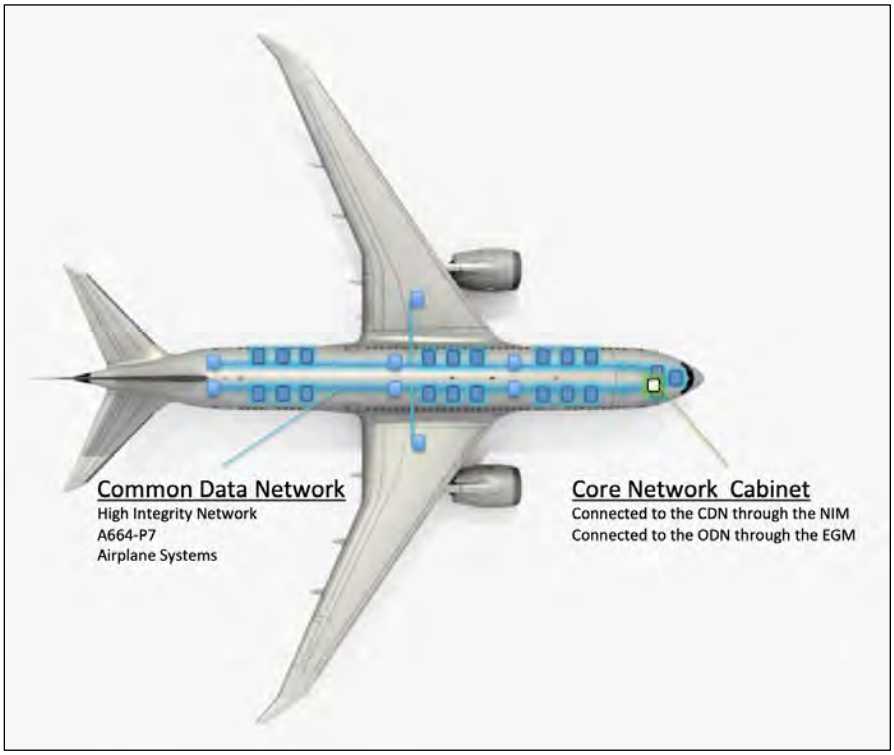


Figure 13. The CNC Provides an Interface to the CDN

Firmware

The collected LSAP files were unpacked, reconstructed, analyzed, and classified to identify the relevant materials, such as binaries or configuration files.

The dominant architecture for the CNC's core components is x86. As explained, our efforts were focused on the EGM, CIS/MS and NIM's End System configuration.

- The EGM is a Linux-based ZNYX ZX4500. In this case, the priority was to analyze the firewall rules to understand what kind of traffic is allowed through the EGM.
- The CIS/MS is implemented on a non-certified/non-ARINC-653-compliant VxWorks 6.2 kernel with a Pentium BSP (CPB4612 board) and Real-Time Processes (RTPs)[28] enabled. It is designed to support interconnection of the 787's onboard and offboard computer networks and data. It does not include any kind of compiler-level buffer overflow mitigations.
- NIM's End System is a GE's Aviation ASIC

  An 664-P7 End System (ES) defines how and when a virtual link is established between a source ES and a destination ES. From the attacker's perspective, everything regarding the A664-P7 except the source and destination ES would be transparent, preventing it from being a target. It is worth mentioning that we are not exploiting any A664-P7 characteristic but merely leveraging the configured virtual links to abuse the functionalities the system was originally designed for.

[28] https://learning.windriver.com/real-time-processes

Versions

The firmware IOActive analyzed dates back to early 2016.

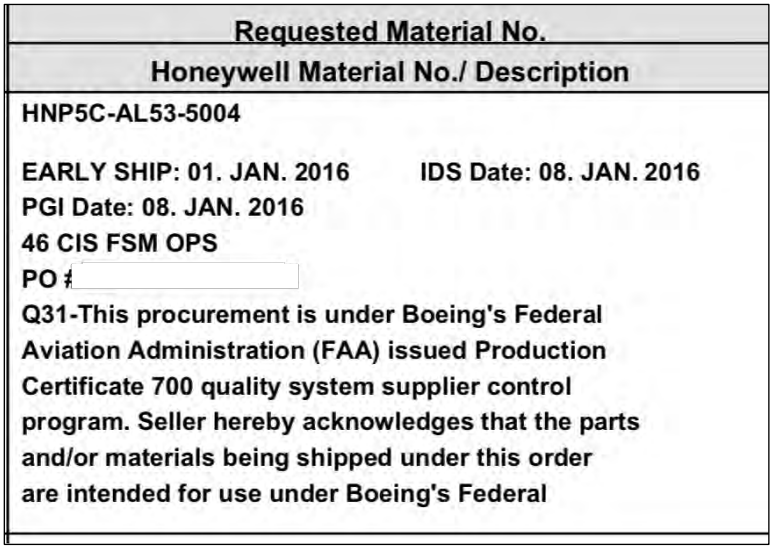


Figure 14. CIS FSM OPS

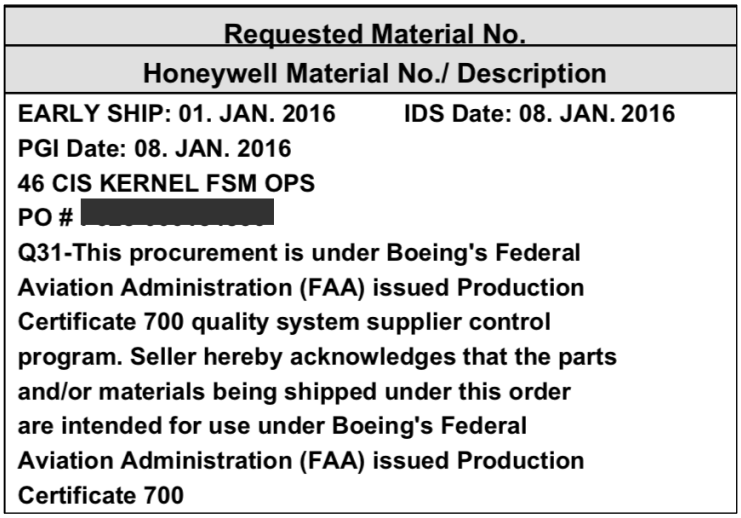


Figure 15. CIS KERNEL FSM OPS

The compilation date seems to be late September 2015. The firmware was built using GCC 3.3.2 and VxWorks 6.2 (x86):

```
 GCC: (GNU) 3.3.2 20030904 (Wind River vxworks-6.2) (built
20050915)
```


For a CPB4612[29] Board:

```
C:/WindRiver/vxworks-6.2/target/config/CPB4612-vx62_fsm
```

The majority of our reverse engineering efforts were spent on the following CIS/MS binaries, which implement the core functionalities.

From the reverse engineering perspective, we find a common VxWorks based system. The only aspect that may need to be highlighted is the 'OBEDS.vxe' RTP, which is a stand-alone executable based on a JamaicaVM[30], a Java VM designed for embedded systems which is widely used in the aviation sector.

VxWorks Kernel	
User-Mode	
Real-Time Processes:	Shared Libraries:
1. FBM.vxe 2. FTS_Manager.vxe 3. MSPE.vxe 4. OBEDS.vxe 5. ODLF.vxe 6. bmt.vxe 7. fsmTgtLdr.vxe 8. ftpd.vxe 9. mtf_main.vxe 10. mtf_rtp.vxe 11. omls.vxe 12. osm.vxe 13. rexec_server.vxe 14. wlanmf_rtp.vxe	1. ACP.so 2. AMI.so 3. DiskUtilities.so 4. DisplayUtilities.so 5. FCCS.so 6. FSMAAircraftVerification.so 7. JSON.so 8. LDI.so 9. Messaging.so 10. OBEDSInterface.so 11. OrderedList.so 12. SNMP.so 13. cisUtil.so 14. mtfIOUtilities.so 15. ossAccessors.so

[29] https://manualsbrain.com/en/manuals/1189567/
[30] https://www.aicas.com/cms/en/JamaicaVM

CIS/MS Vulnerabilities

Surprisingly, it was possible to find hundreds of references to insecure function calls, such as `strcpy`, `sprintf`, and `strcat`, in the custom parts of the VxWorks kernel as well as in the RTPs and shared libraries. These insecure programming patterns are certainly one of the most significant sources of vulnerabilities for the CIS/MS, although fortunately, these kinds of flaws can be easily fixed and not every one will likely matter from a security perspective due to its location in the trust model or practical exploitability. However, a series of additional vulnerable patterns that are usually harder to spot and fix were found as well, such as integer overflows, buffer overflows, denial of service, out-of-bound reads and writes, memory corruptions, etc.

Without deviating from the original approach, this paper will mainly cover a minimum set of vulnerabilities that allow the CIS/MS to be compromised from the ODN, thus allowing an attacker to reach the CDN.

FTS_Manager.vxe - TFTP Opcode Stack Overflow

FTS_Manager.vxe implements multiple file transfer services for different functionalities, such as OBEDS. Among these services, it provides a TFTP server running in multiple instances at different ports.

A remote unauthenticated attacker can exercise the vulnerable execution path.

When the TFTP server receives a request, it uses the opcode (first two bytes in a TFTP request) as an index into an array of strings, which contains the usual TFTP[31] operations, such as `RRQ`, `ACK`, and `DATA`. It then uses `sprintf` to generate a log entry stored at `log_buffer`, which is a buffer of 0x400 bytes allocated in the stack.

