IOActive®
Research-fueled Security Services

\ WHITE PAPER \

Reverse Engineering of DAL-A Certified Avionics: Collins' Pro Line Fusion — AFD-3700

Ruben Santamarta
Security Researcher, IOActive

April 19, 2022

Contents

Notices

No Warranties or Representations

The information presented herein is provided "AS IS" and IOActive disclaims all warranties whatsoever, whether express or implied. Further, IOActive does not endorse, guarantee, or approve, and assumes no responsibility for nor makes any representations regarding the content, accuracy, reliability, timeliness, or completeness of the information presented. Users of the information contained herein assume all liability from such use.

Publicly Available Material

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

Fair Use

This primary purpose of this presentation is to educate and inform. It may contain 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. This material is distributed without profit for the purposes of criticism, comment, news reporting, teaching, scholarship, education, and research, and constitutes fair use as provided for in section 107 of the Copyright Act of 1976.

Trademarks

IOActive, the IOActive logo and the hackBOT logo are trademarks and/or registered trademarks of IOActive, Inc. in the United States and other countries. All other trademarks, product names, logos, and brands are the property of their respective owners and are used for identification purposes only.

No Endorsement or Commercial Relationship

The use or mention of a company, product or brand herein does not imply any endorsement by IOActive of that company, product, or brand, nor does it imply any endorsement by such company, product manufacturer, or brand owner of IOActive. Further, the use or mention of a company, product, or brand herein does not imply that any commercial relationship has existed, currently exists, or will exist between IOActive and such company, product manufacturer, or brand owner.

Copyright

Abstract

Modern avionic systems are designed according to the Integrated Modular Avionics concept. Under this paradigm, safety-certified avionic applications and non-critical airborne software share the same computing platform but are running at different partitions. In this context the underlying safety-critical certified RTOS provides the logical isolation, which should prevent unintended interactions between software with different criticalities.

This paper provides a comprehensive analysis of the architecture and vulnerabilities found on the Adaptive Flight Display component of the Collins Aerospace's Pro Line Fusion solution. This integrated avionics system, deployed both in military and commercial aircraft, is certified as DO-178B/C Design Assurance Level A.

Introduction

Research Context

A series of precautions must be considered within the context of a vulnerability disclosure that affects the aviation industry, where even a minimal inaccuracy may be used to discredit and invalidate the research as a whole. In IOActive's experience, affected entities in the aviation sector tend to maintain an opaque attitude, compared with other industries. Therefore, the burden of the proof is almost entirely on the researcher's side, which poses a significant challenge in such a complex field.

This specific scenario requires not only a comprehensive description, a plausible explanation, and a complete technical analysis, but also enough evidence to sustain the conclusions of the research. Additionally, it is worth mentioning the inability to physically access neither a fully working aircraft nor a simulator to legally test the attacks in a live environment.

Neither Collins Aerospace nor its customers or partners provided any technical support to IOActive: the research has been performed by following a static black-box[1] approach, solely based on the reverse engineering of the firmware, without having physical access to the hardware.

The main objectives of this research are the following:

- Demonstrate that the target in scope is actually certified for safety-critical operations
- Demonstrate that the target, a safety-critical certified avionics component, can be compromised, either remotely or via inter-partition attacks, during any phase of flight
- Demonstrate the potential safety implications derived from a compromised target

The structure of this paper, as well as its narrative, have been conceived according to these objectives. All content in this paper has been included for a reason, even if it appears obvious

[1] No access to source code, documentation, or resources beyond what it is publicly available.

or redundant. The reader should carefully note all the references that can be found throughout the document, as they point out external sources that can be used to contrast the claims presented herein. Special effort has been put into introducing those concepts for which there are no references available, without covering in detail others for which a large amount of literature is already available, such as IMA[2].

Disclosure

IOActive and Collins Aerospace have been coordinating the issues herein described since March 2021.

Several pre-publication versions of this paper were shared with Collins Aerospace. In their recent letter dated April 7, 2022 they acknowledge the vulnerabilities ("defects" according to their nomenclature) described in this research and will proceed "to make updates which will address issues you've described as part of our next major release with development starting this year. Once changes have been made to the software, verification and certification will be required across multiple configurations and platforms".

They also asked for deletion of two statements regarding one of the post-exploitation scenarios as well as the list of the impacted aircraft. Additionally, their assessment of the potential safety implications is not aligned with ours, as they state that 'defects do not adversely impact operational safety'.

IOActive has highlighted these three disputed statements in the paper, to provide the reader a clear view of both Collins Aerospace and IOActive respective positions.

Disputed statement 1

Disputed statement 2

Disputed statement 3

Pro Line Fusion® and the AFD-3700

Pro Line Fusion from Collins Aerospace is an integrated avionics suite (see Figure 2).[3] [4] Its architecture is comprised of multiple systems, and it provides safety-critical functionality.

Pro Line Fusion®

Unprecedented safety, efficiency and predictability on every mission

Figure 1. Pro Line Fusion Banner - Collins Aerospace Website

[2] https://en.wikipedia.org/wiki/Integrated_modular_avionics

[3] https://www.collinsaerospace.com/what-we-do/Business-Aviation/Flight-Deck/Pro-Line-Fusion

[4] Challenger 604 – Pro Line Fusion Tour https://www.youtube.com/watch?v=BbV9iqdfVaM

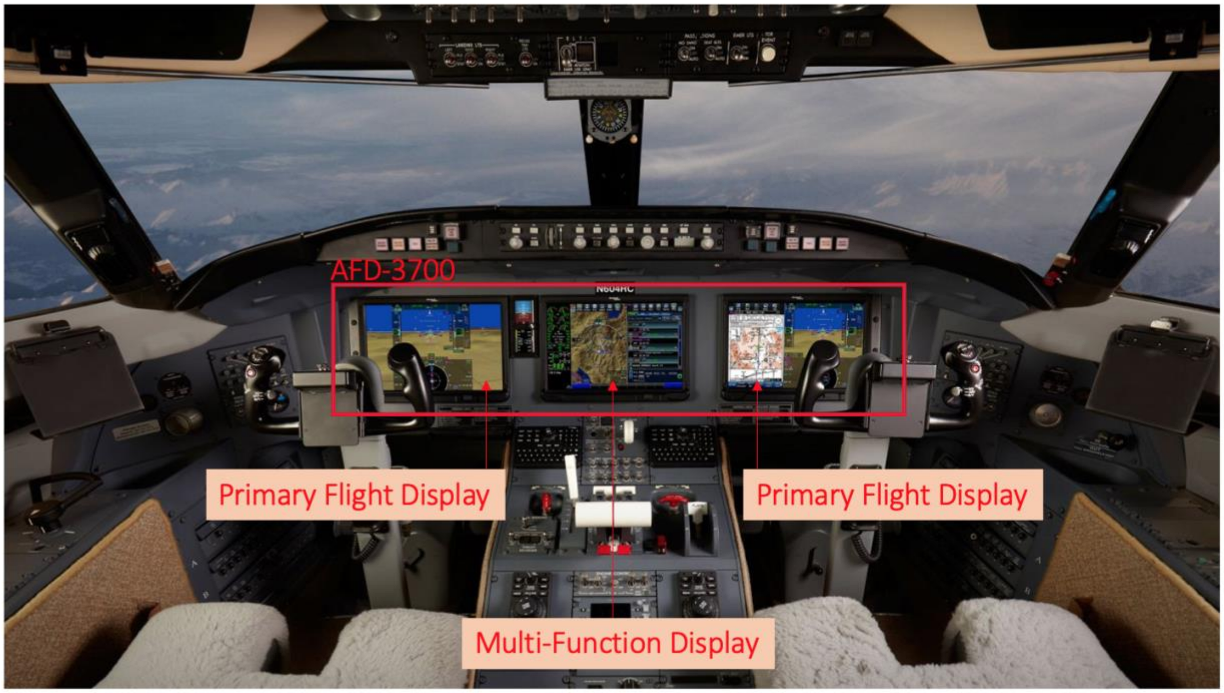


Figure 2. Pro Line Fusion Avionics Suite - Challenger CL604, Bombardier

In the context of the Pro Line Fusion, the Electronic Flight Instrumentation System (EFIS) implements at least three[5] model AFD-3700 Display Units (DUs, see Figure 3) that provide display and control capabilities for features such as:

- Synthetic Vision System (SVS)
- Advanced Terrain Functions (ATF)
- Traffic Alert Collision Avoidance System (TCAS)
- Engine Indicating and Crew Alerting System (EICAS)
- Attitude Heading and Reference System (AHRS)
- Flight Management System (FMS)
- Weather Radar System (WXR)
- File Server Application (FSA)
- Flight Display System Application (FDSA)
- Radio Tuning System Application (RTSA)

It is important to clarify that the extent of these aforementioned applications is not limited to the scope of the AFD-3700, but also usually integrate with multiple systems across different

[5] Two in light helicopters/aircraft.

components in the aircraft. For instance, the EICAS functional application in the AFD-3700 DU may consume data from different sensors and systems.

Figure 3. Display Units (AFD-3700)[6]

At the factory, Collins Aerospace loads the DUs with the runtime system AFDR-3700 (Adaptive Flight Display Runtime), which is certified as DO-178B/C Design Assurance Level (DAL) A[7]. The DAL-A is associated with functions whose anomalous behavior could cause or contribute to a catastrophic failure condition for the aircraft. The AFDR-3700 consists of the real-time operating system (RTOS), drivers, configuration tables, and applications that enable the DU to properly operate as well as to perform field loading operations both via USB and wirelessly through an external data loader, such as the IMS.[8] Later on, aircraft manufacturers can load the DU with the proper functional applications (EICAS, FMS, etc.) and configuration tables required for their respective aircraft (see Figure 4 and Figure 5).

[6] https://support.cessna.com/custconf/pageview?as_id=46540
[7] https://en.wikipedia.org/wiki/DO-178B
[8] https://www.youtube.com/watch?v=s20Xjq4HnEQ

Figure 4. AFD-3700 Nameplates

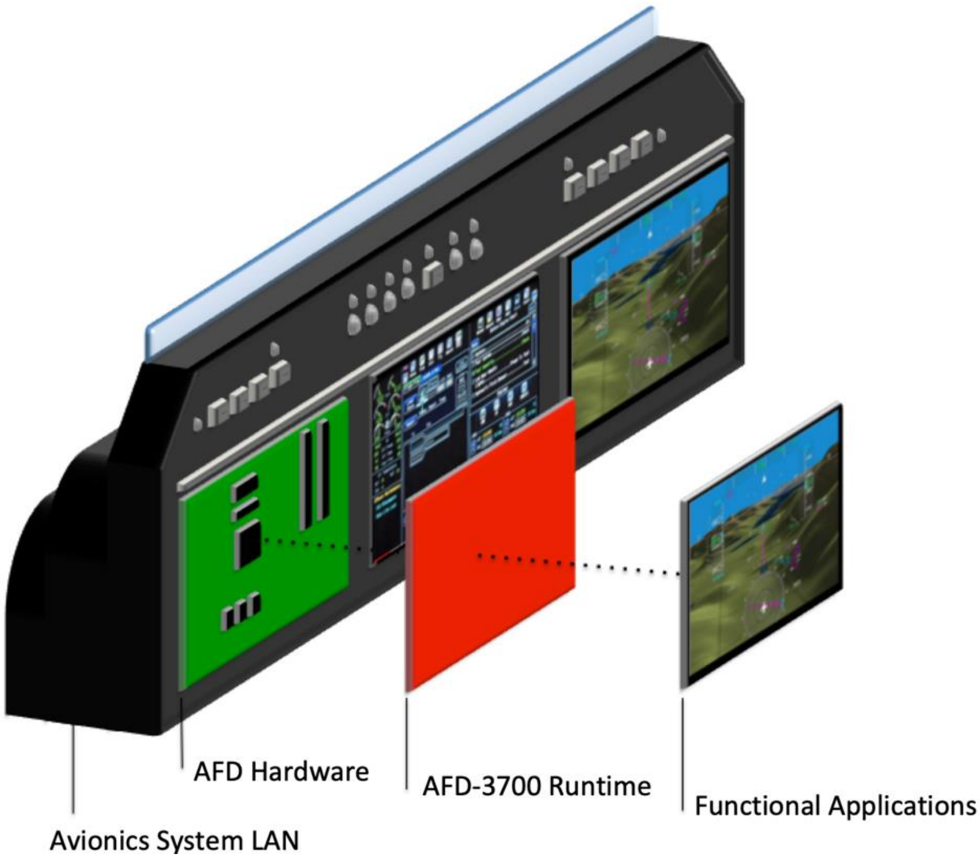


Figure 5. AFD-3700 Components

As depicted in Figure 5. AFD-3700 Components, the AFDR-3700 manages the AFD hardware and software resources, and provides common services that let the functional applications run. This essentially means that a compromised AFD-3700 Runtime may directly influence the loaded functional applications.

The files that enabled this research were retrieved from the publicly accessible Rockwell Collins support portal (see Table 1).[9] This server exposed unauthenticated downloads, including the Black Label (production release) version of the ARINC665-3 Loadable Software Parts of AFDR-3700 intended for distribution to King Air[10] aircraft.

Table 1. Exposed files

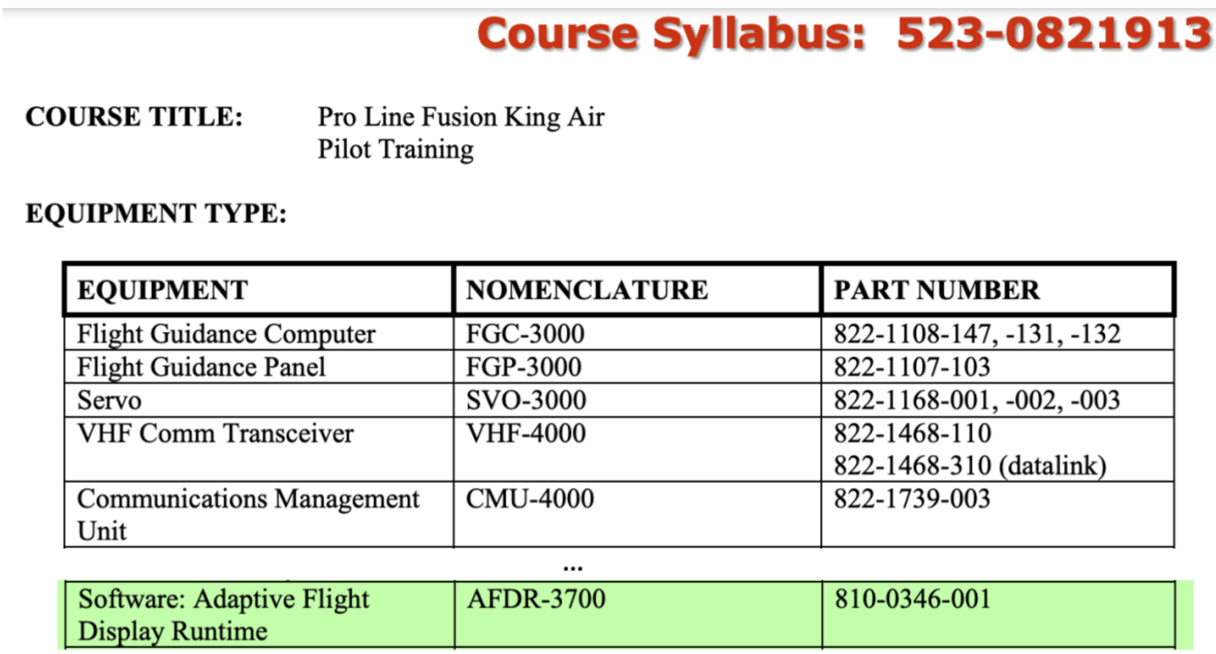
File	Description
`COL_Application01.001`	LynxOS-178 Kernel Downloadable Image (KDI) (AFDR-3700)
`COL_Application01.002`	Rockwell Collins AFDR-3700 User Filesystem
`COL_Application01.luh`	A665-3 Load Upload Header
`COL_Table01.001`	Product version and certification
`COL_Table01.002`	Product version and certification
`COL_Table01.004`	VCT for the following functional applications: EICAS-6000, RTSA-6000, and ECDA-6000
`COL_Table01.005`	VCT for the following functional application: ATF-3500
`COL_Table01.003-033`	AFD Functional Configuration Tables
`COL_Table01.luh`	A665-3 Load Upload Header
`FILES.lum` and `LOADS.lum`	A665-3 LUM files
`COL_Table01.012` (cached)	IOActive found a version of this file (accessible via Google searches) that was different from the file downloaded from the server. The cached version is the `SL03.vct` file for the FDSA-6500 functional application. https://portal.rockwellcollins.com › COL_Table01.012 Virtual Machine Configuration Table // // File Name: S3-SL01.VCT ... ... <VM0> // VCT2177 GroupIds=; // VCT1187 **LogicalName**=AFDR-3700; // VCT1188 ... <VM1> // VCT2178 GroupIds=; // VCT2147 **LogicalName**=FDSA-6500; ...