[31] https://tools.ietf.org/html/rfc1350


```
value = recvfrom(
          serversocket,
          &requestbuffer,
          0x200u,
          0,
          (struct sockaddr *)&clientaddr,
          &clientaddrlength);
if ( value == -1 )
{
  sprintf(&log_buffer, "TFTP --> could not read on TFTP port %d", fs_listen_port[server_instance]);
  rtpLog(3, 0, &log_buffer);
  goto LABEL_107;
}
opcode = ((unsigned __int8)requestbuffer << 8) | ((requestbuffer & 0xFF00) >> 8);
strncpy(&fileforoptneg, (const char *)&requestbuffer + 2, 0x80u);
v20 = 0;
sprintf(
  &log_buffer,
  "%s --> %s Request Received for file %s from ",
  &fs_tftp_task_name[20 * server_instance],
  &opcode_string[5 * opcode],
  &fileforoptneg);
```

Figure 16. Stack Overflow

Exploitability

In this insecure operation, there are two parameters that are controlled by the attacker in the TFTP request.

- Destination File
- Opcode

In order to gain arbitrary code execution, the attacker would need to overflow `log_buffer` with a total of 0x700 bytes. The key to exploiting this vulnerability is the ability to reference a string large enough using the `opcode` index. Taking into account that `opcode_string` is located at `.data` section, it is possible to dereference an arbitrary length string within adjacent 0x4FFFB bytes, which covers the remaining `.data` section and also part of the `.bss`.

```
.data:080DF040                          public opcode_string
.data:080DF040 opcode_string            db 'INV',0
.data:080DF044                          db    0
.data:080DF045 aRrq                     db 'RRQ',0
.data:080DF049                          db    0
.data:080DF04A aWrq                     db 'WRQ',0
.data:080DF04E                          db    0
.data:080DF04F aData                    db 'DATA',0
.data:080DF054 aAck                     db 'ACK',0
.data:080DF058                          db    0
.data:080DF059 aErr_0                   db 'ERR',0
.data:080DF05D                          db    0
.data:080DF05E aOack                    db 'OACK',0
.data:080DF063                          db    0
```

Figure 17. Operations Array

The analysis of the variables within this range revealed multiple targets that can be used to write `log_buffer` beyond its limits. As a result, an attacker can exploit this

vulnerability by building a ROP (assuming a conservative stack exploitation approach) chain using TFTP's `Destination File` to control EIP and required registers, thus effectively executing arbitrary code within the `FTS_Manager`'s RTP context.

ODLF.vxe – Multiple Vulnerabilities

ODLF.vxe RTP provides the NIM with the Onboard Data Loading Functionalities (ODLF), which basically implements the Data Loader for the 787, based on the regular ARINC 615A and ARINC 665-2/3 standards[32]. This is crucial functionality as it allows handling software updates for onboard LRUs.

From the security perspective, it is a highly valuable target for two reasons:

- This process exposes a remote attack surface through its TFTP implementation and also by parsing different kinds of files.
- It requires a special condition across the different networks in order to accept connections from the different Data Loaders but also has to be allowed to connect to the LRUs that will be updated.

As expected, the frequent use of insecure functions makes it trivial to find remote code execution vulnerabilities. The vulnerabilities below illustrate different insecure patterns.

TFTP RRQ/WRQ Filename Buffer Overflow

This part of the code handles RRQ/WRQ[33], and `inBuffer_ptr` points to this request. At address 0x8068664, `inBuffer_ptr` is incremented by two bytes to (opcode field size) in order to `strcpy` the `Destination File` to a local buffer of 0x100 bytes allocated in the stack `tmpstr`, resulting in a stack overflow.

[32] https://www.ijettjournal.org/2016/volume-37/number-5/IJETT-V37P244.pdf

[33] Read and write request packets respectively. See: https://www.freesoft.org/CIE/RFC/1350/5.htm


```
.text:0806865E loc_806865E:                             ; DATA XREF: .rodata:0808CFCC↓o
.text:0806865E                                          ; .rodata:0808CFD0↓o
.text:0806865E                  sub     esp, 8
.text:08068661                  mov     eax, [ebp+inBuffer_ptr]
.text:08068664                  add     eax, 2
.text:08068667                  push    eax             ; src
.text:08068668                  lea     eax, [ebp+tmpstr]
.text:0806866E                  push    eax             ; dest
.text:0806866F                  call    strcpy
.text:08068674                  add     esp, 10h
.text:08068677                  sub     esp, 0Ch
.text:0806867A                  lea     eax, [ebp+tmpstr]
.text:08068680                  push    eax             ; s
.text:08068681                  call    strlen
.text:08068686                  add     esp, 10h
.text:08068689                  mov     [ebp+nFilenameLen], eax
.text:0806868C                  sub     esp, 8
.text:0806868F                  push    2Eh             ; c
.text:08068691                  lea     eax, [ebp+tmpstr]
.text:08068697                  push    eax             ; s
.text:08068698                  call    strchr
.text:0806869D                  add     esp, 10h
.text:080686A0                  mov     [ebp+ptrTmp], eax
.text:080686A3                  cmp     [ebp+ptrTmp], 0
.text:080686A7                  jz      loc_8068883
.text:080686AD                  sub     esp, 8
.text:080686B0                  mov     eax, [ebp+ptrTmp]
.text:080686B3                  inc     eax
.text:080686B4                  push    eax             ; src
.text:080686B5                  lea     eax, [ebp+extstr]
.text:080686BB                  push    eax             ; dest
.text:080686BC                  call    strcpy
.text:080686C1                  add     esp, 10h
.text:080686C4                  sub     esp, 8
.text:080686C7                  push    offset aLui_0   ; "LUI"
.text:080686CC                  lea     eax, [ebp+extstr]
.text:080686D2                  push    eax             ; s1
.text:080686D3                  call    strcmp
```

Figure 18. TFTP RRQ/WRQ Opcode Handler

Exploitability

The number of bytes available to read from the TFTP socket is a minimum of 0x200 bytes, which allows an attacker to effectively control the EIP and those registers required to initiate a ROP chain.

A remote unauthenticated attacker can exercise the vulnerable execution path. It is worth mentioning that any compromised LRU that is about to be updated may also trigger this vulnerability, as ODLF acts as a server/client.

duParseLUSFile Memory Corruption

`duParseLUSFile` is implemented in `diskUtils.so`, but the vulnerable code path can be exercised from `ODLF.vxe`

When parsing status files, `duParseLUSFile` does not properly validate the number of headers a LUS (ARINC Load Upload Status) file may contain (`numberHeaders`). This value is a 16-bit integer that is directly read from the LUS buffer (the buffer which is under the attacker's control).


```
for ( idxf = bytesReadc; idxf < bytesReadc + 2; ++idxf )
  fileData->numberHeaders += buffer[idxf] << (8 - 8 * (idxf - bytesReadc));
bytesReadd = bytesReadc + 2;
for ( hdrIndex = 0; hdrIndex < fileData->numberHeaders; ++hdrIndex )
{
  fileData->files[hdrIndex].headerNameLength = 0;
  for ( idxg = bytesReadd; idxg < bytesReadd + 1; ++idxg )
    fileData->files[hdrIndex].headerNameLength += buffer[idxg] << -8 * (idxg - bytesReadd);
  v4 = bytesReadd + 1;
  strncpy(fileData->files[hdrIndex].headerName, (const char *)&buffer[v4], fileData->files[hdrIndex].headerNameLength);
  fileData->files[hdrIndex].headerName[fileData->files[hdrIndex].headerNameLength] = 0;
  bytesReade = fileData->files[hdrIndex].headerNameLength + v4;
  fileData->files[hdrIndex].partNumberLength = 0;
```

Figure 19. Memory Corruption

The function will then use the number of headers as an index into a fixed array of files (internal `A665LUSFileType` structure) allocated in the stack of the caller, which is received as an argument to the function. The memory corruption happens when this index points to out-of-bounds memory.

Exploitability

The attacker can corrupt the stack buffer in a solid way, using controllable values from the LUS file, which allows the attacker to effectively control the EIP and those registers required to initiate a ROP chain.

A remote unauthenticated attacker can exercise the vulnerable execution path. It is worth mentioning that any compromised LRU that is about to be updated may also trigger this vulnerability as the ODLF acts as a server/client.

FsmTgtLdr.vxe – LUH Part Number Stack Overflow

This RTP handles the Software Update for the own Core Network FSMs. When parsing `.LUH` files (ARINC Load Upload Headers), the part number length is not properly checked. Later on, this part number (directly from the LUH file) is copied to a local buffer in the stack, causing the corruption.

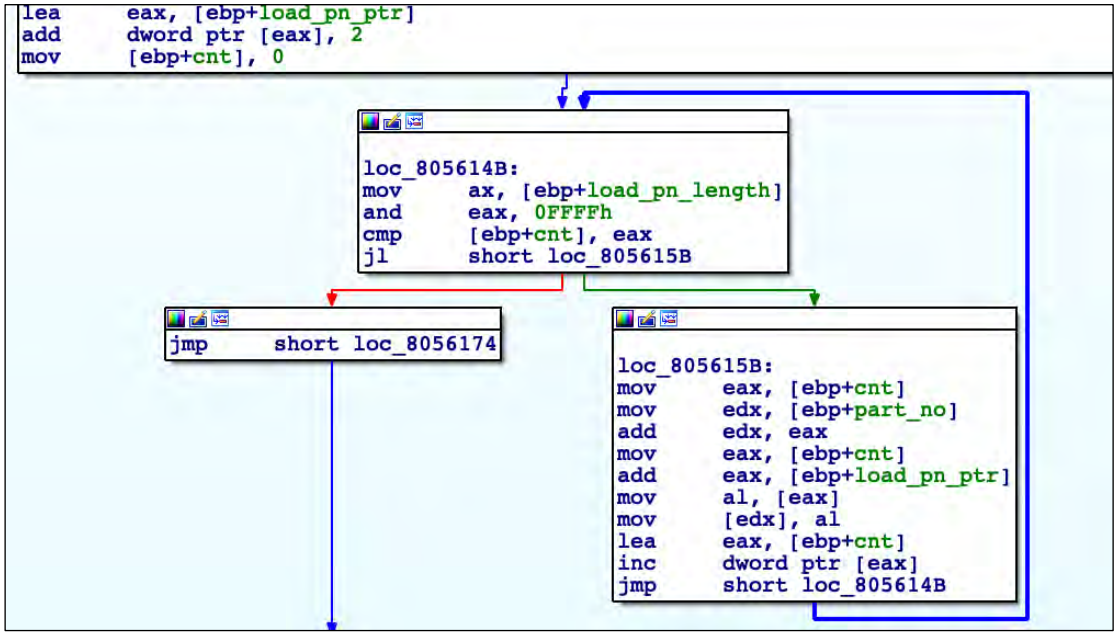


Figure 20. LUH Stack Overflow

Exploitability

The attacker is able to corrupt the stack buffer in a solid way, using controllable values from the LUH file, which allows the attacker to effectively control the EIP and those registers required to initiate a ROP chain.

A remote unauthenticated attacker can exercise the vulnerable execution path.

VxWorks – Insecure Syscall Handlers Privilege Escalation

The previous vulnerabilities can be used to compromise one of the RTPs running in user-mode; however, in order to fully compromise the CIS/MS, code must be executed as kernel.

The CIS/MS's VxWorks kernel implements a custom set of syscalls, `FSMSSYSTEM`, that are invoked in the RTPs via a callgate.

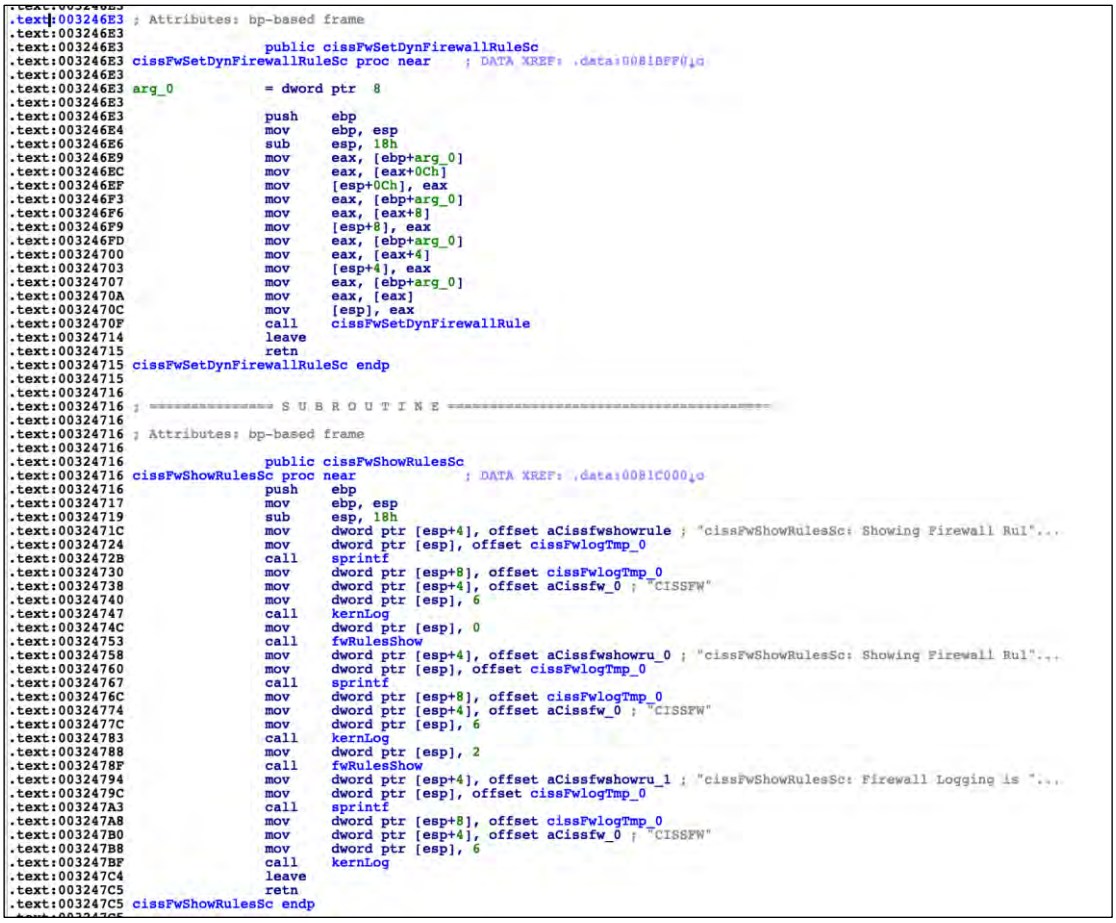
```
.data:0081C020                  public syscallGroupTbl
.data:0081C020 syscallGroupTbl dd 0                     ; DATA XREF: rtpSysctlSyscall+4D↑r
.data:0081C020                                          ; rtpSysctlSyscall+AE↑r ...
.data:0081C024 dword_81C024    dd 0                     ; DATA XREF: rtpSysctlSyscall+11E↑r
.data:0081C024                                          ; syscallGroupRegister+12E↑w ...
.data:0081C028                  align 10h
.data:0081C030                  dd offset FSMSYSTEMRtnTbl
.data:0081C034                  db  26h ; &
```

Figure 21. VxWorks Syscall Groups Table

There are several problems with the implemented syscall handlers:

- They are not validating any pointer received from user-mode, so it is possible to read/write arbitrary kernel memory.
- They use insecure functions and other insecure patterns, which can be used to trigger different kinds of vulnerabilities.

Figure 22 illustrates these problems. `cissFwSetByDynFirewallRuleSc` is the syscall 0x224 handler, and it is clear that no parameter validation is performed. Additionally, the implementation of `cissFwSetByDynFirewallRule`, which enables RTPs to add arbitrary firewall rules to the CIS/MS packet filter, also uses insecure `sprintf` functions that may result in buffer overflows. As a result, an attacker with access to a compromised RTP may exploit these vulnerabilities to gain kernel privileges without requiring a reset.



```
.text:003246E3 ; Attributes: bp-based frame
.text:003246E3
.text:003246E3                 public cissFwSetDynFirewallRuleSc
.text:003246E3 cissFwSetDynFirewallRuleSc proc near     ; DATA XREF: .data:0081DFF0↓o
.text:003246E3
.text:003246E3 arg_0           = dword ptr  8
.text:003246E3
.text:003246E3                 push    ebp
.text:003246E4                 mov     ebp, esp
.text:003246E6                 sub     esp, 18h
.text:003246E9                 mov     eax, [ebp+arg_0]
.text:003246EC                 mov     eax, [eax+0Ch]
.text:003246EF                 mov     [esp+0Ch], eax
.text:003246F3                 mov     eax, [ebp+arg_0]
.text:003246F6                 mov     eax, [eax+8]
.text:003246F9                 mov     [esp+8], eax
.text:003246FD                 mov     eax, [ebp+arg_0]
.text:00324700                 mov     eax, [eax+4]
.text:00324703                 mov     [esp+4], eax
.text:00324707                 mov     eax, [ebp+arg_0]
.text:0032470A                 mov     eax, [eax]
.text:0032470C                 mov     [esp], eax
.text:0032470F                 call    cissFwSetDynFirewallRule
.text:00324714                 leave
.text:00324715                 retn
.text:00324715 cissFwSetDynFirewallRuleSc endp
.text:00324715
.text:00324716
.text:00324716 ; =============== S U B R O U T I N E =======================================
.text:00324716
.text:00324716 ; Attributes: bp-based frame
.text:00324716
.text:00324716                 public cissFwShowRulesSc
.text:00324716 cissFwShowRulesSc proc near              ; DATA XREF: .data:0081C000↓o
.text:00324716                 push    ebp
.text:00324717                 mov     ebp, esp
.text:00324719                 sub     esp, 18h
.text:0032471C                 mov     dword ptr [esp+4], offset aCissfwshowrule ; "cissFwShowRulesSc: Showing Firewall Rul"...
.text:00324724                 mov     dword ptr [esp], offset cissFwlogTmp_0
.text:0032472B                 call    sprintf
.text:00324730                 mov     dword ptr [esp+8], offset cissFwlogTmp_0
.text:00324738                 mov     dword ptr [esp+4], offset aCissfw_0 ; "CISSFW"
.text:00324740                 mov     dword ptr [esp], 6
.text:00324747                 call    kernLog
.text:0032474C                 mov     dword ptr [esp], 0
.text:00324753                 call    fwRulesShow
.text:00324758                 mov     dword ptr [esp+4], offset aCissfwshowru_0 ; "cissFwShowRulesSc: Showing Firewall Rul"...
.text:00324760                 mov     dword ptr [esp], offset cissFwlogTmp_0
.text:00324767                 call    sprintf
.text:0032476C                 mov     dword ptr [esp+8], offset cissFwlogTmp_0
.text:00324774                 mov     dword ptr [esp+4], offset aCissfw_0 ; "CISSFW"
.text:0032477C                 mov     dword ptr [esp], 6
.text:00324783                 call    kernLog
.text:00324788                 mov     dword ptr [esp], 2
.text:0032478F                 call    fwRulesShow
.text:00324794                 mov     dword ptr [esp+4], offset aCissfwshowru_1 ; "cissFwShowRulesSc: Firewall Logging is "...
.text:0032479C                 mov     dword ptr [esp], offset cissFwlogTmp_0
.text:003247A3                 call    sprintf
.text:003247A8                 mov     dword ptr [esp+8], offset cissFwlogTmp_0
.text:003247B0                 mov     dword ptr [esp+4], offset aCissfw_0 ; "CISSFW"
.text:003247B8                 mov     dword ptr [esp], 6
.text:003247BF                 call    kernLog
.text:003247C4                 leave
.text:003247C5                 retn
.text:003247C5 cissFwShowRulesSc endp
```

Figure 22. Insecure Syscall Handler in VxWorks Kernel

Attack Scenarios

Scenario #1 – From IFE to CDN

As previously outlined, one of the worst-case scenarios is where an attacker coming from the IFE domain (ODN) is able to reach the CDN (Aircraft Control). Figure 23 illustrates how this is possible.