[9] https://web.archive.org/web/20210119190712/https://portal.rockwellcollins.com/web/support-self-service/kidde-claim/-/document_library/T8Mdho6qCThZ/view/1910640?_com_liferay_document_library_web_portlet_DLPortlet_INSTANCE_T8Mdho6qCThZ_redirect=https%3A%2F%2Fportal.rockwellcollins.com%3A443%2Fweb%2Fsupport-self-service%2Fkidde-claim%3Fp_p_id%3Dcom_liferay_document_library_web_portlet_DLPortlet_INSTANCE_T8Mdho6qCThZ%26p_p_lifecycle%3D0%26p_p_state%3Dnormal%26p_p_mode%3Dview

[10] https://en.wikipedia.org/wiki/Beechcraft_Super_King_Air

File	Description
`COL_Table01.003` (cached)	IOActive found a version of this file (accessible via Google searches) that was different from the file downloaded from the server. The cached version is the `SL02.vct` file for the FSA-6000 functional application. https://portal.rockwellcollins.com › COL_Table01.003 Virtual Machine Configuration Table // // File Name: S1-SL02.VCT ... ... SysRamMemLim=52428800; // VCT1203 PersStorOnLocalLim=512; // VCT1206 </VM0> <VM1> // VCT53 GroupIds=; // VCT2212 LogicalName=**FSA-6000**; ...

An initial analysis of the `COL_Application01.001` and `COL_Application01.002` files revealed an AFDR-3700 version dating back to 2014 with part number 810-0346-001 (see Figure 7), which matches the official part number referenced in official documents from Collins (see Table 1. Exposed files).



Course Syllabus: 523-0821913

COURSE TITLE: Pro Line Fusion King Air Pilot Training

EQUIPMENT TYPE:

EQUIPMENT	NOMENCLATURE	PART NUMBER
Flight Guidance Computer	FGC-3000	822-1108-147, -131, -132
Flight Guidance Panel	FGP-3000	822-1107-103
Servo	SVO-3000	822-1168-001, -002, -003
VHF Comm Transceiver	VHF-4000	822-1468-110 822-1468-310 (datalink)
Communications Management Unit	CMU-4000	822-1739-003
	...	
Software: Adaptive Flight Display Runtime	AFDR-3700	810-0346-001

Figure 6. Pro Line Fusion Course for King Air[11]

[11] https://portal.rockwellcollins.com/documents/1904088/2147097/SYB5230821913.pdf/ed9d4f14-65f2-764d-78ab-bd8995b30f61

```
3E3200 54797065 20414644 522D3337 30302020 20202020 20202020 20202020 20202020  Type AFDR-3700
3E3220 20202020 20202020 20202020 20202020 0D0A426C 61636B20 4C616265 6C202020                             Black Label
3E3240 20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020
3E3260 20200D0A 454D4F44 20313220 20202020 20202020 20202020 20202020 20202020        EMOD 12
3E3280 20202020 4D465220 30454644 30202020 20202020 0D0A504E 52203831 302D3033        MFR 0EFD0             PNR 810-03
3E32A0 34362D30 30312020 20202020 20202020 20202020 2020444D 46203230 31342D30  46-001                       DMF 2014-0
3E32C0 392D3136 20200D0A 526F636B 77656C6C 20436F6C 6C696E73 2C20496E 632E2043  9-16      Rockwell Collins, Inc. C
3E32E0 65646172 20526170 6964732C 20494120 35323439 38205553 0D0A4352 432D3332  edar Rapids, IA 52498 US   CRC-32
3E3300 20464644 43434537 38000000 00000000 00000000 00000000 00000000 00000000   FFDCCE78
3E3320 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3E3340 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3E3360 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3E3380 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3E33A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3E33C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3E33E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
3E3400 54797065 20414644 522D3337 30302020 20202020 20202020 20202020 20202020  Type AFDR-3700
3E3420 20202020 20202020 20202020 20202020 0D0A4641 41205453 4F204331 31336120                             FAA TSO C113a
3E3440 20202020 20202020 20202020 20202020 20202020 20202020 20202020 20202020
3E3460 20200D0A 444F2D31 37384220 4C657665 6C20412F 44202020 20202020 20202020        DO-178B Level A/D
3E3480 20202020 4D465220 30454644 30202020 20202020 0D0A504E 52203831 302D3033        MFR 0EFD0             PNR 810-03
3E34A0 34362D30 30312020 20202020 20202020 20202020 2020444D 46203230 31342D30  46-001                       DMF 2014-0
3E34C0 392D3136 20200D0A 526F636B 77656C6C 20436F6C 6C696E73 2C20496E 632E2043  9-16      Rockwell Collins, Inc. C
3E34E0 65646172 20526170 6964732C 20494120 35323439 38205553 0D0A4352 432D3332  edar Rapids, IA 52498 US   CRC-32
3E3500 20363644 38353743 46000000 00000000 00000000 00000000 00000000 00000000   66D857CF
```

Figure 7. Detail of `nameplate.txt` and `TSO_nameplate.txt` (C113a) Files[12] Found in `Col_application01.002` (AFDR-3700 USRFS)

LynxOS-178[13] is a POSIX/ARINC-653 conformant real-time operating system (RTOS) that has been granted DO-178B/C DAL-A certification by FAA/EASA regulators for safety-critical applications. The origin of the LynxOS-178 is VMOS, an avionics RTOS developed by Rockwell Collins.

The following statement[14] is publicly available on the Lynx (manufacturer of LynxOS, which at the time was named LynuxWorks) website, showing that LynxOS-178 is used in several other components besides the AFD runtime:

> Earlier this year, LynuxWorks received Advisory Circular AC 20-148 approval from the FAA for reusable software components (RSC) authorized for the LynxOS-178 operating system used in the Rockwell Collins Adaptive Flight Display Runtime, Common Computing Module Runtime, Data Concentration Module Runtime and Synthetic Vision Module Runtime for Pro Line Fusion.

Figure 8. LynxOS-178 Running in Additional Components

Table 2 provides the complete list of the archives that were extracted from the 'Col_Application.002' file system.

Table 2. Rockwell Collins USRFS files

12 https://rgl.faa.gov/Regulatory_and_Guidance_Library/rgTSO.nsf/0/dd968e96d184041e862579f10070b452/$FILE/TSO-113a.pdf

13 https://www.lynx.com/products/lynxos-178-do-178c-certified-posix-rtos

14 https://www.lynx.com/press-releases/lynxos-178-rtos-deployed-by-rockwell-collins-in-pro-line-fusion-series-of-flight-deck-systems

File	Type	Description
SL01.vct	LynxOS-178 virtual machine (VM) configuration table	VCT for Simple Display Application (SDA)
SL03.vct	LynxOS-178 VM configuration table	VCT for the following functional applications: EICAS-6000, RTSA-6000, and ECDA-6000
SL04.vct	LynxOS-178 VM configuration table	VCT for the following functional application: ATF-3500
nameplate.txt	Text file	Product and certification information
tso_nameplate.txt	Text file	Product and certification information
pcieinfo_default.info	LynxOS-178 driver info file	Default info file for PCIE driver
pcieinfo_policing_on_100MbsFull.info	LynxOS-178 driver info file	Unused info file for PCIE driver
pcieinfo_policing_on_autoneg.info	LynxOS-178 driver info file	Unused info file for PCIE driver
afdx_asl_info_0	LynxOS-178 driver info file	Default info file for AFDX driver
afdx_asl_info_default_0	LynxOS-178 driver info file	Default info file for AFDX driver
network.cfg	Proprietary Collins Aerospace file	Network (Avionics System LAN) Configuration file for AFDX and PCIE drivers
norflash.info	LynxOS-178 driver info file	Default info file for NORFLASH driver
iod.info	LynxOS-178 driver info file	Default info file for IOD driver
touch.info	LynxOS-178 driver info file	Default info file for TOUCH driver
rs422.info	LynxOS-178 driver info file	Default info file for RS422 driver
apm_info.info	LynxOS-178 driver info file	Default info file for APM driver
rtc.info	LynxOS-178 driver info file	Default info file for RTC driver
fat32fs.info	LynxOS-178 driver info file	Default info file for FAT32FS driver
usb_20rs.info	LynxOS-178 driver info file	Default info file for USB_20RS driver

File	Type	Description
ge4A.info	LynxOS-178 driver info file	Default info file for GE4 driver
ge4B.info	LynxOS-178 driver info file	Unused info file for GE4 driver
gecko.info	LynxOS-178 driver info file	Default info file for GECKO driver
merge.info	LynxOS-178 driver info file	Default info file for MERGE driver
ati_info_0	LynxOS-178 driver info file	Default info file for ATI_DRVR driver
pdkminfo_afd3700.info	LynxOS-178 driver info file	Default info file for PDKM driver
vm0.pct	Proprietary Collins Aerospace file	VM0 process configuration table
nand_system.info	LynxOS-178 driver info file	Default info file for NAND_FS_DRVR driver
ONFI_nand_bank{0-7}.info	LynxOS-178 driver info file	Info files related to NAND_FS_DRVR
drmlite_c1.info	LynxOS-178 driver info file	Info file related to DRMLITE driver
drmlite_c4.info	LynxOS-178 driver info file	Info file related to DRMLITE driver
drmlite_nodeferred.info	LynxOS-178 driver info file	Info file related to DRMLITE driver
drmlite_c5.info	LynxOS-178 driver info file	Info file related to DRMLITE driver
drmlite_c3.info	LynxOS-178 driver info file	Info file related to DRMLITE driver
drmlite_c2.info	LynxOS-178 driver info file	Info file related to DRMLITE driver
app_launcher	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	Collins Aerospace proprietary binary that executes the application configured in the VCT's PCT file
mkffs	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	Creates a flash filesystem for the VM
ffsck	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	Validates the VM's filesystem
arinc615a	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	ARINC615A data loading functionality
hm_main	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	Mandatory Health Monitoring/main application running in the privileged VM0

File	Type	Description
afdx_asl_drvr.obj	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	AFDX Avionics System LAN driver
pcie.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Low-level PCIE communication driver for End-System
norflash.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	NORFLASH driver
iod.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Flash partitions related driver
touch.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Touchscreen UART driver
rs422.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	RS422 driver
apm_drvr.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Aircraft Personality Module driver
rtc.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Real-Time Clock driver
fat32fs.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Fat32 Filesystem driver
usb_20rs.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	USB 2.0 driver
ge4.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Graphics Engine 4 driver
gecko.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Graphics Engine related driver
merge.dldd	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	Resource Manager driver
ati_drvr.obj	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	ATI RADEON E2400 driver

File	Type	Description
`nand_fs_drvr.dldd`	Proprietary Collins Aerospace LynxOS-178 Dynamic Loadable Device Driver (XCOFF)	NAND FS driver
`drmlite.dldd`	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	Device Resource Manager
`pdkm.dldd`	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	Graphics Engine related driver
`sda`	Proprietary Collins Aerospace LynxOS-178 User Binary (XCOFF)	SDA (Simple Display App - Field Software load/validation)

Approach

The top priority for this research is to ensure the technical accuracy of the claims presented herein.

As a result, IOActive decided that both the firmware and the security issues found would be analyzed, documented, and reported solely based on the disassembled code, without relying on a decompiler's output or an emulator. This avoids an additional layer of uncertainty derived from the use of a specific tool, which eventually might be called into question by the affected entities, as happened previously. This approach also facilitates the independent verification and reproduction of the results in a manner consistent with the scientific method.

This research is based on a static reverse engineering analysis of the exposed files listed in Tables 1 and 2, assisted by the information collected from publicly available materials, such as technical documents, presentations, maintenance manuals, patents, FAA/EASA publications, resumes, and training videos. These sources are referenced throughout the document.

Unfortunately, there is a lack of publicly accessible technical literature comprehensively detailing real-world vulnerabilities affecting either safety-critical avionics or more specifically Lynx178-OS-based deployments. Thus, IOActive believes it is important to document every step of this research as thoroughly as possible, to demonstrate the attack vectors as well as to bring some light into this opaque area of risk.

The AFDR-3700 system has been fully reverse engineered using IDA Pro[15], reconstructing the deterministic network configuration, execution flows and interactions between their different components, identifying the security boundaries, and eventually discovering security vulnerabilities that would allow a malicious actor to compromise the AFDR-3700, thus taking control of the AFD-3700 DUs and its functional applications.

This technical document is intended to comprehensively detail these efforts, such that it can be used to demonstrate the feasibility, validity, and reproducibility of the identified security issues as well as the potential safety impacts.

[15] IDA Pro - https://hex-rays.com/ida-pro/

Attack Surface

In the context of an IMA architecture, the focus of this work has been put on finding those attack vectors that would enable either remote or inter-partition exploitation of safety-critical certified avionics during any phase of a flight. Thus, attack vectors requiring physical access through USB or maintenance connectors as well as those depending on an active 'on-ground' discrete signal (see image below) were excluded from the priorities.

Figure 9. Data Loading Capabilities in Pro Line Fusion Suite

This means that data loading attacks were not considered (all data loading needs to be performed while the aircraft is on ground) despite being an otherwise valid attack vector actively evaluated by the aviation industry. The main reason behind this decision is our past experience with Boeing 787 research[16]. IOActive discovered a significant number of issues in the ARINC615 and ARINC665 (data loading standards) implementation, but unfortunately, the inherent mitigations for this attack surface were used to discredit that research, regardless of whether they were applicable. It is also worth noting that one of the main

[16] https://ioactive.com/arm-ida-and-cross-check-reversing-the-787s-core-network/

arguments employed against that research's conclusions was that the kind of security issues found in non-certified systems would never occur in certified avionics.

Although this research does not cover the data loading attack surface in detail, analysis of the binaries involved[17] revealed that the security posture of the data loading logic implemented in the AFDR-3700 is not any better: it lacks any kind of cryptographically secure logic to validate the integrity and authenticity of the loadable software parts, neither of which are encrypted or signed, thus relying on CRC only.

However, as will be elaborated, the devices and network infrastructure involved in the data loading functionalities (including airborne navigation databases) are actually considered as part of a plausible attack path.

This research was not focused on finding as many issues as possible, as it does not provide any actual value beyond a certain point. Instead, the priority was to find a minimum set of those vulnerabilities and logic issues that allow an attacker to bypass the implemented security boundaries in a safety-critical certified avionics product.

[17] 'sda', 'hm_main' and 'arinc615a'

Impact and Safety Implications

The following section elaborates the approach IOActive followed to demonstrate that the AFD-3700 is a DAL-A device providing actual safety-critical functionalities. This is an important topic in this research, as entities may adduce that the AFD-3700 is certified as a DAL-A merely due to a specific customer request, but actually its functionality is not aligned with a safety-critical certification.

A. FLIGHT DISPLAY SYSTEM

The Rockwell Collins Electronic Flight Instrumentation System (EFIS) consists of an two touchscreen-enabled AFD-3700 Primary Flight Display (PFD) on the pilot's and copilot's panels and a touchscreen-enabled AFD-3700 Multi-Function Display (MFD) located in the center of the panel. Each display includes molded finger grips and may be operated with gloves. Each display is capable of being configured to display information in full screen, half screen and quarter screen windows.

The PFD includes primary attitude, heading, altitude, airspeed, navigation, flight guidance and pilot selectable formats.