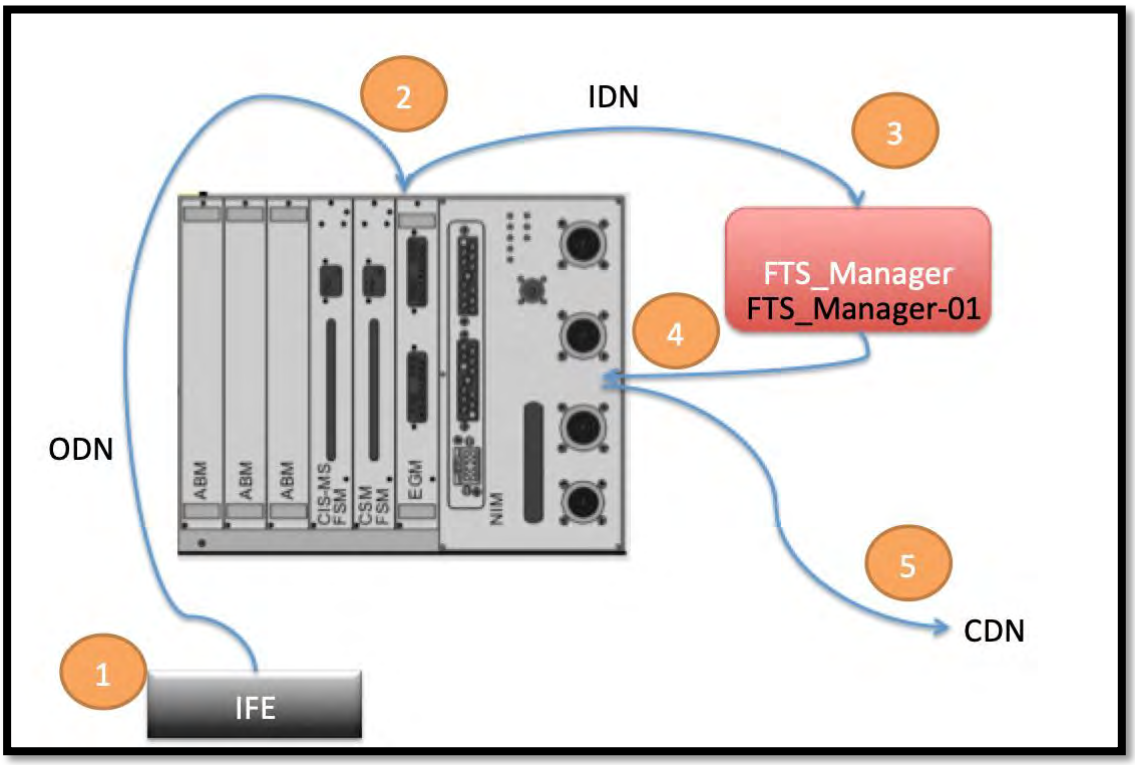


Figure 23. IFE to CDN Schema

The EGM can reach one of the TFTP server instances, running at port 16005, in the `FTS_Manager` RTP:

File: 'S54egmcfg'

```
# VLAN 140; In-Flight Entertainment System
zconfig zhp22 : vlan140=zre20
….
iptables -A IFE -j ACCEPT -i zhp22 -s 172.27.40.2 -d 172.24.10.12 -p
udp --sport 1024:65535 --dport 16005
```

172.27.40.2 - ife-router.odn.pnet

172.24.10.12 - cis-ms-active.idn.pnet

The NIM Boundary Router also allows this connection:

The following data is part of the 'AimCfg.xml'. This configuration file is generated based on the 787 Interface Control Document (ICD) in order to define the network paths through the NIM's Avionics Gateway and communication paths through the ODN/IDN boundary routers.

File: 'AimCfg.xml'

```
<!-- *** IFE FTS *** -->
        <!-- CABINET-CABIN_EQPMT_CENTER-IFE interfaces with APP-CIS MS
ACTIVE      (FTS Service) -->
        <Rule chain="FORWARD" target="ACCEPT">
            <Parameters>-i eth0 -p udp -s 172.27.40.2 --sport 1024:65535
-d 172.24.10.12 --dport 16005</Parameters>
            <Tag>Boundary router rule</Tag>
            <Sanction>Do-Nothing</Sanction>
        </Rule>
```

As a result, an attacker that has compromised the IFE would be able to reach the CDN via the following steps:

1. Attacker gains access to the IFE-Router, located in the ODN[34]
2. The EGM routes to the CIS/MS's `FTS_Manager` at TFTP port 16005 (PlaneNet TFTP Server)
3. Attacker exploits the TFTP vulnerability described in the `FTS_Manager`
4. The exploit uses the syscall 0x224 (`cissFwSetByDynFirewallRule`) to manipulate the firewall and unblock access to the CDN from the attacker's IP
5. The attacker can now reach the CDN

1. IOActive has previously published vulnerabilities in several in-flight entertainment systems

Scenario #2 – From an Arbitrary LRU to CDN

The ODLF is capable of updating an LRU's firmware through the ARINC615 protocol. The firewall rules that allow this kind communication, between the LRU and the ODLF Data Loading service, are defined both statically and dynamically based on the Loadable Dynamic Information (LDI).

The following static rules enable the Data Loading (TFTP Port 59/UDP) operation from the CIS/MS to the Electronic Flight Bag (EFB), Terminal Wireless LAN Unit (TWLU), Crew Wireless LAN Unit (CWLU), and Terminal Cellular Unit (TCU). The TWLU, CWLU and TCU all are connected to the ODN.

File: 'S54egmcfg'

```
iptables -A EFB -j ACCEPT -i zhp28 -s 172.24.10.12 -d 172.20.100.2 -p udp
--sport 16069 --dport 59
iptables -A EFB -j ACCEPT -i zhp17 -s 172.20.100.2 -d 172.24.10.12 -p udp
--sport 1024:65535 --dport 16069
…
iptables -A TWLU -j ACCEPT -i zhp28 -s 172.24.10.12 -d 172.27.60.2 -p udp
--sport 16063 --dport 59
iptables -A TWLU -j ACCEPT -i zhp28 -s 172.24.10.12 -d 172.27.60.2 -p udp
--sport 16063 --dport 1024:65535
…
iptables -A CWLU -j ACCEPT -i zhp28 -s 172.24.10.12 -d 172.20.30.2 -p udp
--sport 16048 --dport 59
iptables -A CWLU -j ACCEPT -i zhp16 -s 172.20.30.2 -d 172.24.10.12 -p udp
--sport 1024:65535 --dport 16048
…
iptables -A TCU -j ACCEPT -i zhp28 -s 172.24.10.12 -d 172.27.70.2 -p udp
--sport 16072 --dport 59
iptables -A TCU -j ACCEPT -i zhp28 -s 172.24.10.12 -d 172.27.70.2 -p udp
--sport 16072 --dport 1024:65535
```

If any of these LRUs is compromised, it would be possible to exploit a vulnerability in `ODLF.vxe` to gain access to the CIS/MS

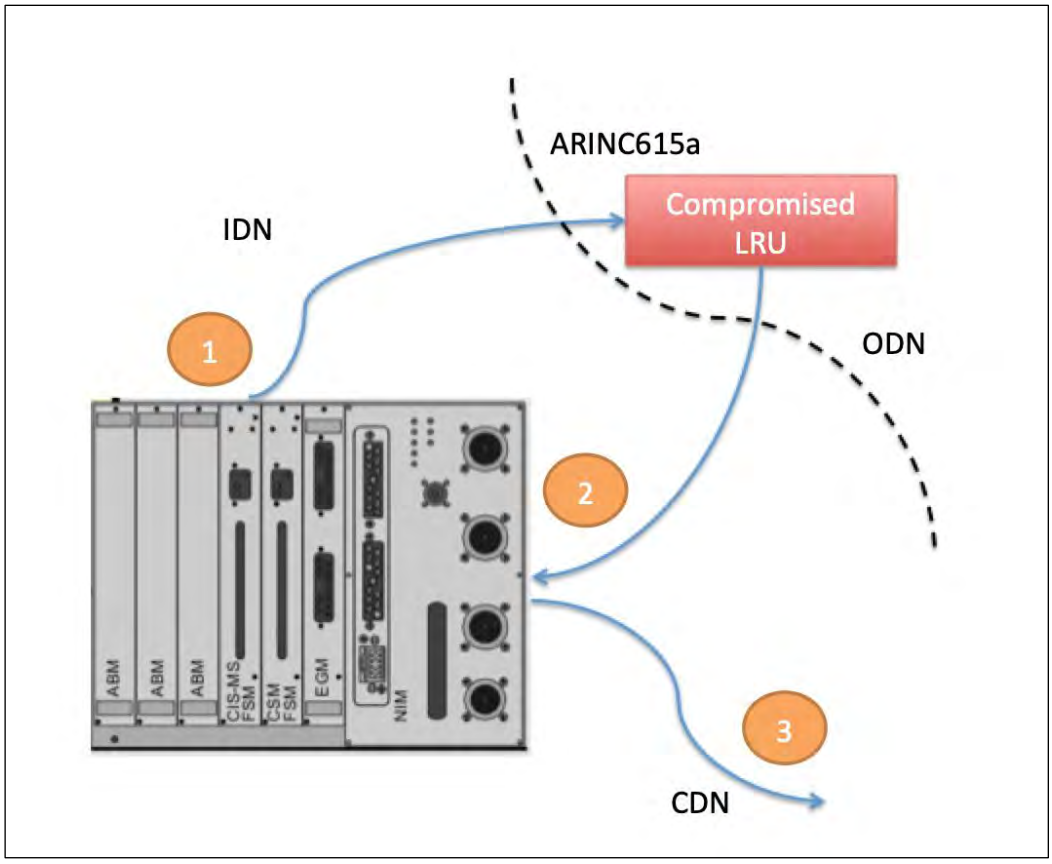


Figure 24. Compromised LRU to CDN Scenario

1. CIS/MS initiates a Data Loading operation against the compromised LRU.
2. The compromised LRU sends a malicious request exploiting one of the multiple vulnerabilities in the `ODLF.vxe` module.
3. The attacker can now execute a specific payload to allow access from the LRU to the CDN (e.g. by using the `cissFwSetDynFirewallRule` syscall).