Figure 10. King Air 250 Specification[18]

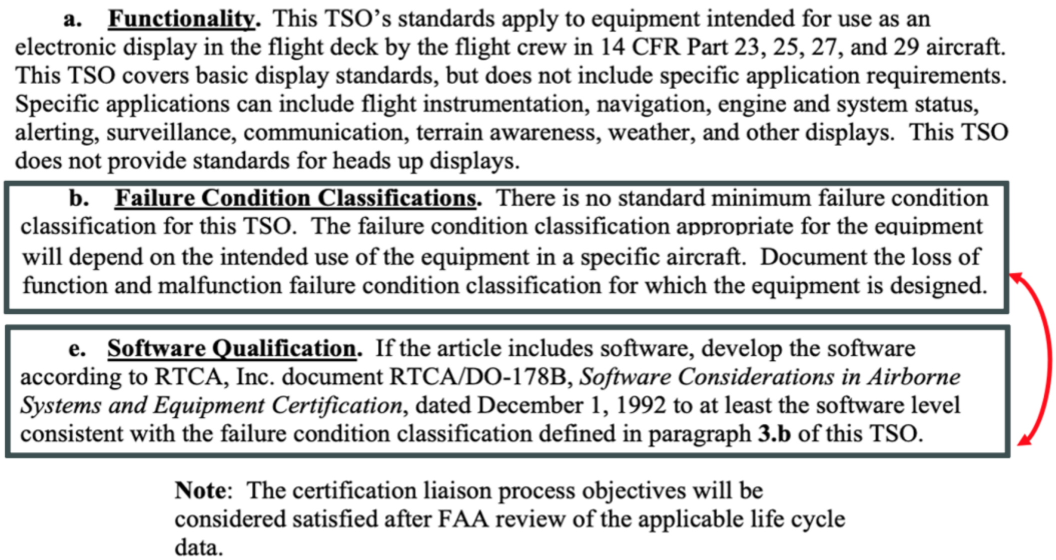
As illustrated in Figure 11, the AFD-3700 is authorized according to the TSO-C113a[19] , the FAA's Technical Standard Order for airborne multipurpose electronic displays intended for use as an electronic display in the flight deck.

Figure 11. Detail of the AFD-3700 Nameplate

From the requirements that TSO defines, we can highlight the following:

[18] http://www.africair.com/wp-content/uploads/2016/03/SD-KA250-Unit-250-to-TBD-2015-Oct.pdf

[19] http://rgl.faa.gov/Regulatory_and_Guidance_Library/rgTSO.nsf/0/dd968e96d184041e862579f10070b452/$FILE/TSO-113a.pdf



a. Functionality. This TSO's standards apply to equipment intended for use as an electronic display in the flight deck by the flight crew in 14 CFR Part 23, 25, 27, and 29 aircraft. This TSO covers basic display standards, but does not include specific application requirements. Specific applications can include flight instrumentation, navigation, engine and system status, alerting, surveillance, communication, terrain awareness, weather, and other displays. This TSO does not provide standards for heads up displays.

b. Failure Condition Classifications. There is no standard minimum failure condition classification for this TSO. The failure condition classification appropriate for the equipment will depend on the intended use of the equipment in a specific aircraft. Document the loss of function and malfunction failure condition classification for which the equipment is designed.

e. Software Qualification. If the article includes software, develop the software according to RTCA, Inc. document RTCA/DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, dated December 1, 1992 to at least the software level consistent with the failure condition classification defined in paragraph **3.b** of this TSO.

Note: The certification liaison process objectives will be considered satisfied after FAA review of the applicable life cycle data.

Figure 12. TSO C113a Requirement Details

An entity applying for the TSO C113a approval would need to define the Failure Condition Classifications as well as the Software Qualification, bearing in mind that both should be consistent with each other. Basically, this means that it should not be reasonable to apply for a TSO C113a approval by stating that a Primary Flight Display is providing the pilots with attitude indication, while its Software Qualification is DO-178B/C DAL-D (a failure will have a minor effect on the aircraft, crew, or passengers).

IOActive does not have access to the Collins safety analysis documents that were shared with the FAA as part of their application for the TSO C113a. However, we can use certain information to confirm that the Software Qualification for the AFD-3700 is DAL-A, which could then be used to infer the Failure Condition Classifications, and vice-versa.

These are the four elements that we will use to perform this task:

- Exposed files
- Resumes (from publicly available websites)
- FAA's Advisor Circular 25-11B
- FAA's Airworthiness Directives

Exposed Files

As illustrated in Figure 7 and Figure 11. Detail of the AFD-3700 Nameplate, the product is certified for DO-178B A/D. Now the task is to demonstrate that the DAL-D certification is not aligned with the main functionality performed by the AFDR-3700, in order to prove the AFDR-3700 is actually DAL-A software.

Based on the analysis of the exposed files, it is possible to determine that the following example applications and file systems[20] depend on the integrity of AFDR-3700 to run properly.

Applications:

- ATF-3500 (Advanced Terrain Functions)
- EICAS-6000 (Engine Indication Crew Alerting System) (Figure 14, VCT1648)

Figure 13. EICAS-6000 Showing an Engine Fire Alert[21]

- RTSA-6000 (Radio Tuning Software Application) (Figure 14, VCT409)
- FDSA-6500 (Flight Display System Application) (See Table 1. Exposed files COL_Table01.012)

Airborne Navigation Databases:

- SVS-RWY (Synthetic Vision System - Airport/Runway) (Figure 15, VCT363)
- SVS-OBST (Synthetic Vision System - Obstacles) (Figure 15, VCT1265)
- HRTDB (Terrain Awareness Warning System - High Resolution Terrain Database) (Figure 15, VCT1322)

Filesystems:

- Onboard Maintenance System Application
- Onboard Data Loader Application

[20] The functional applications and file systems depend on the integrity of AFDR-3700, so if it is compromised via a VM0 exploit as it is herein described, then it would be possible to take control of them.
[21] https://www.youtube.com/watch?v=jwUdYwIyWIw&list=PLMBKNyGwDnjoiGp6R5QxtfR9VCHUPl4X1

- Onboard Maintenance System Tables
- IMA Configuration Index Table (ICIT)

```
/////////////////////////////
// Virtual Machine Table
/////////////////////////////

<VM0>                                                   // VCT78
GroupIds=;                                              // VCT1187
LogicalName=AFDR-3700;                                  // VCT1188
CommandLine=/usr/bin/app_launcher;                      // VCT1189
EnvironmentVars=HealthMonitorIndex=255
        Field_Load_Id_List = AFDR:IMAT:ICIT:RTSA:EICAS:ECDA:OMSA:ODLA:OMST:ECL-DB:OMSTAR
        PctPathFname=/usr/etc/vm0.pct;                  // VCT1190
StdInNodeFname=/dev/null;                               // VCT1191
StdOutNodeFname=/dev/null;                              // VCT1192
StdErrNodeFname=/dev/null;                              // VCT1193
WorkingDir=/;                                           // VCT1194
RamFsMount=/tmp/;                                       // VCT1195
RamFsLim=1048576;                                       // VCT1196
RamFsNumOfInodes=24;                                    // VCT1197
ActionOnVmErr=0;                                        // VCT1198
SysRamMemLim=52428800;                                  // VCT1203
PersStorOnLocalLim=512;                                 // VCT1206
</VM0>

<VM1>                                                   // VCT79
GroupIds=;                                              // VCT408
LogicalName=RTSA-6000;                                  // VCT409
CommandLine=/usr/bin/app_launcher;                      // VCT410
EnvironmentVars=PctPathFname=/mnt/rtsa/rtsa.pct
        HealthMonitorIndex=255;                         // VCT411
StdInNodeFname=/dev/null;                               // VCT412
StdOutNodeFname=/dev/null;                              // VCT413
StdErrNodeFname=/dev/null;                              // VCT414
WorkingDir=/mnt/rtsa/;                                  // VCT415
RamFsMount=/tmp/;                                       // VCT416
RamFsLim=0;                                             // VCT417
RamFsNumOfInodes=4;                                     // VCT418
ActionOnVmErr=3;                                        // VCT419
SysRamMemLim=6291456;                                   // VCT424
NumOfProcessesLim=10;                                   // VCT1218
NumOfThreadsLim=10;                                     // VCT846
NumOfTimersLim=4;                                       // VCT847
PersStorOnLocalLim=8192;                                // VCT848
FsCacheLim=163840;                                      // VCT851
FsCacheAttr=WriteThrough;                               // VCT852
NumOfOpenFdsPerVmLim=256;                               // VCT853
NumOfMsgQueuesLim=2;                                    // VCT854
NumOfPipesLim=2;                                        // VCT855
NumOfSharedMemObjsLim=1;                                // VCT856
NumOfSemaphoresLim=100;                                 // VCT857
</VM1>

<VM2>                                                   // VCT80
GroupIds=;                                              // VCT1647
LogicalName=EICAS-6000;                                 // VCT1648
CommandLine=/usr/bin/app_launcher;                      // VCT1649
EnvironmentVars=PctPathFname=/mnt/eicas/eicas.pct
```

Figure 14. `S1-SL03.vct` *- AFDR-3700 file showing functional applications*


```
<FS2>                                                    // VCT502
Mount=/mnt/Apt_Rwy/;                                     // VCT363
NumOfInodes=9;                                           // VCT365
MkffsArgs=-F 0;                                          // VCT366
FfsckArgs=-F -r;                                         // VCT367
OwnerId=1;                                               // VCT368
GroupId=1;                                               // VCT369
Permissions=0400;                                        // VCT370
IntegrityAttr=;                                          // VCT371
DataloadHdrPath=;                                        // VCT372
Size=32;                                                 // VCT2502
Chip=1;                                                  // VCT2503
</FS2>

<FS3>                                                    // VCT503
Mount=/mnt/Obstacles/;                                   // VCT1265
NumOfInodes=6;                                           // VCT1267
MkffsArgs=-F 0;                                          // VCT1268
FfsckArgs=-F -r;                                         // VCT1269
OwnerId=1;                                               // VCT1270
GroupId=1;                                               // VCT1271
Permissions=0400;                                        // VCT1272
IntegrityAttr=;                                          // VCT1273
DataloadHdrPath=;                                        // VCT1274
Size=64;                                                 // VCT2504
Chip=1;                                                  // VCT2505
</FS3>

<FS4>                                                    // VCT1321
Mount=/mnt/atf-svs-config/;                              // VCT1254
NumOfInodes=32;                                          // VCT1256
MkffsArgs=-F 0;                                          // VCT1257
FfsckArgs=-F -r;                                         // VCT1258
OwnerId=1;                                               // VCT1259
GroupId=1;                                               // VCT1260
Permissions=0400;                                        // VCT1261
IntegrityAttr=HardwareNonCritical;                       // VCT1262
DataloadHdrPath=/mnt/atf-svs-config/;                    // VCT1263
Size=32;                                                 // VCT2506
Chip=0;                                                  // VCT2507
</FS4>

<FS5>                                                    // VCT1322
Mount=/mnt/atf-hi-resolution-terrain/;                   // VCT1243
NumOfInodes=1000;                                        // VCT1245
MkffsArgs=-F 0;                                          // VCT1246
FfsckArgs=-F -r;                                         // VCT1247
OwnerId=1;                                               // VCT1248
GroupId=1;                                               // VCT1249
Permissions=0400;                                        // VCT1250
IntegrityAttr=;                                          // VCT1251
DataloadHdrPath=;                                        // VCT1252
Size=3776;                                               // VCT2508
Chip=1;                                                  // VCT2509
</FS5>
```

Figure 15. `S1-SL04.vct` – Mounted filesystems

Resumes

The following extracts from the publicly available resumes of Collins' engineers provide a clear indication that the EFIS project, and thus the AFDR-3700, in the Pro Line Fusion product line is being developed following DAL-A standards (core applications such as EICAS or FDSA may be certified as DAL-B or above)

Rockwell Collins
5 yrs
Hyderabad Area, India

Senior Engineer
Aug 2014 - Jun 2015 · 11 mos

Engineer
Aug 2012 - Jul 2014 · 2 yrs

Projects: Bombardier CSeries, Gulfstream G280, Bombardier M145, Embraer Executive Jet EEJ, Embedded Display System EDS:

• Electronic Flight Instrument System (EFIS):
These projects are related to Display Systems and come under Electronic Flight Instrument System (EFIS).
I was involved in verification of software component in Electronic Flight Instrument System. The purpose of verification is to verify the compliance of the software to DO-178B standards (Level A). The verification includes Requirement Based Test Coverage Analysis, Structural Coverage Analysis and Traceability Analysis. The verification will primarily be via implementation and execution of Test Procedures/Scripts in qualified tools. Software Software Integration Testing(SSIT) is performed for the verification activities and Structural Coverage Analysis Reports are generated.

Actively involved in:
• Writing of Test Cases and Test Procedures in Python and XML corresponding to the requirements.
• Using MATLAB to verify requirements against the model.
• Generate the Structural Coverage Analysis Report corresponding to the Test Procedure/Scripts so as to verify the compliance with DO-178B standards (Level A).
Performing Peer Reviews for the Test Case, Test Procedures, Test Results and Structural Coverage Reports.

Figure 16. Resume of Engineer #1

Software Engineer
Rockwell Collins
Aug 2010 - Present · 11 yrs 7 mos
Cedar Rapids, Iowa Area

• Two years of experience as a software and system development, verification and validation engineer

• Verifying EFIS software for Embraer Legacy 450/500(EEJ) Pro Line Fusion® Avionics Suite using DO-178B Level A Standards
- Test correct functionality and establish conditions that reveal potential errors
- Task includes interpreting FDSA software requirements, capturing test cases in DOORs, writing automated/visual test procedures in python or xml, verifying and/or analyzing code for missing coverage, reviewing test cases and test procedures for correct implementation and compliance to standards.

• Developed test procedures to verify system requirements using ARP4754A (2012)
Aircrafts Supported: Bombardier CSeries/Learjet 85, Mitsubishi Regional Jet (MRJ), Embraer Legacy 450/500 (EEJ)
- Served as the domain focal for FDSA
- Designed test procedures which are ran on standalone cockpit rigs and/or test stations for verification

• Developed EFIS software for Pro Line Fusion® Avionics Suite using DO-178B Level A Standards
Aircrafts Supported: Bombardier Global Express XRS/Global 5000/CSeries, MRJ, Gulfstream G250 and EEJ

Figure 17. Resume of Engineer #2

FAA's Advisor Circular 25-11B

The FAA's Advisor Circular 25-11B provides a guidance for design, integration, installation approval of electronic flight deck displays[22], which will be used to check the consistency between the safety assessment required by the Software Failure Conditions and the Software Qualification.

The following examples on the hazard classification level can be linked directly to some of the scenarios that can be achieved by compromising the AFDR-3700 (see Figure 90. Scenario for a Compromised AFDR-3700) which provide the malicious actor the ability to maliciously influence the functional applications (e.g. EICAS and FDSA) that depend on it. At this point we should recall that catastrophic failures in the Failure Condition Classifications would require a DAL-A Software Qualification to be consistent.

Table 4-1. Example Safety Objectives for Attitude Failure Conditions

Failure Condition	Hazard Classification	Qualitative Probability
Loss of all attitude displays, including standby display	Catastrophic	Extremely Improbable
Loss of all primary attitude displays	Major – Hazardous[1]	Remote – Extremely Remote[1]
Display of misleading attitude information on both primary displays	Catastrophic	Extremely Improbable

Figure 18. Hazard Classification Level for Display of Misleading Attitude Information

Table 4-7. Example Safety Objectives for Engine Failure Conditions

Failure Condition	Hazard Classification	Qualitative Probability
Loss of one or more required engine indications for a single engine	Major	Remote
Misleading display of one or more required engine indications for a single engine	Major	Remote
Loss of one or more required engine indications for more than one engine	Hazardous	Extremely Remote[1]
Misleading display of any required engine indications for more than one engine	Catastrophic	Extremely Improbable[2]

Figure 19. Hazard Classification Level for Display of Misleading Engine Information

[22] https://www.faa.gov/documentlibrary/media/advisory_circular/ac_25-11b.pdf

The following point in the guidance relates to a Windowing architecture.

4.6.9 For those systems that integrate windowing architecture into the display system, a means should be provided to control the information shown on the displays, such that the integrity of the display system as a whole will not be adversely impacted by anomalies in the functions being integrated. This means of controlling the display of information, called window manager in this AC, should be developed to the software assurance level at least as high as the highest integrity function of any window. For example, a window manager should be level "A" if the information displayed in any window is level "A" (see RTCA DO-178C, *Software Considerations in Airborne Systems and Equipment Certification*). SAE ARP4754A, *Guidelines for Development of Civil Aircraft Systems*, provides a recommended practice for system development assurance.

Figure 20 25-11B guidance

We can directly match the point above with the resume in Figure 21, where the DS6000 Window Manager application is developed under the DAL-A standard, meaning that at least one of the windows contains DAL-A data.

Senior Software Engineer
Jan 2010 - Oct 2013 · 3 yrs 10 mos

•Commercial: Displays Applications Department
•Project: DS6000 Window Manager / Nav Master (DWM/NM) Applications for Proline Fusion (DO-178 Level A)
•Lead Developer and Architect for Pro Line Fusion Navigation Master Applications (Model Based Development)
•Defined high and low level requirements and provided linking to implementation using DOORS for DWM and NM
•Develop DWM/NM software both logical and graphical using ARINC 661, Matlab/Simulink and VAPS XT 661
•Peer reviews DWM/NM software implementation and code review
•Support SOI audits by providing thread analysis and overview of peer review history
•Support DWM/NM Verification team including coverage analysis using LDRA
•Helped setup working relationship with Rockwell's new offsite India Design Center (IDC) by traveling to Hyderabad, India in July 2010
•Train new DWM/NM engineers

Figure 21 Resume from Engineer #3

VAPS XT[23] is a safety-critical DO-178B/C DAL-A HMI for avionics systems, which is being used as part of the development of functional applications for the Pro Line Fusion DUs.

[23] https://www.presagis.com/en/product/vaps-xt/

FAA's Airworthiness Directives

It was also possible to confirm that the AFD-3700 sustains safety-critical functionality by consulting the Airworthiness Directive database published by the FAA:

1. A potential failure in the ASIC of the AFD-3010 (a previous version of the AFD-3700) required the release of an Airworthiness Directive[24] (AD) in 2002.

Summary	This amendment adopts a new airworthiness directive (AD) that applies to certain Rockwell Collins, Inc. (Rockwell Collins) AFD-3010 adaptive flight display units that are installed on aircraft. This AD requires you to inspect the AFD-3010 unit to determine if it contains an MFP386 Application Specific Integrated Circuit (ASIC) device with a date code of 0128. This AD also requires you to have any AFD-3010 units with an MFP386 device with a date code of 0128 modified. This AD is the result of reports of a manufacturing defect. The actions specified by this AD are intended to prevent premature failure of the ASIC, which could result in the AFD-3010 unit displaying erroneous primary flight and engine parameter information. Such failure could lead to the pilot using incorrect information when making critical flight safety decisions.

Figure 22. Summary of the AD for the AFD-3010

2. A potential failure in the FDSA-6500 functional application (One of the applications depending on the AFDR-3700, see Table *1*. Exposed files) required the release of an AD[25] from the FAA/EASA in 2019, to address an "unsafe condition."

SUMMARY:

The FAA is superseding Airworthiness Directive (AD) AD 2019-12-09 for certain Rockwell Collins, Inc. (Rockwell Collins) FDSA-6500 flight display system applications installed on airplanes. AD 2019-12-09 imposed operating limitations on the traffic collision avoidance system (TCAS). AD 2019-12-09 was prompted by conflict between the TCAS display indications and aural alerts that may occur during a resolution advisory (RA) scenario. This AD retains the requirements of AD 2019-12-09 until a software upgrade is completed. The FAA is issuing this AD to address the unsafe condition on these products.

Figure 23. Summary of the AD for the FDSA-6500

[24] https://www.govinfo.gov/app/details/FR-2002-10-16/02-25717/summary

[25] https://www.federalregister.gov/documents/2021/03/25/2021-06156/airworthiness-directives-rockwell-collins-inc-flight-display-system-application

This AD provides a clear description of the safety problem:

> **(e) Unsafe Condition**
>
> This AD was prompted by a conflict between the traffic collision avoidance system (TCAS) primary display indications and aural alerts during a resolution advisory (RA) scenario. The FAA is issuing this AD to prevent conflicting TCAS information, which could result in the pilot under-correcting or over-correcting and may lead to inadequate aircraft separation and a mid-air collision.

Figure 24 Unsafe Condition description

Obviously, this kind of catastrophic error can only be caused by a failure of a DAL-A software, assuming there is no single point of failure in safety-critical avionics.

Thus, it is reasonable to assume our initial premise of the AFDR-3700 being an actual DAL-A sustaining safety-critical functionality is correct, as we have that:

- The FDSA-6500 is a DAL-A application, managed by a DAL-A Window Manager, running in a DAL-A device.
- The DAL-A FDSA-6500 functional application can only rely on a DAL-A AFDR-3700 according to the "Rely-Guarantee" model, used in certification of modular systems. This means that application X (FDSA-6500) is guaranteed to access the resources provided by system Y (in this case the AFDR-3700). This must be true, otherwise it could not be certified as application X (DAL-A) would be relying on a system Y that is certified using a lower level (such as DAL-D). That situation does not guarantee the proper functioning of application X, which breaks the model.

Also, the AFD-3700 DUs are generally part of the Master Minimum Equipment List (MMEL) of a Pro Line Fusion-equipped aircraft.

U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION

MASTER MINIMUM EQUIPMENT LIST

AIRCRAFT: Textron Aviation Model 300	REVISION NO. 10 DATE: 04/27/2018	PAGE NO. 34-5

MMEL TABLE KEY

SYSTEM & SEQUENCE NO. — ITEM — 1. REPAIR CATEGORY — 2. NUMBER INSTALLED — 3. NUMBER REQUIRED FOR DISPATCH — 4. REMARKS OR EXCEPTIONS

34. NAVIGATION

Sequence No.	Item	1	2	3	4	Change Bar
-25-02	Adaptive Flight Display System (AFD) (Collins Pro Line Fusion Equipped Airplanes)					|
-01	Single Pilot (PFD 2 Control, PFD 2 Fan Inop, PFD Fan 1 Inop messages)	**B**	**3**	**2**	May be inoperative provided PFD 1 and the MFD are operative	|
-02	Dual Pilot (MFD Control, MFD Fan Inop messages)	**B**	**3**	**2**	May be inoperative provided PFD 1 and PFD 2 are operative.	|

Figure 25 MMEL Textron Aviation Model 300[26]

Potentially Affected Aircraft

Based on reputable publicly available information, the list of those aircraft potentially equipped with the impacted version of the Pro Line Fusion suite[27] may include, but is not limited to:

- Embraer Legacy 450/500[28] (Business)
- Gulfstream G280[29] (Business)
- Bombardier Global 5000/6000[30] (Business)
- Bombardier Challenger 604[31] (Business)

[26] https://fsims.faa.gov/wdocs/mmel/be-300_rev_10.pdf
[27] It does not mean all these aircraft are vulnerable. This requires to be evaluated on a case-by-case basis.
[28] https://www.collinsaerospace.com/what-we-do/Business-Aviation/Flight-Deck/Pro-Line-Fusion/Embraer-Legacy-450-500
[29] https://www.collinsaerospace.com/what-we-do/Business-Aviation/Flight-Deck/Pro-Line-Fusion/Gulfstream-G280-With-Pro-Line-Fusion-And-HGS
[30] https://www.collinsaerospace.com/what-we-do/Business-Aviation/Flight-Deck/Pro-Line-Fusion/Pro-Line-Fusion-For-Bombardier-Global-5000-6000
[31] https://www.collinsaerospace.com/what-we-do/Business-Aviation/Platforms/Bombardier/Challenger-604/Avionics

- Beechcraft King Air[32] (Military/Business)
- Cessna Citation CJ1+, CJ2+, and CJ3[33] (Business)
- Viking Air CL-125T, CL-415[34] (Firefighting)
- Embraer KC-390[35] (Military)

IOActive selected reputable, published sources for the above information such as company websites to compile this list, we recognize not all reputable sources are created accurate or remain accurate as time progresses.

Disputed statement 1

A pre-publication version of the paper shared with Collins Aerospace contained a list of affected aircraft, based on publicly available information.

Collins Aerospace explicitly communicated to IOActive in a letter dated April 7, 2022 that:

- The list was incorrect.
- A corrected list of the affected aircraft will not be provided as it is not necessary to support the research.

IOActive considers that this information is certainly necessary to support the research, as it provides a valuable information about its impact.

That original list included certain commercial and military Airbus models, which have been removed from this current list, according to some consistent information received from different sources.

If any additional information is received, that clearly demonstrates this list is still incorrect, IOActive will proceed to update the paper accordingly, also publicly rectifying if required.

[32] https://www.collinsaerospace.com/what-we-do/Business-Aviation/Flight-Deck/Pro-Line-Fusion/Pro-Line-Fusion-Upgrade-For-Beechcraft-King-Air

[33] https://www.collinsaerospace.com/what-we-do/Business-Aviation/Flight-Deck/Pro-Line-Fusion/Pro-Line-Fusion-Upgrade-For-Citation-Cj3

[34] https://www.ainonline.com/aviation-news/business-aviation/2019-03-19/viking-launches-avionics-upgrade-its-fire-bombers

[35] https://www.collinsaerospace.com/-/media/project/collinsaerospace/collinsaerospace-website/product-assets/marketing/k/kc-390-brochure-0711.pdf?rev=787c1c35ebdd4cbebb2365fdd748b686

Technical Analysis

Reverse Engineering Notes

The KDI (`COL_Application01.001`) contains a symbol table where each entry is 0x12 bytes (see Figure 27). The first 8 bytes hold the symbol name followed by its address. If the symbol name length is longer than 8 bytes, the first 4 bytes are then NULL and the next 4 bytes contain an offset into an array of strings where the symbol name can be resolved (see Figure 26).

For the remaining binaries (XCOFF), the symbols and debug information were found in VM0's `hm_main` as well as in most of the drivers.

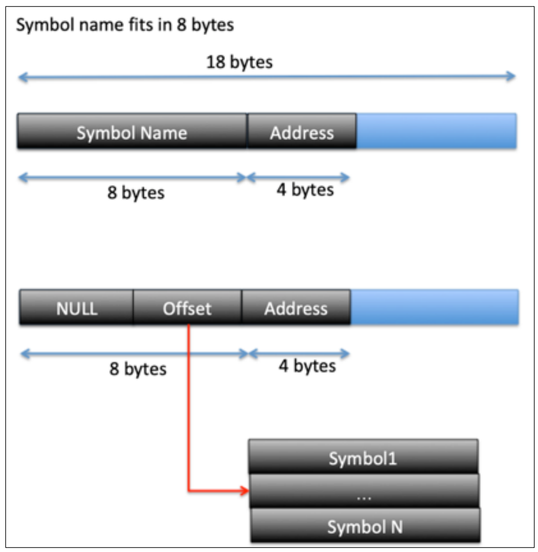


Figure 26. Kernel Symbol Table Structure

```
     Symbol Name          Address
  0  2E73 7461 7274 0000  B001 0020 0001 0000 0200  .start   ∞
 18  2E67 6574 5F74 6561 B001 0288 0001 0000 0200  .get_tea∞   à
 36  2E73 6574 5F74 6F63 B001 0290 0001 0000 0200  .set_toc∞   ê
 54  0000 0000 0000 0004 B001 0298 0001 0000 0200           ∞   ò
 72  7374 6172 7400 0000 B010 5280 0002 0000 0200  start    ∞ RÄ
 90  6765 745F 7465 6100 B010 5290 0002 0000 0200  get_tea  ∞ Rê
108  7365 745F 746F 6300 B010 52A0 0002 0000 0200  set_toc  ∞ R†
126  0000 0000 0000 0013 B010 52B0 0002 0000 0200           ∞ R∞
144  2E72 6B5F 626F 6F74 B001 0760 0001 0020 0200  .rk_boot∞   `
162  524F 4D6B 0000 0000 B010 0000 0002 0000 0200  ROMk     ∞
180  524F 4D6B 7000 0000 B010 00C4 0002 0000 0200  ROMkp    ∞   ƒ
198  726B 5F62 6F6F 7400 B010 5320 0002 0000 0200  rk_boot  ∞ S
216  2E6C 6F77 496E 6974 B001 1000                 .lowInit∞
```

Figure 27. Detail of Kernel Symbol Table

It was possible to infer the PowerPC family through one of the CPU Support Package (CSP) functions in the kernel (see Figure 28).

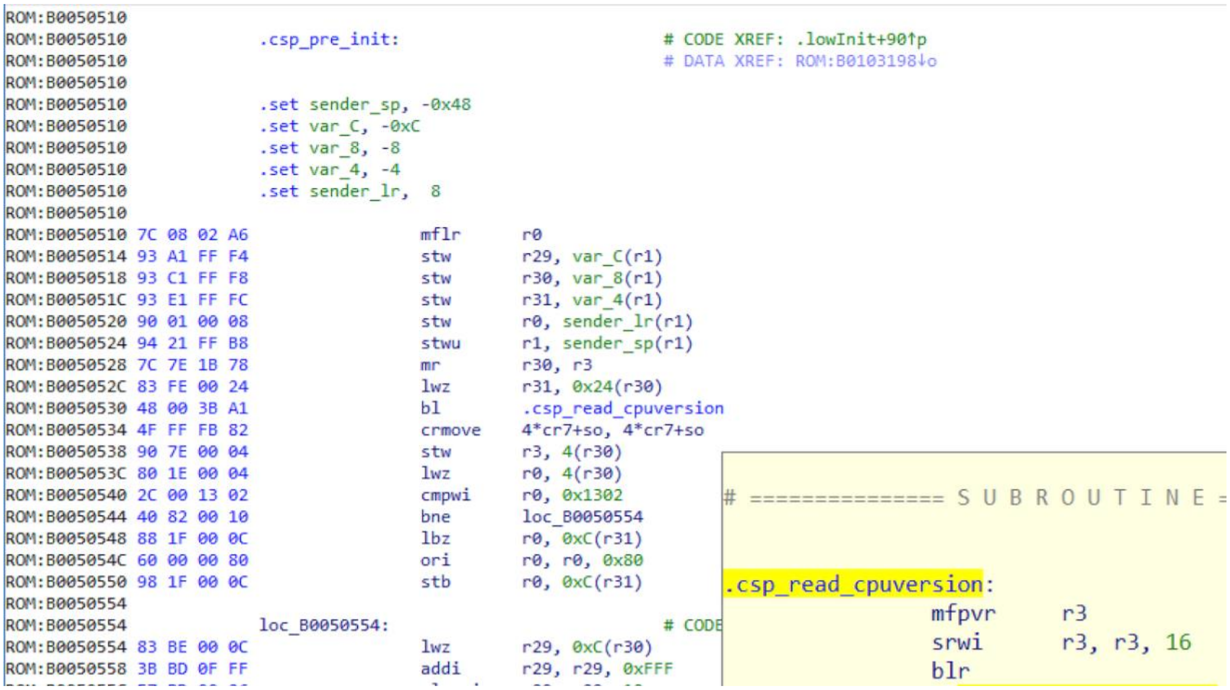


Figure 28. Kernel `csp_pre_init` Function

At `0xB0050540` the CPU ID 0x1302 indicates an AMCC PowerPC 440EP. This is also corroborated by the register values used during the initialization of the on-chip Ethernet MAC controller in the pcie.dldd driver, which corresponds to the PowerPC 4XX family.

Attacking a LynxOS-178-based System

What is LynxOS-178?

LynxOS-178 is Lynx Software Technologies Inc.'s Real-Time Operating System (RTOS) for safety-critical systems. Lynx Software Technologies, Inc. is the premier developer of POSIX conformant real-time operating systems. Our flagship product, called LynxOS, is in use in hundreds of thousands of installations where high reliability and hard real-time determinism are essential. LynxOS-178 is based on LynxOS and has the features necessary for safety-critical applications such as aviation, defense, medicine, along with other business-critical fields. Along with the operating system and the development tools, Lynx Software Technologies can optionally provide the necessary artifacts to permit LynxOS-178 to be used in systems that are certifiable up to level A of the RTCA DO-178C standard. In addition, LynxOS-178 provides the ability to run multiple levels of DO-178C criticality on the same platform.