Scenario #2.1 – Wireless LRU to CDN

A variant of the compromised LRU to CDN scenario involves a wireless LRU onboard the 787.

1. An onboard attacker gains access to the wireless WELS' WCU by exploiting a potential vulnerability or supply chain compromise. (IOActive has not evaluated this product and knows of no confirmed vulnerabilities in it)[35]
2. ODLF vulnerabilities can be exploited through the compromised WCU.
3. The attacker gains access to the CIS/MS and then to the CDN.

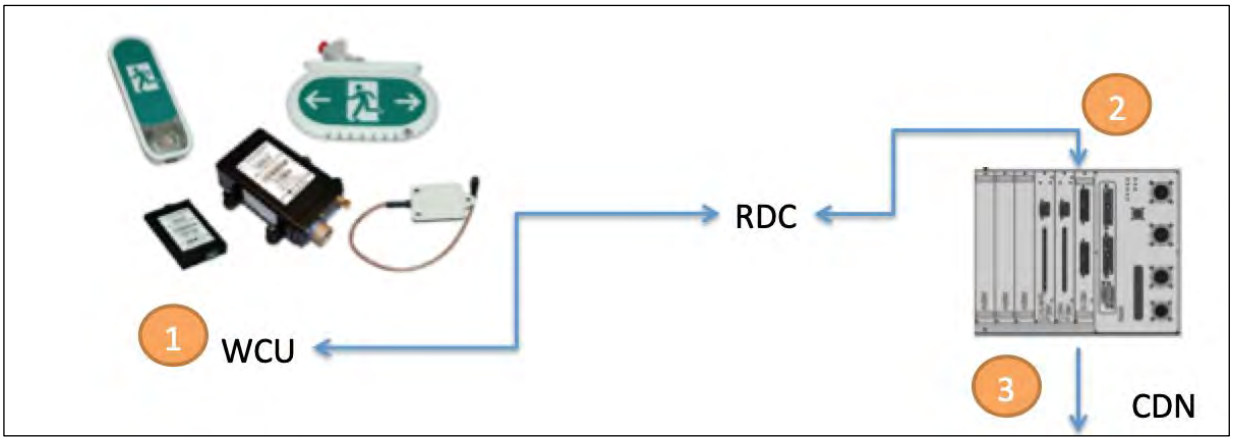


Figure 25. WELS Attack Scenario

File: 'AimCfg.xml'

```
<!-- PDT = [CAN Data Load]    ODLF_200kb_CAN_S1_Tx1 |Usg to UNIT-WELS
CONTROL-PRIME-DR 3L |Usg.R5_C2_ODLF_WKP_HFQI_10 |Usg -->
            <!--*** APP-CIS MS ACTIVE-1 |Usg to UNIT-WELS CONTROL-PRIME-
DR 3L |Usg ***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.42.165.10</CDN_Dest_IP>
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>66112</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62911</IDN_Source_UDP>
            </Uni_Tx
```

[35] https://www.securaplane.com/products/wireless/

Scenario #3 – External Network to CDN

1. An attacker compromises an Internet-facing vulnerable LSAP proxy server.[36]
2. The attacker controls LSAP repository/uplink-downlink requests (OBEDS[37])
3. The Gatelink822 Airport's local infrastructure may also expose an attack vector.[38]
4. The attacker reaches the IDN through the TWLU/CWLU EGM rules.
5. The attacker gains CDN access by exploiting any of the documented vulnerabilities.

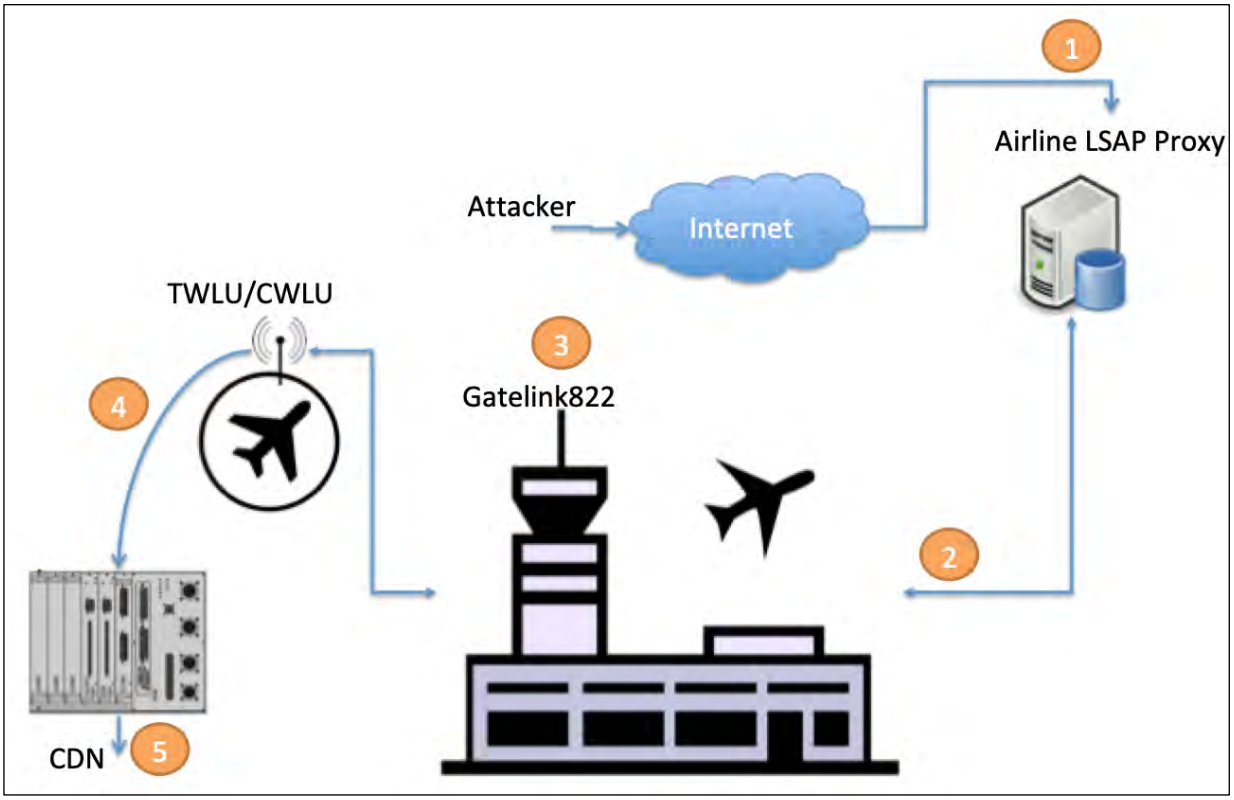


Figure 26. External Network to CDN Scenario

[36] IOActive discovered two vulnerable instances of Internet-accessible LSAP proxy servers belonging to airlines operating Boeing aircraft and shared the details with Boeing.

[37] Onboard Boeing electronic distribution system

[38] Gatelink822 infrastructure and reachability may vary between airports. As an example, in Terminal 4S of the Barajas Adolfo Suarez Airport in Madrid (Spain), the Gatelink822 SSID is publicly broadcast throughout the terminal.

Scenario #4 – Communication Link to CDN

1. TCU/SATCOM providers may assign a public IP that is exposed to the Internet.[39]
2. An attacker gains access to the TCU/SATCOM device.[40]
3. The attacker reaches the CIS/MS through the EGM rules for TCU/SATCOM interfaces (if any, as SATCOM may be optional)
4. The attacker gains CDN access by exploiting any of the vulnerabilities documented in the CIS/MS services.