Figure 29. LynxOS-178 Description (Extracted from LynxOS-178 documents[36] Found at GitHub)

From a functional and security perspective, a LynxOS-178 target is more similar to any modern desktop OS than the usual RTOS found in most Common-Off-The-Shelf (COTS) embedded devices (see Figure 29).

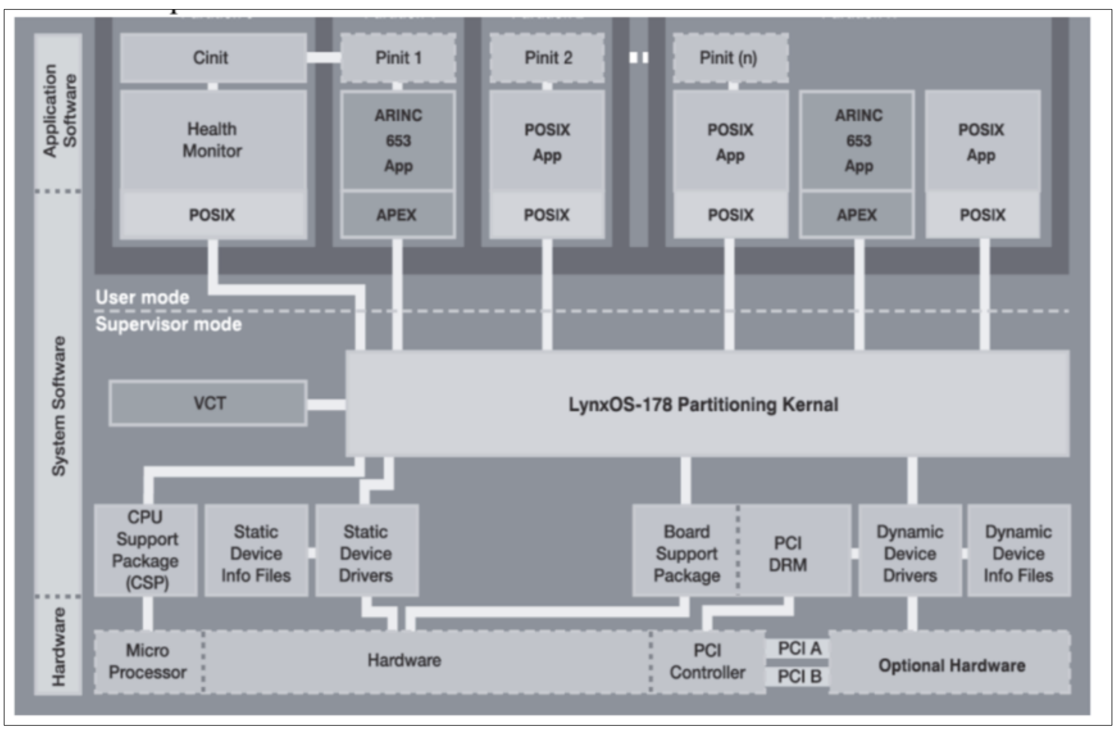


Figure 30. LynxOS-178 Architecture (Extracted from Leaked LynxOS-178 Documents[36])

[36] https://github.com/blackqbit/lynxos-178_arm_docs/blob/main/2203-00_los178_ig.pdf

It is highly recommended to review the documents referenced in Figure 30 to get a complete understanding of the LynxOS-178 environment. At a high level, there are four important concepts that need to be briefly introduced to provide the required context:

1. **VCT files**

Virtual Machine Configuration Table

The Virtual Machine Configuration Table (VCT) is a file that contains configuration information for the target system running LynxOS-178. The contents of the VCT should be thought of as a well-defined set of descriptors that configures the LynxOS-178 Operating System. It supports the ability to create partitions (also known as virtual machines (VMs)) and configure each partition to match its design as determined by the user.

Figure 31. VCT Definition

It is important to clarify that despite the naming conventions, LynxOS-178 is not a hypervisor. The VM concept in this context is similar to the process concept in any modern desktop OS: neither memory nor resources are shared between the VMs. From now on, the VM term will be used according to the LynxOS-178 specification.

2. **VM0**

VM0 is a unique VM with special privileges. These privileges are similar to the root privileges in a UNIX system. For example, VM0 can override protections set in other VMs and can reboot the computer. In addition, VM0 monitors the state of the processes and threads contained within the other VMs. This is crucial to understand the implications of this research because we are exploiting an application running in VM0, so a successful attack leads to complete control over the AFD-3700 system, as will be elaborated in the coming sections.

3. **Inter-Partition Mechanisms**

As defined by ARINC-653 inter-partition communication (communication between VMs) is based on message passing through message ports. These messages are exchanged through channels, which are a logical link between a source VM and one or more destination VM. In the context of the LynxOS-178, the different VMs can send and receive messages through multiple channels via defined access points, called ports (queuing or sampling).

The standard does not define the underlying transport mechanism, so it is transparent to the applications, allowing ARINC 653 applications to communicate in the same way regardless of whether they run on the same shared computing resource or even across an AFDX avionics network. These communication flows are fully deterministic and are statically defined as part of the system configuration process.

The analysis of this implementation (developed by Collins Aerospace), including its configuration, has been a core part of this research as it helped to demonstrate the plausible attack paths.

4. **Avionics System LAN**

   A Pro Line Fusion-equipped aircraft may be considered an e-Enabled aircraft, thus presenting certain functional similarities to other e-Enabled aircraft, such as the Boeing 787 or an Airbus A380. In this case, the AFDX network implemented by Collins Aerospace is called the ‘Avionics System LAN.’ In this network we can find the usual components, such as AFDX switches, data concentrators (IOC) and data loaders, as well as the AFD-3700 Dus obviously.

Security Boundaries

In order to bypass the security boundaries implemented in the AFDR-3700 we are required to uncover vulnerabilities that enable executing arbitrary code in a privileged domain, either VM0’s main app or kernel/drivers, coming from a less privileged partition (VM) or even remotely, through the Avionics System LAN.

In general terms, the ability to compromise a non-certified partition running DAL D/E applications (i.e. In-Flight Entertainment Systems) should be assumed. For the B/C levels, this task may be more difficult as the code requires additional certification requirements.

AFDR-3700 Boot Sequence

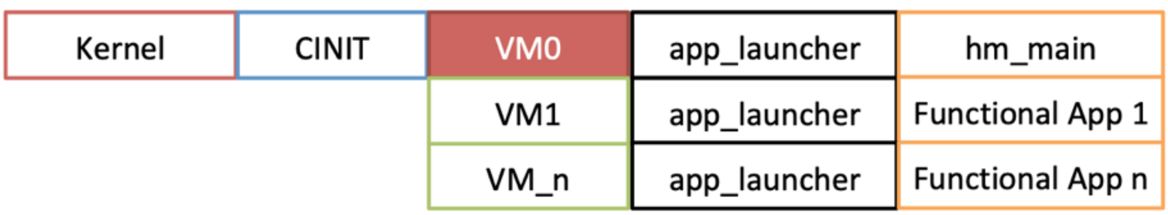


Figure 32. Regular Boot Sequence in AFD-3700

The boot sequence depicted in Figure 32 may vary according to the boot mode (AFDR-3700 defines six different boot modes described below) and its corresponding VCT, but the AFDR-3700 implements a common approach to launch the required VM applications.

`App_launcher` is the main binary that runs by default for any VM defined in the VCT file. Actually, this binary is in charge of parsing the Collins Aerospace’s `Process Configuration Table` file referenced by `PctPathFName` (only `vm0.pct` was present in the leaked files) and launching the corresponding application defined in it. This PCT file format is not documented, so it is considered a custom part added by Collins Aerospace to the VCT logic.

```
1  ////////////////////////////////////////////////////////////////////////////////
2  //  Virtual Machine Configuration Table
3  //
4  //  File Name: S1-SL03.VCT
5  //  Generated By: jaflore9
6  //  Generated On: 07 May 2014 18:45:55
7  //
8  //  CRs Implemented: FUSN00400510,
9  //
10 // (C)2014 Rockwell Collins.  All rights reserved.
11 //
12 ////////////////////////////////////////////////////////////////////////////////
13
14
15 //////////////////////////////
16 // Scalar Sub Table
17 //////////////////////////////
18
19 VctCrc=;                                                    // VCT2677
20 VctCpn=;                                                    // VCT161
21 VctVersion=150;                                             // VCT162
22 NumOfVms=6;                                                 // VCT163
23 NumOfDdds=3;                                                // VCT164
24 NumOfFs=19;                                                 // VCT165
25 ActionOnModuleErr=3;                                        // VCT166
26 IbitDuration=0;                                             // VCT171
27 NetworkInterface=Winsock2.2;                                // VCT173
28 ColdStartSchedule=;                                         // VCT174
29 ColdStartDuration=0;                                        // VCT175
30 RunTimeSchedule=0[1] 1[3] 2[7] 3[12] 5[2] 0[2] 1[2] 2[8] 3[11] 5[2];    // VCT178
31
32 //////////////////////////////
33 // Virtual Machine Table
34 //////////////////////////////
35
36 <VM0>                                                       // VCT78
37 GroupIds=;                                                  // VCT1187
38 LogicalName=AFDR-3700;                                      // VCT1188
39 CommandLine=/usr/bin/app_launcher;                          // VCT1189
40 EnvironmentVars=HealthMonitorIndex=255
41         Field_Load_Id_List = AFDR:IMAT:ICIT:RTSA:EICAS:ECDA:OMSA:ODLA:OMST:ECL-DB:OMSTAR
42         PctPathFname=/usr/etc/vm0.pct;                      // VCT1190
43 StdInNodeFname=/dev/null;                                   // VCT1191
44 StdOutNodeFname=/dev/null;                                  // VCT1192
45 StdErrNodeFname=/dev/null;                                  // VCT1193
46 WorkingDir=/;                                               // VCT1194
47 RamFsMount=/tmp/;                                           // VCT1195
48 RamFsLim=1048576;                                           // VCT1196
49 RamFsNumOfInodes=24;                                        // VCT1197
50 ActionOnVmErr=0;                                            // VCT1198
51 SysRamMemLim=52428800;                                      // VCT1203
52 PersStorOnLocalLim=512;                                     // VCT1206
53 </VM0>
```

Figure 33. *S1-SL03.vct*

In Figure 33 at line 42, we can see the reference to the `vm0.pct` file, which `app_launcher` has to parse in order to know the process that needs to be launched.

```
// PCT file for 460 EDS board to run hm_main in vm0.
// File: vm0.pct
// Note: PctCrc can not be left blank
PctCrc = 0x87E5DB9A;
///////////////////////////////////////////////////////////////////////////////
// PE0 - ApplicationX
// PExx   where xx is the sequential numbering (0, 1, 2) of the process.
//        Leading zeroes are suppressed.
///////////////////////////////////////////////////////////////////////////////
<PE0> // Start of Application Process 0 Table.
    // Process specific information
    CommandLine=/usr/bin/hm_main;

    EnvironmentVars=;  //Just use VCT settings

    // Device nodes to use for standard I/O streams.
    StdInNodeFname=/dev/null;
    StdOutNodeFname=/dev/rs232A_nonblocking;
    StdErrNodeFname=/dev/rs232A_nonblocking;

    // File systems info
    WorkingDir=/; // Home Directory
    // Process priority.  It doesn't mattersince hm_main sets it's own priority
    Priority=80;
</PE0> // End of Application Process 0
```

Figure 34. `vm0.pct`

As shown in Figure 34, the `vm0.pct` file contains the reference to the binary implementing the functional application that should be running in that specific VM, in this case `hm_main` for VM0.

AFD-3700 Health Monitor Application: hm_main

This is a Collins Aerospace's application which implements part of the Health Monitoring logic mandated by the ARINC 653 standard. In addition, it is the core user-mode application in the AFDR-3700 as it initializes, supervises, and controls key functionalities of the DU. Essentially, the AFD-3700 cannot run properly without a fully working `hm_main` application.

As previously mentioned, the VM0 partition is, by default, a privileged partition within the LynxOS-178 architecture. From a security perspective, this has several implications. By exploiting the `hm_main` application, we would gain control over key functionalities that can be used to fully compromise the entire LynxOS-178 deployment. For instance, once the ability to execute code in `hm_main` has been achieved, it is possible to directly load an arbitrary driver via the `dr_install` (see Figure 35) syscall, which requires the VM0's UID.

```
ROM:B0026D00             .dr_install:                          # DATA XREF: ROM:dr_install↓o
ROM:B0026D00
ROM:B0026D00             .set sender_sp, -0x60
ROM:B0026D00             .set saved_toc, -0x4C
ROM:B0026D00             .set var_28, -0x28
ROM:B0026D00             .set var_24, -0x24
ROM:B0026D00             .set var_1C, -0x1C
ROM:B0026D00             .set var_18, -0x18
ROM:B0026D00             .set var_14, -0x14
ROM:B0026D00             .set var_10, -0x10
ROM:B0026D00             .set var_C, -0xC
ROM:B0026D00             .set var_8, -8
ROM:B0026D00             .set var_4, -4
ROM:B0026D00             .set sender_lr,  8
ROM:B0026D00
ROM:B0026D00 7C 08 02 A6                 mflr     r0
ROM:B0026D04 93 21 FF E4                 stw      r25, var_1C(r1)
ROM:B0026D08 93 41 FF E8                 stw      r26, var_18(r1)
ROM:B0026D0C 93 61 FF EC                 stw      r27, var_14(r1)
ROM:B0026D10 93 81 FF F0                 stw      r28, var_10(r1)
ROM:B0026D14 93 A1 FF F4                 stw      r29, var_C(r1)
ROM:B0026D18 93 C1 FF F8                 stw      r30, var_8(r1)
ROM:B0026D1C 93 E1 FF FC                 stw      r31, var_4(r1)
ROM:B0026D20 90 01 00 08                 stw      r0, sender_lr(r1)
ROM:B0026D24 94 21 FF A0                 stwu     r1, sender_sp(r1)
ROM:B0026D28 7C 7A 1B 78                 mr       r26, r3
ROM:B0026D2C 81 22 00 18                 lwz      r9, 0x18(r2)
ROM:B0026D30 81 29 00 00                 lwz      r9, 0(r9)
ROM:B0026D34 A3 89 00 74                 lhz      r28, 0x74(r9)
ROM:B0026D38 2C 1C 00 00                 cmpwi    r28, 0      # UID == 0?
ROM:B0026D3C 41 82 00 0C                 beq      loc_B0026D48
ROM:B0026D40 38 60 00 01                 li       r3, 1
ROM:B0026D44 48 00 01 70                 b        loc_B0026EB4
```

...

```
ROM:B0026E0C             loc_B0026E0C:                       # CODE XREF: .dr_install+104↑j
ROM:B0026E0C 80 7A 00 00            lwz      r3, 0(r26)     # 1st parameter - path to driver file
ROM:B0026E10 38 80 00 00            li       r4, 0
ROM:B0026E14 4B FF A8 69            bl       .file_open
ROM:B0026E18 4F FF FB 82            crmove   4*cr7+so, 4*cr7+so
ROM:B0026E1C 7C 7F 1B 79            mr.      r31, r3
```

...

```
ROM:B0026EFC 7F E3 FB 78            mr       r3, r31
ROM:B0026F00 7F C4 F3 78            mr       r4, r30
ROM:B0026F04 7F A5 EB 78            mr       r5, r29
ROM:B0026F08 48 02 D7 D9            bl       .load_module_xcoff
```

Figure 35. `dr_install` Partial Implementation

Vulnerable SNMP Daemon in hm_main

With this information in mind, it seems clear that `hm_main` is a top priority. The initial analysis of the binary revealed a `snmpd` daemon, which was found to be vulnerable (see Figure 36) to a previously unknown vulnerability.