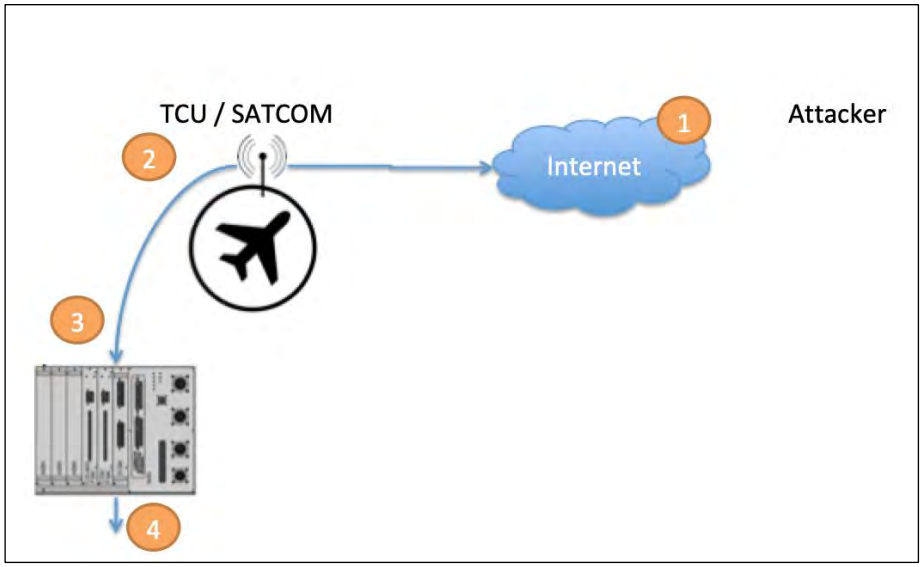


Figure 27. Communication Link to CDN Scenario

[39] https://ioactive.com/wp-content/uploads/2018/08/us-18-Santamarta-Last-Call-For-Satcom-Security-wp.pdf

[40] https://www.teledynecontrols.com/en-us/Product%20Brochures/Teledyne%20TCU%20Brochure.pdf

Exploitability Assessment

Boeing has stated that they are unable to confirm that the reported vulnerabilities can be exploited in a real production environment, essentially by adducing empiric, rather than technical, reasons.

According to our experience and capabilities, we would like to provide a comprehensive explanation on why we consider that these vulnerabilities are, in fact, exploitable.

Potential Mitigations

Lack of NX/XD[41] Support

As previously mentioned, the CIS/MS module is a COTS CPB4612[42] board. There are different revisions of this board that may fit within the date range, so it is not possible to know the specific Pentium M model that has been used, but we certainly know it is an Intel Pentium 32-bit processor. As a result, we can analyze the VxWorks' kernel MMU initialization routines to check whether the PAE bit is enabled, which would be the known case where an x86 32-bit CPU may support the well-known No-Execute (NX/XD) hardware mitigation.

[41] https://en.wikipedia.org/wiki/NX_bit

[42] https://manualsbrain.com/en/manuals/1189567/

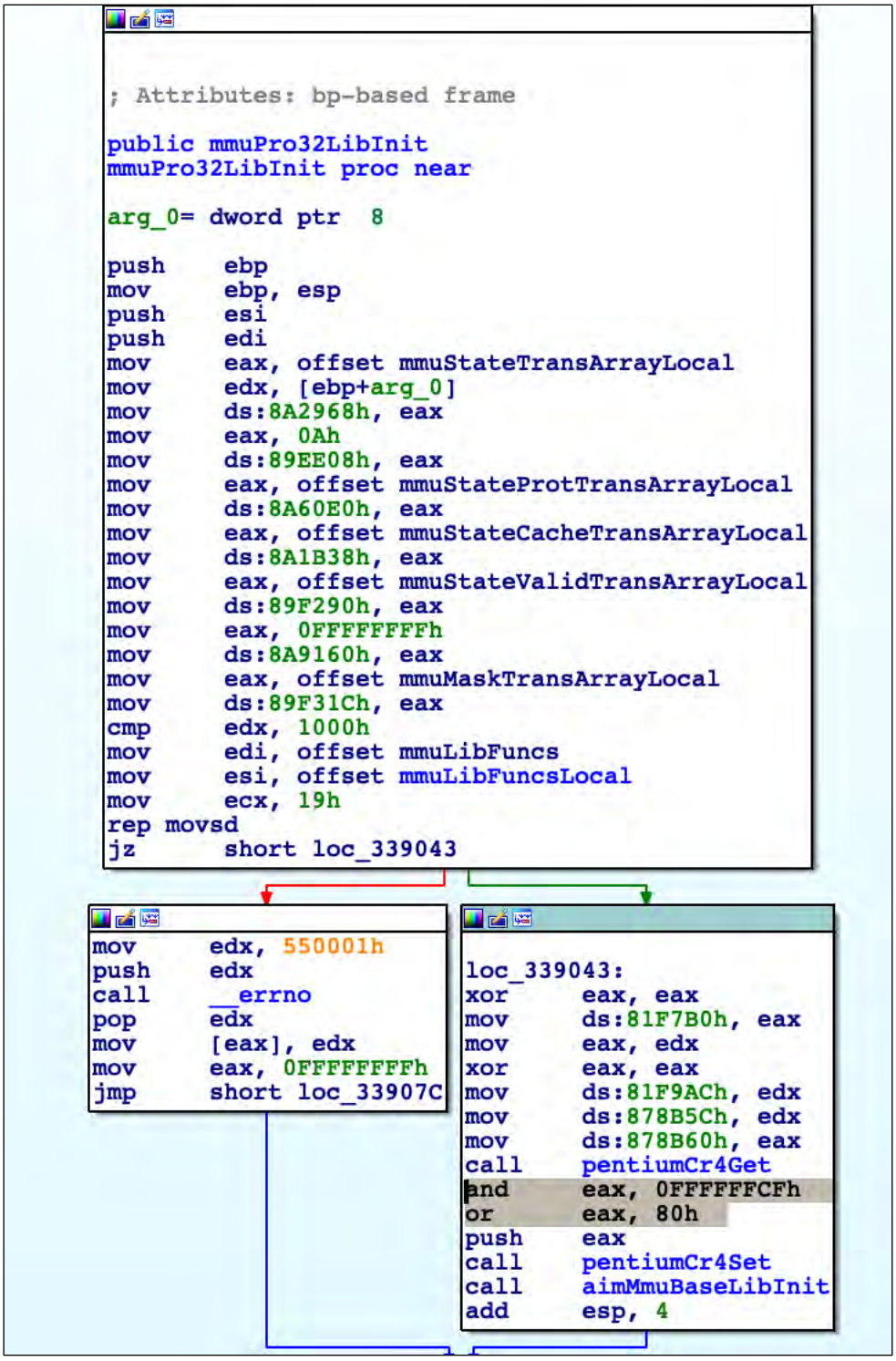


Figure 28. MMU Initialization Routine in CIS/MS VxWorks

In the highlighted instructions, we can see how the PAE/PSE bits are cleared in the Cr4[43] register. Therefore, we can discard that the NX/XD hardware mitigation is supported in the CIS/MS.

Lack of Compiler-level Mitigations

Boeing stated that in their tests, insecure functions, such as `strcpy`, `sprintf`, and `strcat`, could not lead to buffer overflows due to compiler-level mitigations. We shared our concerns with them regarding this claim, as our research revealed no compiler-level mitigations in the firmware version we analyzed. As previously explained, Boeing refused to answer our question about whether these mitigations could have been implemented in a later version, which would mean the system was exploitable (if we assume Boeing's claim that this kind of mitigations cannot be bypassed) until that specific date.

There is no other option, as a compiler-level mitigation necessarily needs to emit extra code to check for buffer overflows, such as stack overflow attacks. We analyzed the CIS/MS binaries looking for some of the common GCC stack protection mechanisms with no results.

We illustrate this statement with the following examples:

1. **RTP – Shared Library 'libc.so.1' - 'strcpy' function**

As Figure 29 shows, there are no compiler-level mitigations in the `strcpy` implementation RTPs are using.