Curiously, this `snmpd` implementation is based on the code[37] provided in "TCP/IP Illustrated Volume 2 – the Implementation[38]." Although the PowerPC assembly presented herein partially matches the original code, some modifications have been added by Collins Aerospace developers; for instance, a bounds check in `.a1readlen`, which receives an additional parameter in comparison to the original implementation. Also, the dynamic memory allocated for the linked list in the original code has been moved to the stack[39] in the `hm_main` implementation. Finally, some fields in the internal structures have been removed.

This SNMP implementation is prone to, at least[40], a stack-based buffer overflow due to a lack of bounds checking in the `a1readoid` function while parsing Object Identifiers (OIDs).

[37] https://cis.temple.edu/~ingargio/cis307/software/TCPIP-vol2/snmp/
[38] https://en.wikipedia.org/wiki/TCP/IP_Illustrated
[39] Memory is statically allocated due to LynxOS-178 VMs deterministic constraints
[40] There are additional vulnerable paths that have not been elaborated in this paper.

`Snmpd` invokes `snmp_poll_request` to receive SNMP requests through `snmp_sock_recv`, which limits the size of the packet to 0x59C bytes (see 0x10012084 in Figure 36 and MTU values at Figure 78. Rx Configuration Index Table and Rx Configuration Table). The received packet is parsed by `snparse` and eventually transformed to an internal format by `sna2b`.

```
.text:10012044                                  .globl .snmp_poll_requests
.text:10012044                         .snmp_poll_requests:                    # CODE XREF: .snmpd+B4↓p
.text:10012044
.text:10012044                         .set sender_sp, -0x1848
.text:10012044                         .set var_1810, -0x1810
.text:10012044                         .set var_1800, -0x1800
.text:10012044                         .set var_1260, -0x1260
.text:10012044                         .set var_CC0, -0xCC0
.text:10012044                         .set var_CB0, -0xCB0
.text:10012044                         .set var_CA9, -0xCA9
.text:10012044                         .set var_CA8, -0xCA8
.text:10012044                         .set var_CA4, -0xCA4
.text:10012044                         .set var_CA0, -0xCA0
.text:10012044                         .set var_C9C, -0xC9C
.text:10012044                         .set var_C90, -0xC90
.text:10012044                         .set var_10, -0x10
.text:10012044                         .set var_C, -0xC
.text:10012044                         .set var_8, -8
.text:10012044                         .set var_4, -4
.text:10012044                         .set sender_lr,  8
.text:10012044
.text:10012044 7C 08 02 A6                      mflr      r0
.text:10012048 93 81 FF F0                      stw       r28, var_10(r1)
.text:1001204C 93 A1 FF F4                      stw       r29, var_C(r1)
.text:10012050 93 C1 FF F8                      stw       r30, var_8(r1)
.text:10012054 93 E1 FF FC                      stw       r31, var_4(r1)
.text:10012058 90 01 00 08                      stw       r0, sender_lr(r1)
.text:1001205C 94 21 E7 B8                      stwu      r1, sender_sp(r1)
.text:10012060 7C 7C 1B 78                      mr        r28, r3
.text:10012064 38 61 0B B8                      addi      r3, r1, 0x1848+var_C90
.text:10012068 90 61 0B AC                      stw       r3, 0x1848+var_C9C(r1)
.text:1001206C 38 80 00 14                      li        r4, 0x14
.text:10012070 4B FF FF 71                      bl        .link_bindings
.text:10012074 4F FF FB 82                      crmove    4*cr7+so, 4*cr7+so
.text:10012078 3B C1 00 48                      addi      r30, r1, 0x1848+var_1800
.text:1001207C 7F 83 E3 78                      mr        r3, r28
.text:10012080 7F C4 F3 78                      mr        r4, r30
.text:10012084 38 A0 05 9C                      li        r5, 0x59C
.text:10012088 38 C1 00 38                      addi      r6, r1, 0x1848+var_1810
.text:1001208C 48 00 05 DD                      bl        .snmp_sock_recv
.text:10012090 4F FF FB 82                      crmove    4*cr7+so, 4*cr7+so
.text:10012094 7C 7D 1B 79                      mr.       r29, r3
.text:10012098 40 81 01 98                      ble       loc_10012230
.text:1001209C 81 22 0C 70                      lwz       r9, in_packets_TC # _snmpd.bss_c
.text:100120A0 81 69 00 00                      lwz       r11, 0(r9)
.text:100120A4 39 6B 00 01                      addi      r11, r11, 1
.text:100120A8 91 69 00 00                      stw       r11, 0(r9)
.text:100120AC 3B E1 0B 88                      addi      r31, r1, 0x1848+var_CC0
.text:100120B0 7F E3 FB 78                      mr        r3, r31
.text:100120B4 7F C4 F3 78                      mr        r4, r30
.text:100120B8 7F A5 EB 78                      mr        r5, r29
.text:100120BC 48 00 07 89                      bl        .snparse
.text:100120C0 4F FF FB 82                      crmove    4*cr7+so, 4*cr7+so
.text:100120C4 2C 03 FF FF                      cmpwi     r3, -1
.text:100120C8 41 82 01 68                      beq       loc_10012230
.text:100120CC 7F E3 FB 78                      mr        r3, r31
.text:100120D0 48 00 0C 19                      bl        .sna2b
.text:100120D4 4F FF FB 82                      crmove    4*cr7+so, 4*cr7+so
.text:100120D8 2C 03 FF FF                      cmpwi     r3, -1
.text:100120DC 40 82 00 18                      bne       loc_100120F4
.text:100120E0 81 62 0C 74                      lwz       r11, asn_parse_error_TC # 0x2000F3D8
.text:100120E4 81 2B 00 00                      lwz       r9, 0(r11)
.text:100120E8 39 29 00 01                      addi      r9, r9, 1
.text:100120EC 91 2B 00 00                      stw       r9, 0(r11)
.text:100120F0 48 00 01 40                      b         loc_10012230
```

Figure 36. Vulnerable `hm_main` Code Flow

`snparse` successfully validates the initial structure of the received SNMP packet, eventually reaching the variable bindings part, where it fills a statically allocated doubly-linked list with pointers to the bindings, performing this operation until the entire packet is parsed. It is worth mentioning that the OID entries within this linked list are not parsed at that point. The number of nodes in the linked list is fixed to 20, each of them intended to hold a variable binding entry from the SNMP packet, as it is statically initialized in the stack by the `link_bindings` function.

`Sna2b` is in charge of transforming those entries into an internal structure. This structure, which is allocated in the stack, also holds additional structures, one of which is intended to hold the OID bytes into an array that has a fixed size of `32 * sizeof(short)` (0x40 bytes).

However, `sna2b` does not validate the length of the `ASN1_OBJID` element, which is returned by `a1readlen` (red basic block in Figure 37) before invoking `a1readoid`, thus passing this potentially malicious length as a parameter (see Figure 37).

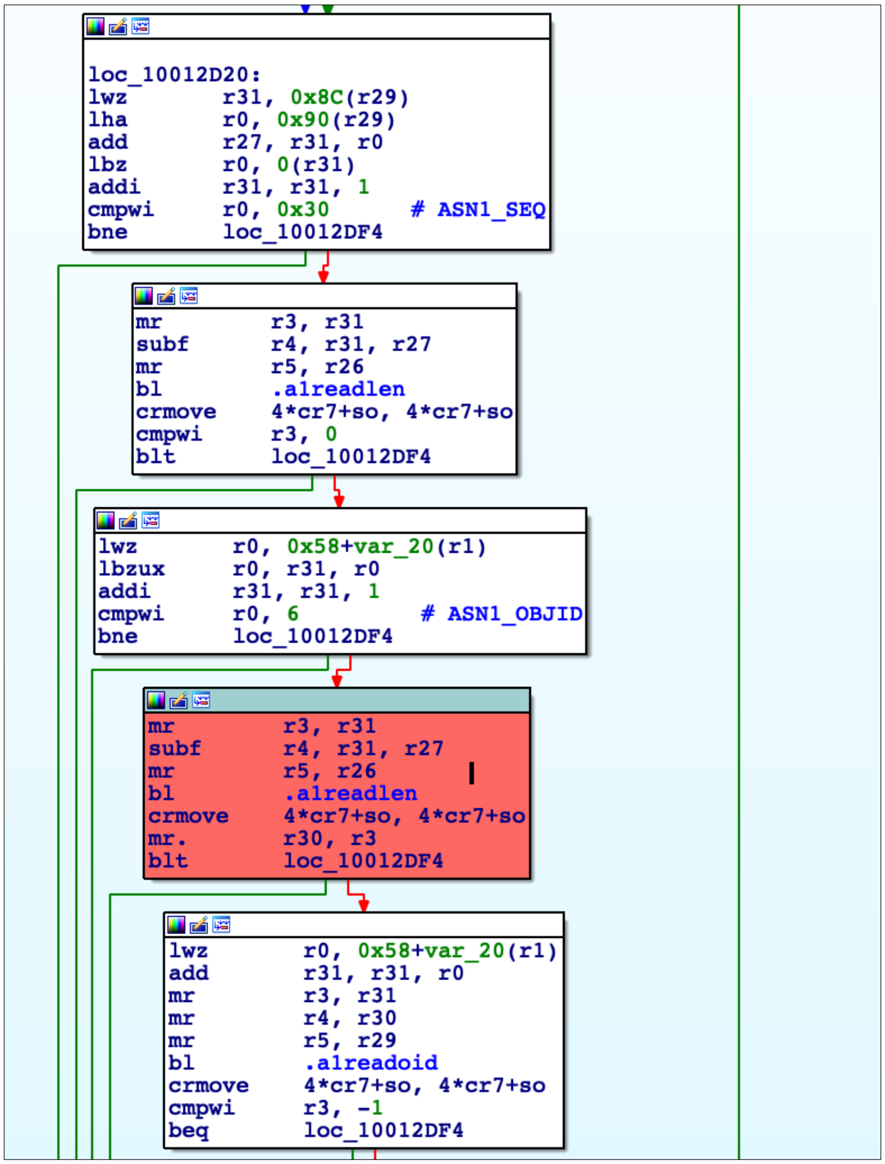


Figure 37. Code Flow with a1readlen

a1readoid then assumes it has to copy the OID bytes from the variable binding entry into the fixed OBJID array (0x40 bytes) until it reaches the potentially malicious length (yellow basic block in Figure 38). As this length is an attacker-controlled value, as a1readoid will corrupt the stack by writing controlled values (OID bytes, see Figure 40. Wireshark Dissection of Exploit Packet) out of the bounds of the aforementioned fixed OBJID array (red basic blocks in Figure 38), which can be then leveraged to execute arbitrary code.

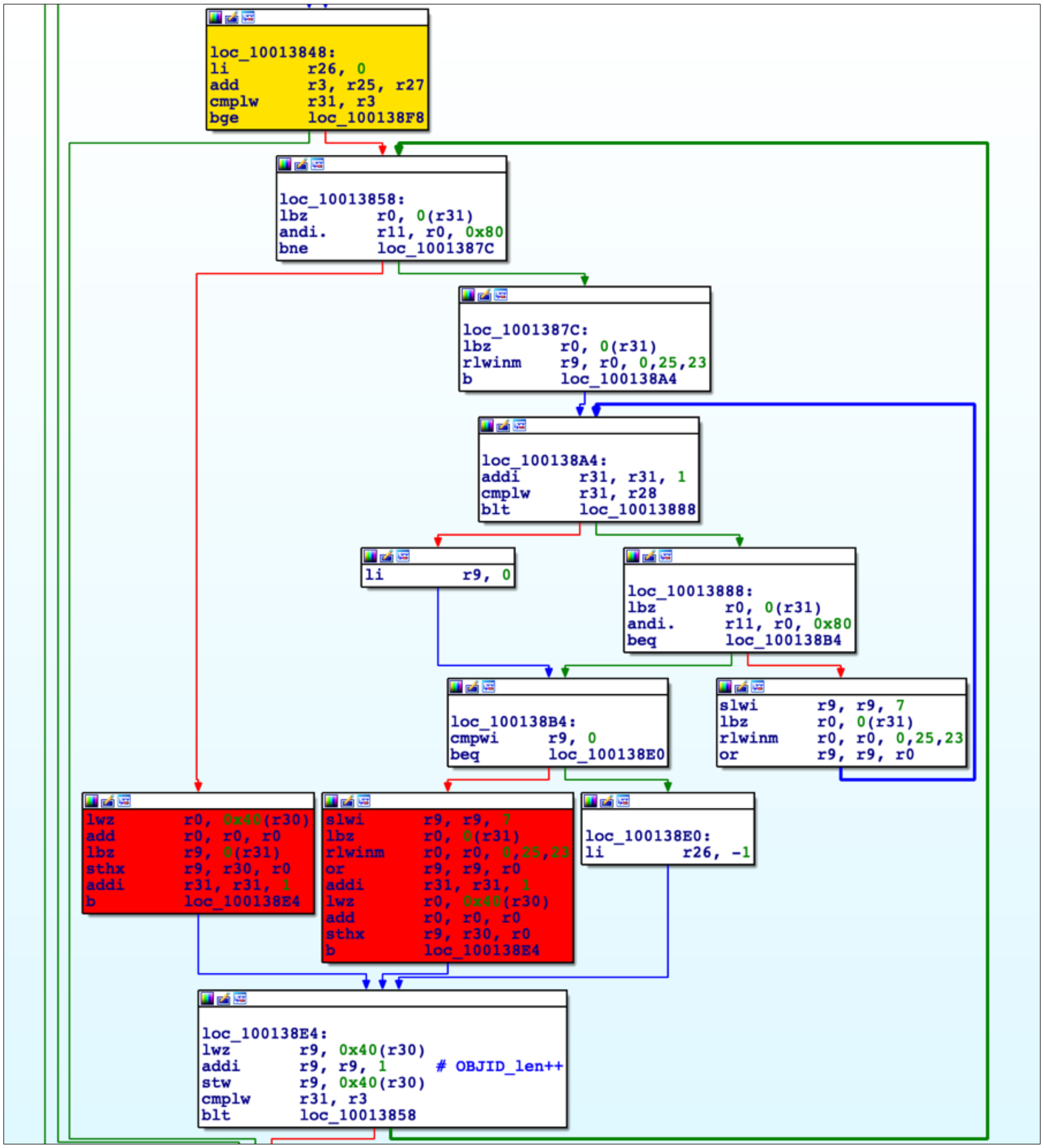


Figure 38. Vulnerable Code Flow

We can clearly show the underlying problem if we look at certain original parts from the 'TCP/IP illustrated v2' code in Figure 39. As `objidlen` is controlled, `a1readoid` will end up corrupting memory in the fixed `id` array within the `objid` structure. Although the code in the Pro Line fusion `snmpd` daemon is partially different, the original vulnerability was not spotted and survived the certification process.

```c
#define SMAXOBJID   32      /* max # of sub object ids  */
#define OBJSUBIDTYPE   unsigned short  /* type of sub object ids   */

struct oid {                    /* object identifier        */
    OBJSUBIDTYPE    id[SMAXOBJID];  /* array of sub-identifiers    */
    int     len;        /* length of this object id */
};


...


/*
 * Each snblist node contains an SNMP binding in one of 2 forms: ASN.1
 * encoded form or internal form.  The bindings list is doubly-linked
 */
struct snbentry {
    struct  oid sb_oid;     /* object id in internal form   */
    struct  snval sb_val;       /* value of the object      */
    u_char  *sb_a1str;      /* ASN.1 string containing the  */
                    /* object id and its value  */
    short   sb_a1slen;      /* length of the ASN.1 string   */
        Bool    sb_a1dynstr;        /* need a1str be freemem()ed?   */
    struct  snbentry *sb_next;  /* next node in the bind list   */
    struct  snbentry *sb_prev;  /* previous node in the list    */
};
```

```c
int
a1readoid(unsigned char *pack, int objidlen, struct oid *objid)
{
    int     val;
    u_char  *pp;

    objid->len = 0;
    pp = pack;

    /* verify the required 1.3.6.1.2.1 prefix */
    if (memcmp(MIB_PREFIX, pp, MIB_PREF_SZ)) {
        return SYSERR;
    }
    pp += MIB_PREF_SZ;

    for (; pp < pack + objidlen; objid->len++) {
        if (! (*pp & CHAR_HIBIT)) {
            objid->id[objid->len] = *pp++;
            continue;
        }
        /*
         * using long form, where bits 6 - 0 of each
         * octet are used; (bit 7 == 0) ==> last octet
         */
        val = 0;
        do
            val = (val << 7) | (int) (*pp & ~CHAR_HIBIT);
        while (*pp++ & CHAR_HIBIT);   /* high bit set */
        objid->id[objid->len] = val;
    }
    return OK;
}
```

Figure 39. TCP/IP Illustrated – Original Vulnerable Code

Exploitation

The exploit packet is limited to 0x59C bytes as it has been previously mentioned (see Figure 40). The stack space allocated for the linked list of bindings is 0xC90 bytes. Although there are several options to approach the exploit the most efficient is shown in the following image. It is worth mentioning that no compiler-level exploit mitigations were found.