[43] https://en.wikipedia.org/wiki/Control_register

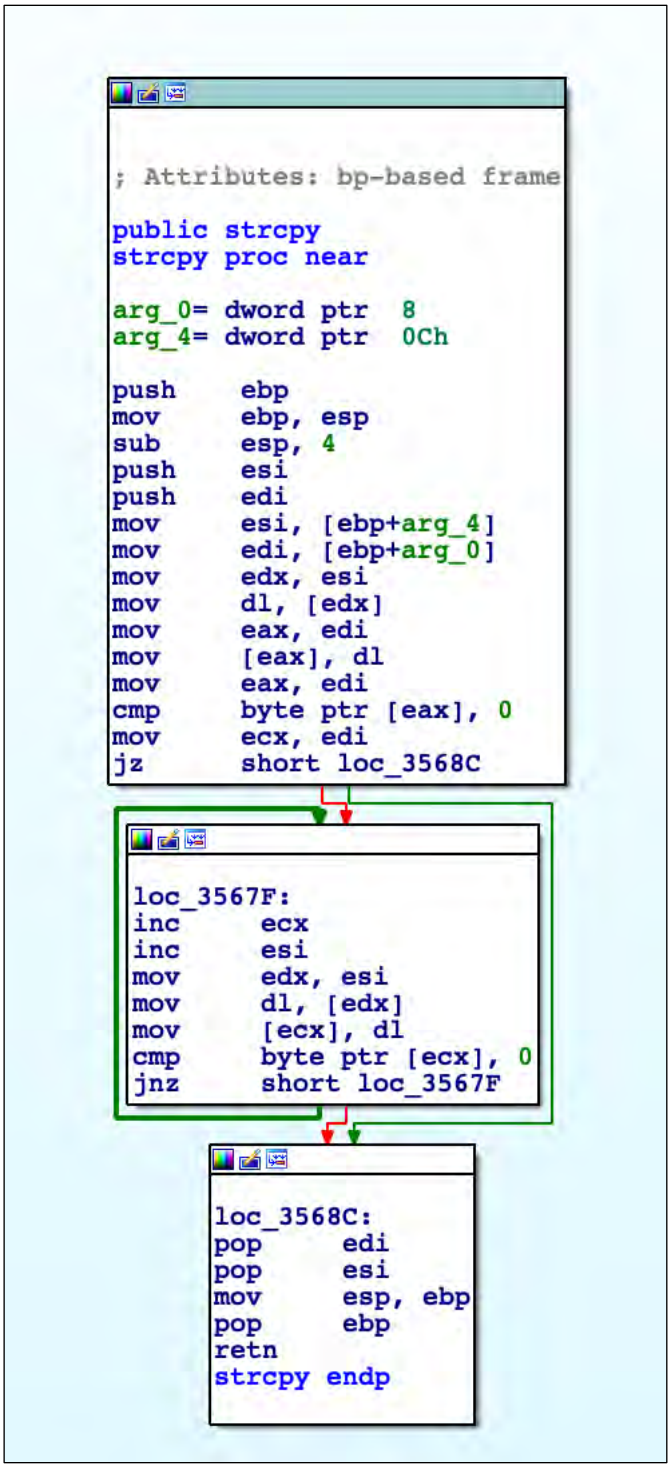


Figure 29. strcpy Implementation

2. RTP 'FTS_Manager.vxe'

We now examine one of the vulnerable functions in `FTS_Manager.vxe`, which has a significant stack frame.

```
.text:08076DC8
.text:08076DC8 ; int __cdecl server_task(int, char *src)
.text:08076DC8                 public server_task
.text:08076DC8 server_task     proc near               ; DATA XREF: sub_806852D+169↑o
.text:08076DC8
.text:08076DC8 var_E2C         = dword ptr -0E2Ch
.text:08076DC8 var_E28         = dword ptr -0E28h
.text:08076DC8 var_E24         = dword ptr -0E24h
.text:08076DC8 var_E20         = dword ptr -0E20h
.text:08076DC8 var_E1C         = dword ptr -0E1Ch
.text:08076DC8 var_E18         = dword ptr -0E18h
.text:08076DC8 var_E14         = dword ptr -0E14h
.text:08076DC8 var_E10         = dword ptr -0E10h
.text:08076DC8 var_E0C         = dword ptr -0E0Ch
.text:08076DC8 var_E08         = byte ptr -0E08h
.text:08076DC8 dest            = byte ptr -808h
.text:08076DC8 var_7C0         = dword ptr -7C0h
.text:08076DC8 var_7BC         = dword ptr -7BCh
.text:08076DC8 var_7B8         = byte ptr -7B8h
.text:08076DC8 var_799         = byte ptr -799h
.text:08076DC8 s1              = byte ptr -738h
.text:08076DC8 var_6B9         = byte ptr -6B9h
.text:08076DC8 var_6AC         = dword ptr -6ACh
.text:08076DC8 var_6A8         = byte ptr -6A8h
.text:08076DC8 var_2A8         = byte ptr -2A8h
.text:08076DC8 var_298         = dword ptr -298h
.text:08076DC8 optval          = dword ptr -294h
.text:08076DC8 var_290         = dword ptr -290h
.text:08076DC8 var_28C         = dword ptr -28Ch
.text:08076DC8 var_288         = dword ptr -288h
.text:08076DC8 var_284         = dword ptr -284h
.text:08076DC8 var_280         = dword ptr -280h
.text:08076DC8 var_27C         = dword ptr -27Ch
.text:08076DC8 var_278         = dword ptr -278h
.text:08076DC8 var_274         = dword ptr -274h
.text:08076DC8 var_270         = dword ptr -270h
.text:08076DC8 var_26C         = dword ptr -26Ch
.text:08076DC8 var_268         = dword ptr -268h
.text:08076DC8 n               = dword ptr -264h
.text:08076DC8 var_260         = dword ptr -260h
.text:08076DC8 fd              = dword ptr -25Ch
.text:08076DC8 var_258         = dword ptr -258h
.text:08076DC8 var_244         = dword ptr -244h
.text:08076DC8 var_240         = dword ptr -240h
.text:08076DC8 ptr             = dword ptr -23Ch
.text:08076DC8 buf             = dword ptr -238h
.text:08076DC8 s               = byte ptr -28h
.text:08076DC8 var_27          = byte ptr -27h
.text:08076DC8 var_26          = word ptr -26h
.text:08076DC8 var_24          = dword ptr -24h
.text:08076DC8 addr            = sockaddr ptr -18h
.text:08076DC8 var_8           = byte ptr -8
.text:08076DC8 var_4           = dword ptr -4
.text:08076DC8 arg_0           = dword ptr  8
.text:08076DC8 src             = dword ptr  0Ch
.text:08076DC8
.text:08076DC8                 push    ebp
.text:08076DC9                 mov     ebp, esp
.text:08076DCB                 push    edi
.text:08076DCC                 sub     esp, 0E34h
.text:08076DD2                 mov     [ebp+ptrl], 0
```

Figure 30. server_task Setting Up the Stack

As we can see in Figure 31, there is no sign of GCC stack protection code.


```
text:0807846C                   add     esp, 10h
text:0807846F                   mov     eax, 0
text:08078474                   mov     edi, [ebp+var_4]
text:08078477                   leave
text:08078478                   retn
text:08078478 server_task       endp
```

Figure 31. server_tasks Epilog

Conclusions

We did not describe anything novel in this section, as these concepts have been largely explained over the past decades; however, we recognize some of the audience for this paper may come from the aviation industry and be unfamiliar with some of the specific details.

Post Exploitation

From CDN to Safety Critical Systems

There are 21 RDCs connected to the CDN, all of them enabling CAN↔A664 gateways for A615a Data Loading operations. Basically, this is something the NIM is designed for, as these LSAPs can be updated either through a maintenance laptop or from ground infrastructure through a communication link.

Taking into account the configured Virtual Links in the NIM's End System required for normal system operation, it is possible to identify one of the most significant attack vectors in the ability to initiate the update of firmware belonging to safety critical units, such as Bus Power Control Unit (BPCU), Generator Control Unit (GCU)[44], Electronic Engine Control (EEC), Wing Ice Protection System (WIPS), etc.

IOActive was not able to determine during this research project if there are firmware integrity controls in place that would limit the impact of an attempt to update firmware in these safety-critical components.

The following is a list of the controllers, actuators, and systems connected to the RDCs.

- Main Engine Data Concentrators
- Brake System Control Cards
- Valve Control Circuit Cards
- Proximity Sensors Data Concentrators
- Electric Motor Pump Controller
- Electric Control Break Actuator
- Fuel Quantity
- Emergency Power Assist System
- Wireless Emergency Light System
- Ram Air Fan Controller
- Maintenance Display Unit
- Cabin Air Compressor
- Shutoff Fuel Module
- Refuel Control Panel

[44] https://www.theguardian.com/business/2015/may/01/us-aviation-authority-boeing-787-dreamliner-bug-could-cause-loss-of-control

- Wing Ice Protection System
- Bus Power Control Unit
- Electronic Control Unit
- Secondary Power Distribution Unit
- Engine Monitor Unit
- Electronic Engine Control
- Remote Power Distribution Unit
- Graphics Generator Display
- Flight Recorder
- Audio Units

The following data demonstrates the CDN rules that allow this type of communication.

File 'AimCfg.xml'

```
   <!-- PDT = [CAN Data Load]   ODLF_200kb_CAN_S1_Tx1 |Usg to UNIT-BRAKE
SYS CTRL CARD-INBD-L |Occ.R1_C3_ODLF_WKP_HFQI_16 |Usg -->
            <!--*** APP-CIS MS ACTIVE-1 |Usg to UNIT-BRAKE SYS CTRL CARD-
INBD-L |Occ ***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.42.161.31</CDN_Dest_IP>
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>66112</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62911</IDN_Source_UDP>
            </Uni_Tx>



…
<!-- PDT = [CAN Data Load]   ODLF_200kb_CAN_S4_Tx1 |Usg to CONTROLLER-ELEC
BRK ACTR CH-AFT-RO |Occ.R1_C6_ODLF_WKP_HFQI_23 |Usg -->
            <!--*** APP-CIS MS ACTIVE-1 |Usg to CONTROLLER-ELEC BRK ACTR
CH-AFT-RO |Occ ***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.42.170.9</CDN_Dest_IP>
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>66115</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62908</IDN_Source_UDP>
            </Uni_Tx>
…
            <!-- PDT = [CAN Data Load]   ODLF_200kb_CAN_S1_Tx1 |Usg to
DATA CONCENTRATOR-FUEL_QTY-CENTER |Usg.R5_C10_ODLF_WKP_HFQI_14 |Usg -->
```



```
            <!--*** APP-CIS MS ACTIVE-1 |Usg to DATA CONCENTRATOR-
FUEL_QTY-CENTER |Usg ***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.42.165.78</CDN_Dest_IP>
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>66112</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62911</IDN_Source_UDP>
            </Uni_Tx>
…
            <!-- PDT = [CAN Data Load]    ODLF_200kb_CAN_S2_Tx1 |Usg to
MODULE-SHUTOFF-FUEL |Usg.R12_C6_ODLF_WKP_HFQI_4 |Usg -->
            <!--*** APP-CIS MS ACTIVE-1 |Usg to MODULE-SHUTOFF-FUEL |Usg
***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.42.172.11</CDN_Dest_IP>
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>66113</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62910</IDN_Source_UDP>
            </Uni_Tx>
…
<!-- PDT = [CAN Data Load]    ODLF_200kb_CAN_S4_Tx1 |Usg to CONTROLLER-
WIPS-1 |Occ.R17_C4_ODLF_WKP_HFQI_12 |Usg -->
            <!--*** APP-CIS MS ACTIVE-1 |Usg to CONTROLLER-WIPS-1 |Occ
***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.42.177.2</CDN_Dest_IP>h
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>66115</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62908</IDN_Source_UDP>
            </Uni_Tx>
…
</Uni_Tx>
            <!-- PDT = [ARINC 615A]     ODLF_10Mb_S1_Tx1 |Usg to APP
PARTITION-CONTROL UNIT-R2-GEN |Usg.R2_GCU_Data_Load_Rx0_HFQI_Port |Usg --
>
            <!--*** APP-CIS MS ACTIVE-1 |Usg to APP PARTITION-CONTROL
UNIT-R2-GEN |Usg ***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.24.11.81</CDN_Dest_IP>
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>10022</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62830</IDN_Source_UDP>
            </Uni_Tx>
…
```



```
        <!-- PDT = [ARINC 615A]    ODLF_10Mb_S2_Tx1 |Usg to CONTROLLER-
EEC_CHANNEL==B-1 |Occ.EECB_Data_Load_Rx0_L |Occ -->
            <!--*** APP-CIS MS ACTIVE-1 |Usg to CONTROLLER-EEC_CHANNEL==B-
1 |Occ ***-->
            <Uni_Tx>
                <CDN_Dest_IP>10.73.2.0</CDN_Dest_IP>
                <CDN_Dest_UDP>59</CDN_Dest_UDP>
                <Es_Tx_Port_ID>10009</Es_Tx_Port_ID>
                <IDN_Source_IP>172.24.10.12</IDN_Source_IP>
                <IDN_Source_UDP>62904</IDN_Source_UDP>
            </Uni_Tx>
```