Each of the nodes in this list is 0xA0 so in order to comply with all the requirements and still be able to corrupt the stack to gain code execution, the exploit will contain up to 20 bindings. The first 19 bindings will be regular ones, occupying the minimum number of bytes to be valid, so we can save space for the payload in the last one, as shown in the image below.

```
    Source Port: 20233
    Destination Port: 161
    Length: 516
  > Checksum: 0x5e7c [validation disabled]
    [Stream index: 0]
v Simple Network Management Protocol
    version: version-1 (0)
    community: public
  v data: get-request (0)
    v get-request
        request-id: 0
        error-status: noError (0)
        error-index: 0
      v variable-bindings: 20 items
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        > 1.3.6.1.2.1.1.5.0: Value (Null)
        v VarBind
            Object Name [truncated]: 1.3.6.1.2.1.1.5.0.5.0.65.65.65.65.65.65.65.65.65.65.65.65.65.65.65
v [Malformed Packet: SNMP]
```

Figure 40. Wireshark Dissection of Exploit Packet

Each of these bindings will be stored, after being parsed, in the corresponding linked list node. Finally, the last binding, for which the corresponding linked list node is closest to the Linkage Area, will be the one containing the malicious OID length. This will allow us to overwrite LR once `snmpd_poll_request` returns, thus gaining control over the execution (see Figure 41 and Figure 42).

Please note that a successful exploitation would allow to recover the process from the exploitation attempt. This is important in the context of avionics, as the exploit impact is essentially similar to an expected execution flow, thus preventing any underlying failure handling and error propagation mitigations mandatory for IMA systems.

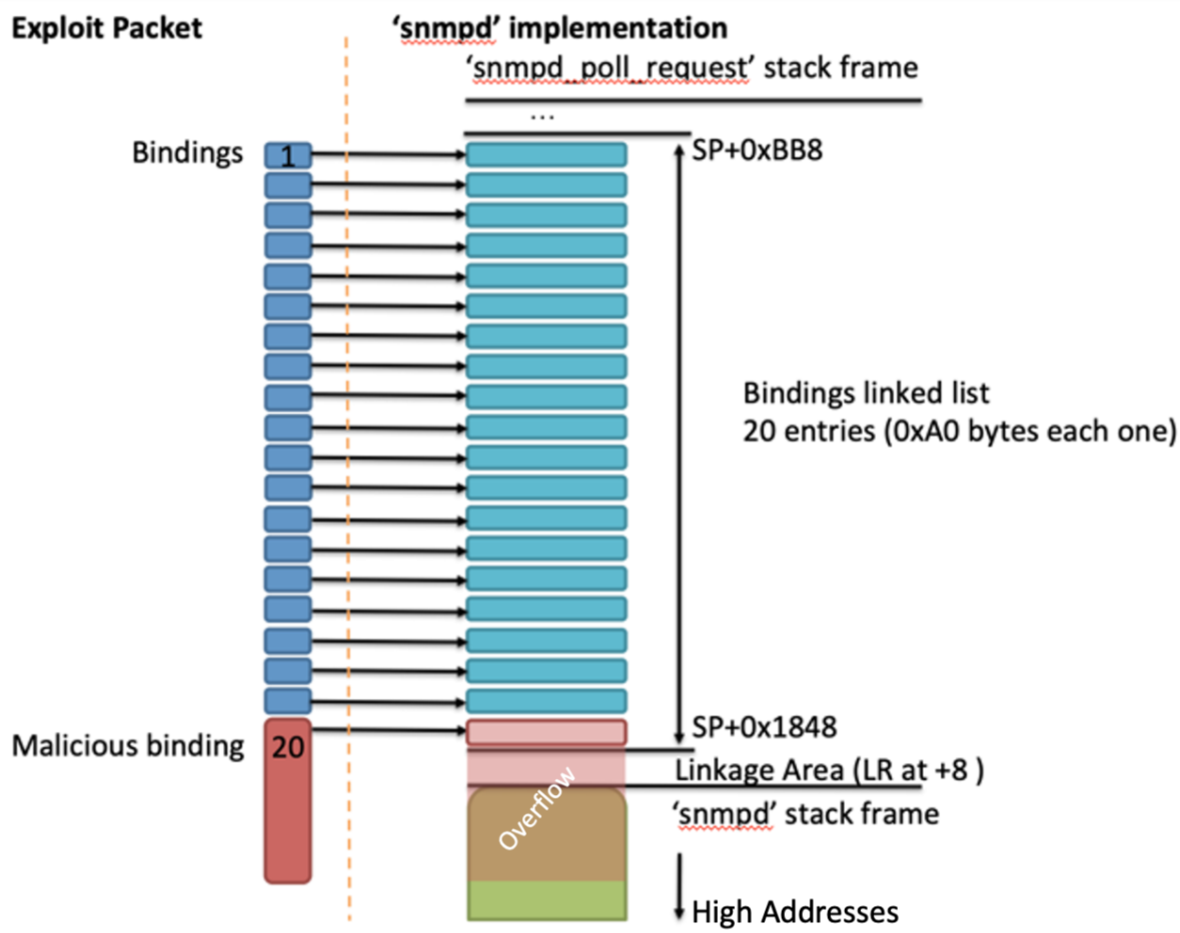


Figure 41. Exploit approach

```
0012230                                              # .snmp_poll_requests+84↑j ...
0012230 38 21 18 48              addi      r1, r1, 0x1848 # 0x1848 - 0xBB8 (start of bindings entries, 0x14 entries * 0xA0 size)
0012234 80 01 00 08              lwz       r0, sender_lr(r1)
0012238 7C 08 03 A6              mtlr      r0             # LR control
001223C 83 81 FF F0              lwz       r28, var_10(r1)
0012240 83 A1 FF F4              lwz       r29, var_C(r1)
0012244 83 C1 FF F8              lwz       r30, var_8(r1)
0012248 83 E1 FF FC              lwz       r31, var_4(r1)
001224C 4E 80 00 20              blr
001224C                  # End of function .snmp_poll_requests
```

Figure 42 Gaining code execution via LR control

Although `snmpd` has been demonstrated to be vulnerable, there is still some work to do in order to verify whether it matches our requirements for remote exploitation during all phases of the flight. The first step was to analyze the conditions under which `snmpd` is launched.

The AFDR-3700's `hm_main` contains logic to handle up to six different system modes shown in Figure 43 ('Normal', 'Dataload', 'IBIT', 'InvalidStrap', 'SwValidate', and 'InvalidConfig'). Obviously, we are interested in any code that is executed under 'Normal' (id 0x11) system mode, which is the regular operational mode for the AFD-3700 DUs.

```
.data:20003804 20 00 38 74        off_20003804:    .long aNormal            # DATA XREF: .data:off_2000387C↓o
.data:20003804                                                              # "Normal"
.data:20003808 00 00 00 11                         .long 0x11
.data:2000380C 20 00 38 68                         .long aDataload          # "Dataload"
.data:20003810 00 00 00 06                         .long 6
.data:20003814 20 00 38 60                         .long aIbit              # "IBIT"
.data:20003818 00 00 00 0C                         .long 0xC
.data:2000381C 20 00 38 50                         .long aInvalidstrap      # "InvalidStrap"
.data:20003820 00 00 00 05                         .long 5
.data:20003824 20 00 38 44                         .long aSwvalidate_0      # "SwValidate"
.data:20003828 00 00 00 0A                         .long 0xA
.data:2000382C 20 00 38 34                         .long aInvalidconfi_0    # "InvalidConfig"
.data:20003830 00 00 00 09                         .long 9
```

Figure 43. System modes

Each supported system mode has a table of associated threads that should be created. `init_threads_for_mode` receives the current boot mode and proceeds to launch the required threads:

```
.text:10001174 48 00 A2 8D        bl      .hm_get_sys_mode
.text:10001178 4F FF FB 82        crmove  4*cr7+so, 4*cr7+so
.text:1000117C 7C 7C 1B 78        mr      r28, r3
```

```
.text:100011FC                    loc_100011FC:                    # CODE XREF: .main+150↑j
.text:100011FC                                                     # .main+168↑j
.text:100011FC 7F 83 E3 78            mr      r3, r28
.text:10001200 7F E4 FB 78            mr      r4, r31
.text:10001204 48 00 AB 61            bl      .hm_mode_initialization
.text:10001208 4F FF FB 82            crmove  4*cr7+so, 4*cr7+so
.text:1000120C 7F 83 E3 78            mr      r3, r28
.text:10001210 7F E4 FB 78            mr      r4, r31
.text:10001214 7F 65 DB 78            mr      r5, r27
.text:10001218 48 00 48 BD            bl      .init_threads_for_mode
```

Figure 44. `init_threads_for_mode`

For the Normal system mode, we have the following threads:

```
.text:1003BA74 00 00 00 11        .long 0x11       #  Boot Mode  - Normal id 0x11
.text:1003BA78 00 00 00 00        .long 0          # 20hz thread
.text:1003BA7C 00 00 00 01        .long 1          # 1hz thread
.text:1003BA80 00 00 00 02        .long 2          # Lifecycle thread
.text:1003BA84 00 00 00 03        .long 3          # RAM test thread
.text:1003BA88 00 00 00 05        .long 5          # Error data thread
.text:1003BA8C 00 00 00 09        .long 9          # Dataload detect thread
.text:1003BA90 00 00 00 07        .long 7          # CIO thread
.text:1003BA94 00 00 00 06        .long 6          # *** snmpd thread ***
.text:1003BA98 00 00 00 08        .long 8          # Processor Sync Thread
.text:1003BA9C 00 00 00 0A        .long 0xA        # end marker
```

Figure 45. Normal System Mode Threads

Thread ID 6 corresponds to the `snmpd` thread:

```
.data:20000BBC 00 00 00 96        .long 0x96
.data:20000BC0 00 00 00 06        .long 6                        # thread id
.data:20000BC4 00 00 00 56        .long 0x56
.data:20000BC8 00 00 00 04        .long 4
.data:20000BCC 20 00 A4 60        .long snmpd
.data:20000BD0 20 00 0A 98        .long _hmthreadcontrol.rw_c_0
.data:20000BD4 20 00 CC 68        .long unk_2000CC68
.data:20000BD8 00 00 00 01        .long 1
.data:20000BDC 00 00 00 00        .long 0
.data:20000BE0 00 00 00 00        .long 0
.data:20000BE4 00 00 00 00        .long 0
.data:20000BE8 20 00 0C D8        .long aSnmpCycleSlip           # "SNMP cycle slip"
```

Figure 46. Thread Structure

`init_threads_for_mode` dereferences the corresponding thread table for the current system mode, initializes the list of active threads, and creates them.

```
.text:10005CB4 4F FF FB 82        crmove   4*cr7+so, 4*cr7+so
.text:10005CB8 57 9C F8 7E        srwi     r28, r28, 1   # r28 = boot mode = 0x11
.text:10005CBC 2C 9C 00 08        cmpwi    cr1, r28, 8
.text:10005CC0 7C 00 00 26        mfcr     r0
.text:10005CC4 54 00 2F FE        extrwi   r0, r0, 1,4
.text:10005CC8 7C 00 00 D0        neg      r0, r0
.text:10005CCC 7F 89 00 38        and      r9, r28, r0
.text:10005CD0 7C 00 00 F8        not      r0, r0
.text:10005CD4 54 00 07 7E        clrlwi   r0, r0, 29
.text:10005CD8 7D 3C 03 78        or       r28, r9, r0
.text:10005CDC 7F 79 DB 78        mr       r25, r27
.text:10005CE0 81 22 03 44        lwz      r9, threads_per_mode_TC # _hmthreadcontrol.rw_c
.text:10005CE4 57 80 08 3C        slwi     r0, r28, 1
.text:10005CE8 7C 00 E2 14        add      r0, r0, r28
.text:10005CEC 54 00 10 3A        slwi     r0, r0, 2
.text:10005CF0 7C 1C 00 50        subf     r0, r28, r0
.text:10005CF4 54 0A 10 3A        slwi     r10, r0, 2
.text:10005CF8 39 29 00 04        addi     r9, r9, 4
.text:10005CFC 7C 09 50 2E        lwzx     r0, r9, r10   # dereference thread table for normal boot mode
```

Figure 47. Dereferencing thread table

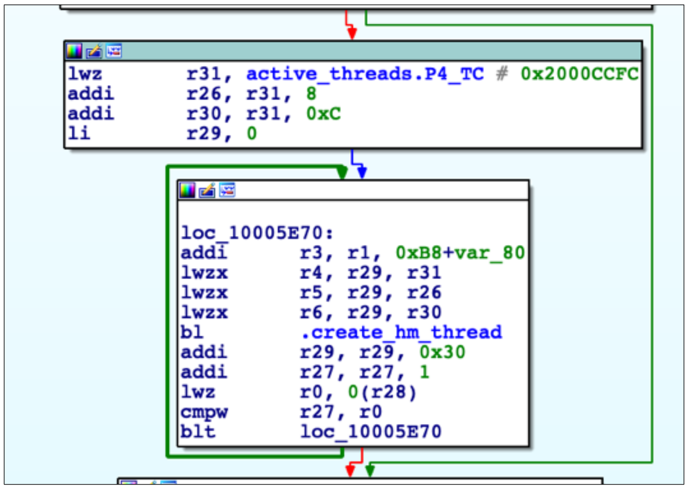


Figure 48. Creating Thread

At this point, we have just confirmed that the `hm_main` application running under regular conditions (Normal system mode) launches the vulnerable `snmpd` daemon.

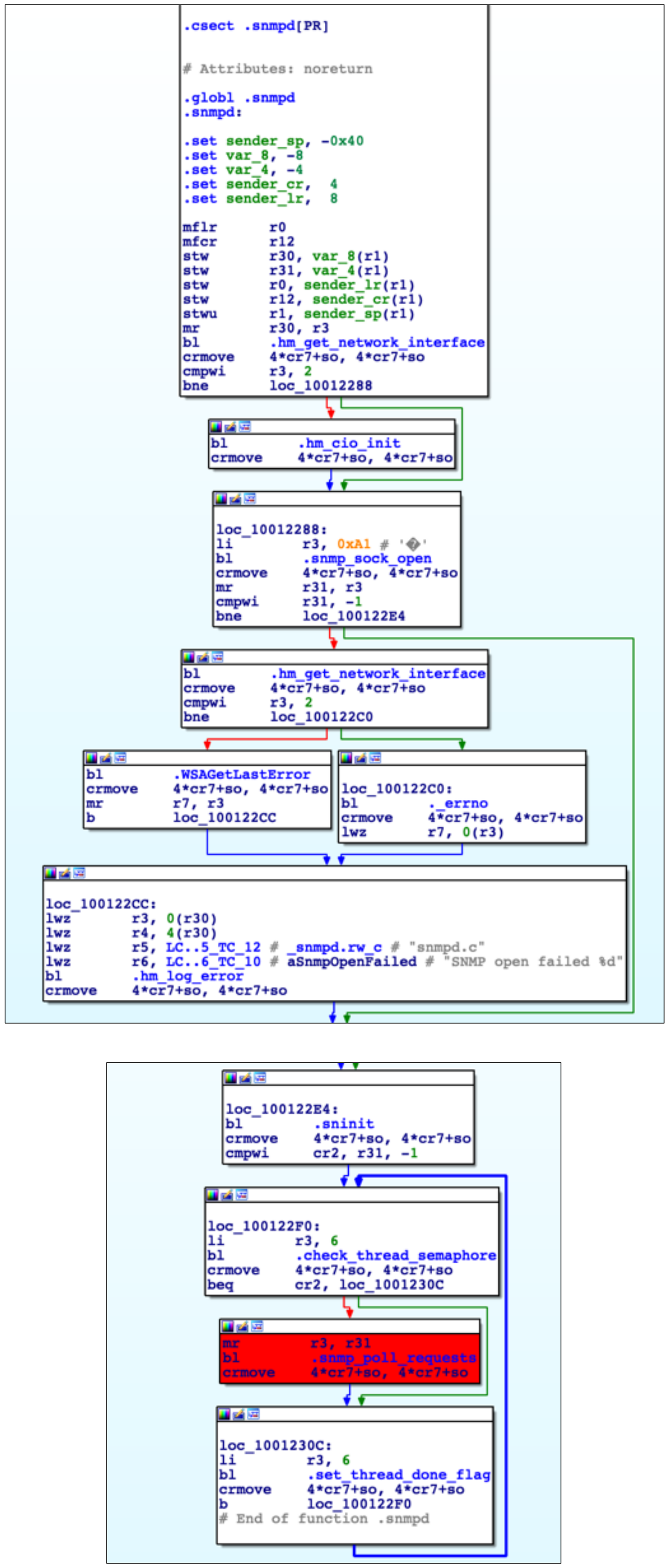


Figure 49. `snmpd` *Code*

As shown in Figure 49, there is no check for either a discrete or a specific condition before reaching the starting point for our vulnerability, which is the red basic block (`snmp_poll_requests`); however, there is still a verification step we have to perform, as we do not yet know how sockets are handled in the AFD-3700.