Again, this post-exploitation scenario may be mitigated if the firmware files for these safety critical units are properly encrypted/signed and the verification steps are implemented accordingly. However, IOActive does not have any further information regarding the use of best practices for firmware integrity such as NIST 800-193.[45]

Maintenance Operations

There are three modes for the maintenance operators depending on how they are connected.

1. **Wired:** When the engineer is connected through one of the three wired ports in the flight deck or equipment centers, it is possible to exercise any maintenance operation available on the system.
2. **Wireless:** When the engineer is connected wirelessly through the CWLU/TWLU, only a limited set of maintenance operations are enabled.
3. **Full Wireless:** This mode enables the engineer who is wirelessly connected through the CWLU/TWLU to 'upgrade' from a Wireless connection to a Full Wireless mode, which is equivalent to the Wired mode. In order to enable all of the operations, the engineer needs to enter a code that is generated in the CIS/MS through the cabin interphones. If the code entered matches the locally generated challenge code, the engineer is upgraded to Full Wireless mode, and the CIS/MS unblocks CDN access for the engineer's Maintenance Terminal IP.

[45] https://doi.org/10.6028/NIST.SP.800-193

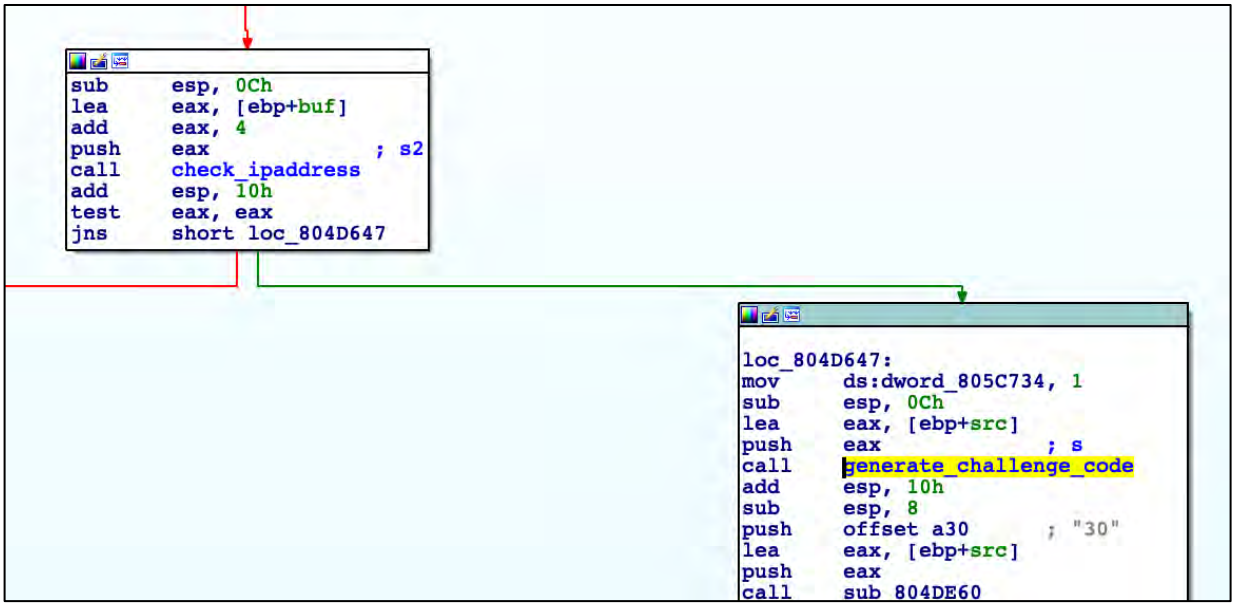


Figure 32. Challenge Code for Full Wireless Mode

This functionality is implemented in the `OMLS.vxe` RTP, which is also vulnerable. As a result, a wireless attacker can gain Full Wireless maintenance mode without being physically onboard. Also, this scenario opens up the cabin interphones as an attack vector.

```
text:0804DA78 loc_804DA78:                            ; CODE XREF: authentication_manager+B71↑j
text:0804DA78                 sub     esp, 8
text:0804DA7B                 lea     eax, [ebp+s1]
text:0804DA81                 push    eax             ; s
text:0804DA82                 push    dword ptr [ebp+addr.sa_data+2] ; char
text:0804DA88                 call    inet_ntoa_b
text:0804DA8D                 add     esp, 10h
text:0804DA90                 sub     esp, 8
text:0804DA93                 push    offset byte_805D14C ; s2
text:0804DA98                 lea     eax, [ebp+s1]
text:0804DA9E                 push    eax             ; s1
text:0804DA9F                 call    _strcmp
text:0804DAA4                 add     esp, 10h
text:0804DAA7                 test    eax, eax
text:0804DAA9                 jz      short loc_804DAE4
text:0804DAAB                 sub     esp, 4
text:0804DAAE                 lea     eax, [ebp+s1]
text:0804DAB4                 push    eax
text:0804DAB5                 push    offset aCabinInterphon ; "Cabin Interphone Message received from "...
text:0804DABA                 lea     eax, [ebp+var_6A8]
text:0804DAC0                 push    eax             ; s
text:0804DAC1                 call    _sprintf
text:0804DAC6                 add     esp, 10h
text:0804DAC9                 sub     esp, 4
text:0804DACC                 lea     eax, [ebp+var_6A8]
text:0804DAD2                 push    eax
text:0804DAD3                 push    0
```

Figure 33. Authentication Message Received from the Cabin Interphone

This post-exploitation scenario could plausibly be used to hide or manipulate maintenance information in order to deceive technicians, as well as to perform certain unexpected maintenance operations or tests.

Conclusions

In this paper, we have documented our detailed attack paths and component vulnerabilities to describe the first plausible, detailed public attack paths to effectively reach the avionics network on a commercial airplane from either non-critical domains, such as Passenger Information and Entertainment Services, or even external networks.

Upon conclusion of the analysis, Boeing and Honeywell confirmed that these vulnerabilities are present in the 787's Core Network codebase; however, the official response IOActive received from Boeing was that they do not consider our reported findings exploitable vulnerabilities, as they could not reproduce these flaws.

In addition, Boeing stated that they have mitigations in place that prevent the vulnerabilities from being exploited; however, they were unwilling to share those details with IOActive. IOActive found this to be deeply disappointing, since this prevents us from independently validating the exploitability of the identified vulnerabilities. Without a 787, a 787 lab environment, or an explanation of the controls, IOActive is unable to confirm Boeing's claims of compensating controls or mitigations for these software vulnerabilities.

As a result, we hope that a determined, highly capable third party can safely confirm that these vulnerabilities are not exploitable due to mitigation controls not visible to us during this analysis. We are confident owners and operators of these aircraft would welcome such independent validation and verification.

We believe as strongly in safety as we do in security. We provide these detailed findings herein so that all stakeholders, members of the security industry, and affected entities can form their own judgment as to the exploitability and impact of these confirmed software vulnerabilities.

IOActive believes follow-on work should occur to assess the layered security controls designed to prevent lateral movement within the vehicle network on aircraft other than the 787 and from manufacturers other than Boeing.

About Ruben Santamarta

Ruben Santamarta is experienced in network penetration and web application testing, reverse engineering, industrial control systems, transportation, RF, embedded systems, AMI, vulnerability research, exploit development, and malware analysis. As a principal consultant at IOActive, Mr. Santamarta performs penetration testing, identifies system vulnerabilities, and researches cutting-edge technologies. Mr. Santamarta has performed security services and penetration tests for numerous global organizations and a wide range of financial, technical, and educational institutions. He has presented at international conferences including Ekoparty and Black Hat USA.

About IOActive

IOActive is a comprehensive, high-end information security services firm with a long and established pedigree in delivering elite security services to its customers. Our world-renowned consulting and research teams deliver a portfolio of specialist security services ranging from penetration testing and application code assessment through to semiconductor reverse engineering. Global 500 companies across every industry continue to trust IOActive with their most critical and sensitive security issues. Founded in 1998, IOActive is headquartered in Seattle, USA, with global operations through the Americas, EMEA and Asia Pac regions. Visit www.ioactive.com for more information. Read the IOActive Labs Research Blog: http://blog.ioactive.com. Follow IOActive on Twitter: http://twitter.com/ioactive.