AFD-3700 Inter-Partition Communication Mechanisms and Network Connectivity

The `snmpd` thread code described above shows a socket API logic that seems pretty similar to the one implemented in Microsoft Windows systems, even using the same function names, such as `WSAGetLastError`, or error codes.

If we pay attention to the VCT file (see Figure 50), we will also find that at line 27 the `NetworkInterface` parameter is `Winsock2.2`, which may initially be surprising.

```
27  NetworkInterface=Winsock2.2;                                 // VCT173
28  ColdStartSchedule=;                                          // VCT174
29  ColdStartDuration=0;                                         // VCT175
30  RunTimeSchedule=0[1] 1[3] 2[7] 3[12] 5[2] 0[2] 1[2] 2[8] 3[11] 5[2];    // VCT178
31
32  //////////////////////////////
33  // Virtual Machine Table
34  //////////////////////////////
35
36  <VM0>                                                        // VCT78
37  GroupIds=;                                                   // VCT1187
38  LogicalName=AFDR-3700;                                       // VCT1188
39  CommandLine=/usr/bin/app_launcher;                           // VCT1189
40  EnvironmentVars=HealthMonitorIndex=255
41          Field_Load_Id_List = AFDR:IMAT:ICIT:RTSA:EICAS:ECDA:OMSA:ODLA:OMST:ECL-DB:OMSTAR
42          PctPathFname=/usr/etc/vm0.pct;                       // VCT1190
```

Figure 50. S1-SL03 VCT File

The explanation behind this move seems to be found in the paper “Commercially available, DO-178B level a certifiable, hard partitioned, posix compliant real-time operating system and TCP/UDP compliant ethernet stack software”[41] published by LynxWorks and Rockwell Collins in 2003. This publication provides an interesting glimpse into the requirements of those Collins avionics products relying on LynxOS-178.

[41] https://ur.booksc.eu/book/31018525/f88b3c

POSIX / Winsock

LynxOS-178 is a POSIX-compatible operating system based on the LynxOS RTOS. LynxOS was strategically subsetted to retain key functionality while minimizing the amount of development code that goes through the expensive process of Level A verification. In this case, strategically subsetting refers to comparing POSIX functionality with avionics requirements in order to determine what POSIX functionality is not required. With the POSIX compatibility, a development environment, subsetted in the same way, can be set up on a workstation. Because a goal of the POSIX standard is to maximize source code reuse across different platforms, applications can initially be developed and tested on a development platform prior to transitioning to a target platform. Additionally, POSIX-aware developers will be able to jump right into developing on the LynxOS-178 system with minimal additional training, reducing project startup costs.

Lynx Certifiable Stack (LCS) uses a strategically subsetted version of the WinSock2 API and an appropriate subset of TCP/UDP/IP RFCs to allow applications to communicate with other applications over a network. In this case, strategic subsetting refers to implementing most, but not all of the functions within the WinSock2 API. Some features of the API, such as indefinite blocking, do not exist within LCS because DO-178B guidance requires the components to act in a deterministic manner. Since indefinite blocking violates this requirement, a configured blocking timeout is used in its place.

Deterministic Hard Partitioned Design

LynxOS-178

Systems in an avionics environment, must behave in a deterministic manner. This means that each component within the system must be analyzable and worst-case bounded. Many factors can affect the deterministic nature of a component. When dealing with an operating system running on an LRU, it quickly becomes evident that to maintain determinism of individual components, those components must be isolated from each other in a manner that ensures determinism. The LynxOS-178 method of achieving this is to use hard, or brick-wall partitioning as shown in Figure 2 below.

Modularity

Because of the WinSock2 interface provided by the LCS package, a standardized interface exists for component interconnectivity. New components can be added to the system based on this published, standardized interface, simplifying the process of third-party vendors developing compatible components and reducing overall system upgrade costs.

Figure 51. Extracted from LynxWorks and Rockwell Collins Avionics Paper[42]

As it is required to assess the feasibility of the discovered vulnerabilities, the underlying stack logic has been fully reverse engineered to completely understand and characterize the configured communication flows between partitions as well as those coming from the Avionics System LAN.

We now briefly introduce the components involved, then we will fully elaborate their functionalities and interactions based on the network configuration.

[42] https://ur.booksc.eu/book/31018525/f88b3c

- AFDX ASL driver (`afdx_asl_drv.obj`): Implements the vast majority of the logic behind the inter-partition communication mechanism and the AFDX network capabilities.
- PCIE driver (`pcie.dldd`): Implements the End-System part, providing the low-level layers to enable the AFD-3700 DUs to communicate with the Avionics System LAN.
- `network.cfg`: Proprietary binary file; contains the complete configuration `AFD-AFDX_asl_driver.obj` and `PCIE.dldd` rely on to allow/deny communication flows between the different partitions and with other components in the Avionics System LAN.

Figure 52 provides a detailed overview of the architecture.

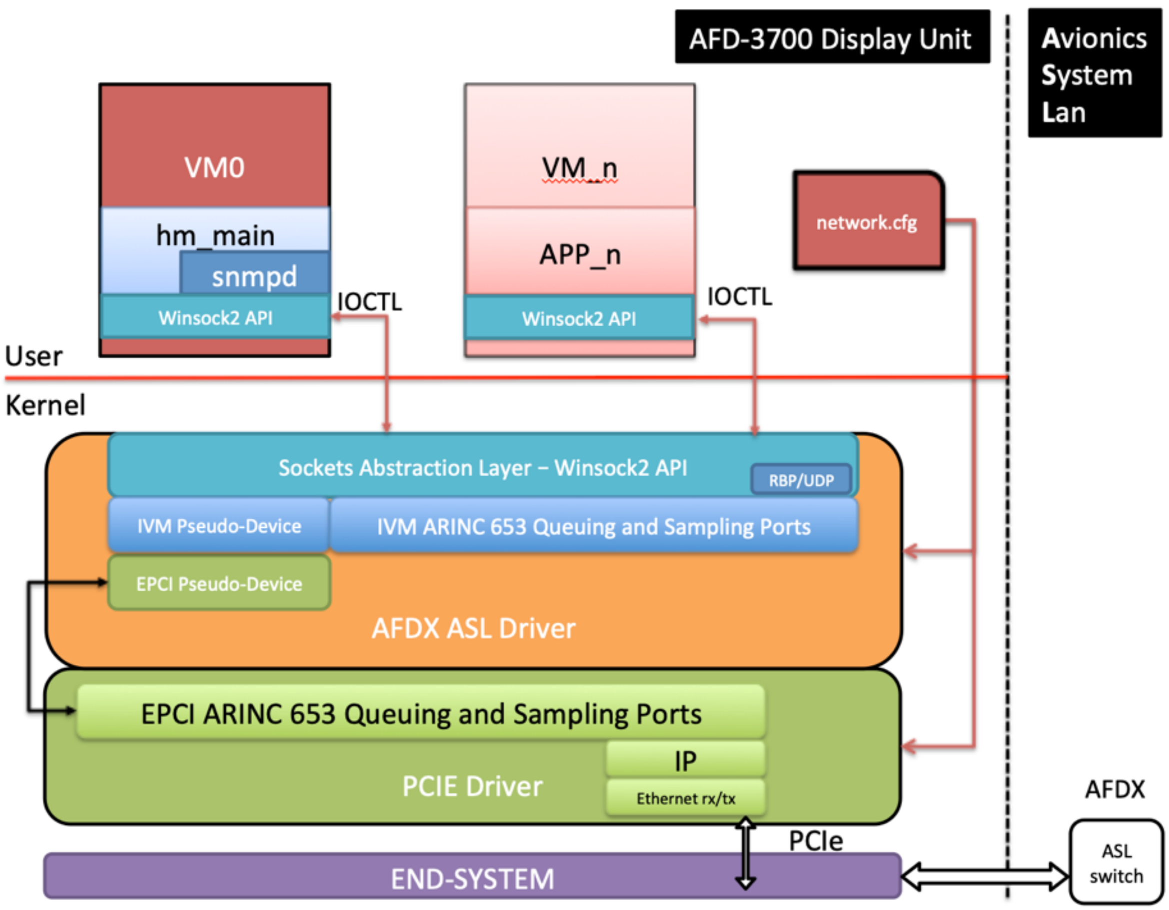


Figure 52. Network and Inter-Partition Communication Architecture

network.cfg Analysis

This file could be parsed based on the reverse engineered logic found in the AFDX and PCIE drivers. This configuration file provides the deterministic rules to be implemented in the ASL.

At boot, when the AFDX driver's `install` entry point is invoked (see Figure 53), it looks for certain information from the mapped INFO file (`/usr/etc/afdx_asl_info_0`) which, for example, includes whether it has to perform some verifications or the path to the network configuration file (`network.cfg`). It proceeds to load, parse, and generate the configuration tables that will be used at runtime.

```
.text:00002F34 7C 08 02 A6                 mflr      r0
.text:00002F38 93 81 FF F0                 stw       r28, var_10(r1)
.text:00002F3C 93 A1 FF F4                 stw       r29, var_C(r1)
.text:00002F40 93 C1 FF F8                 stw       r30, var_8(r1)
.text:00002F44 93 E1 FF FC                 stw       r31, var_4(r1)
.text:00002F48 90 01 00 08                 stw       r0, sender_lr(r1)
.text:00002F4C 94 21 FF B0                 stwu      r1, sender_sp(r1)
.text:00002F50 7C 7C 1B 78                 mr        r28, r3        # mapped INFO file
.text:00002F54 7C 9E 23 78                 mr        r30, r4
.text:00002F58 81 22 01 B0                 lwz       r9, verification_enabled_TC # verification_enabled
.text:00002F5C 88 1C 00 3F                 lbz       r0, 0x3F(r28) # verification enabled in INFO? (false in production)
.text:00002F60 90 09 00 00                 stw       r0, 0(r9)
.text:00002F64 80 62 01 B4                 lwz       r3, verify_info_TC # verify_info
.text:00002F68 38 80 00 0C                 li        r4, 0xC
.text:00002F6C 48 02 40 ED                 bl        .bzero
.text:00002F70 80 41 00 14                 lwz       r2, 0x50+saved_toc(r1)
.text:00002F74 83 E2 01 B4                 lwz       r31, verify_info_TC # verify_info
.text:00002F78 38 00 00 00                 li        r0, 0
.text:00002F7C 90 1F 00 04                 stw       r0, 4(r31)
.text:00002F80 90 1F 00 08                 stw       r0, 8(r31)
.text:00002F84 81 22 01 A8                 lwz       r9, curr_vm_TC # curr_vm
.text:00002F88 80 62 01 B8                 lwz       r3, main_TC # off_27920
.text:00002F8C 38 80 20 00                 li        r4, 0x2000
.text:00002F90 80 A9 00 00                 lwz       r5, 0(r9)
.text:00002F94 38 C0 01 F4                 li        r6, 0x1F4
.text:00002F98 80 E2 01 BC                 lwz       r7, LC..114_TC # aAfdxaslverific # "AfdxAslVerificationThread"
.text:00002F9C 39 00 00 01                 li        r8, 1
.text:00002FA0 39 20 00 00                 li        r9, 0
.text:00002FA4 48 00 2E 41                 bl        .afdx_asl_vmos_ststart
.text:00002FA8 4F FF FB 82                 crmove    4*cr7+so, 4*cr7+so
.text:00002FAC 90 7F 00 00                 stw       r3, 0(r31)
.text:00002FB0 2C 03 FF FF                 cmpwi     r3, -1
.text:00002FB4 40 82 00 0C                 bne       loc_2FC0
.text:00002FB8 38 60 FF FF                 li        r3, -1
.text:00002FBC 48 00 01 6C                 b         loc_3128
.text:00002FC0                 # ---------------------------------------------------------------------------
.text:00002FC0
.text:00002FC0                 loc_2FC0:                              # CODE XREF: .afdx_aslinstall+80↑j
.text:00002FC0 38 7F 00 08                 addi      r3, r31, 8
.text:00002FC4 38 80 FF FF                 li        r4, -1
.text:00002FC8 48 00 46 41                 bl        .afdx_asl_swait
.text:00002FCC 4F FF FB 82                 crmove    4*cr7+so, 4*cr7+so
.text:00002FD0 81 22 01 C0                 lwz       r9, asl_debug_state_TC # asl_debug_state
.text:00002FD4 80 1C 00 2C                 lwz       r0, 0x2C(r28)
.text:00002FD8 90 09 00 00                 stw       r0, 0(r9)
.text:00002FDC 81 22 01 34                 lwz       r9, skip_tests_TC # skip_tests
.text:00002FE0 80 1C 00 34                 lwz       r0, 0x34(r28)
.text:00002FE4 90 09 00 00                 stw       r0, 0(r9)
.text:00002FE8 38 60 00 18                 li        r3, 0x18
.text:00002FEC 48 00 44 C1                 bl        .afdx_asl_sysbrk
.text:00002FF0 4F FF FB 82                 crmove    4*cr7+so, 4*cr7+so
.text:00002FF4 7C 7F 1B 78                 mr        r31, r3
.text:00002FF8 38 80 00 18                 li        r4, 0x18
.text:00002FFC 48 02 40 5D                 bl        .bzero
.text:00003000 80 41 00 14                 lwz       r2, 0x50+saved_toc(r1)
.text:00003004 88 1C 00 3B                 lbz       r0, 0x3B(r28)
.text:00003008 98 1F 00 10                 stb       r0, 0x10(r31)
.text:0000300C 88 1C 00 3C                 lbz       r0, 0x3C(r28)
.text:00003010 98 1F 00 11                 stb       r0, 0x11(r31)
.text:00003014 88 1C 00 3A                 lbz       r0, 0x3A(r28)
.text:00003018 98 1F 00 12                 stb       r0, 0x12(r31)
.text:0000301C 80 1C 00 E4                 lwz       r0, 0xE4(r28)
.text:00003020 90 1F 00 14                 stw       r0, 0x14(r31)
.text:00003024 83 A2 01 0C                 lwz       r29, pAfdxAslStatics_TC # pAfdxAslStatics
.text:00003028 93 FD 00 00                 stw       r31, 0(r29)
.text:0000302C 38 7C 00 40                 addi      r3, r28, 0x40 # path to network.cfg ('/usr/local/etc/network.cfg')
.text:00003030 7F C4 F3 78                 mr        r4, r30
.text:00003034 48 00 0A 09                 bl        .LoadConfigTables
```

Figure 53. AFDX ASL Driver - `install` Entry Point

The first function related to the network configuration is `LoadConfigTables` that parses a set of initial table records found in the `network.cfg` file, looking for the `normal_table` record (identified by the 0xFFFF marker, see Figure 54).

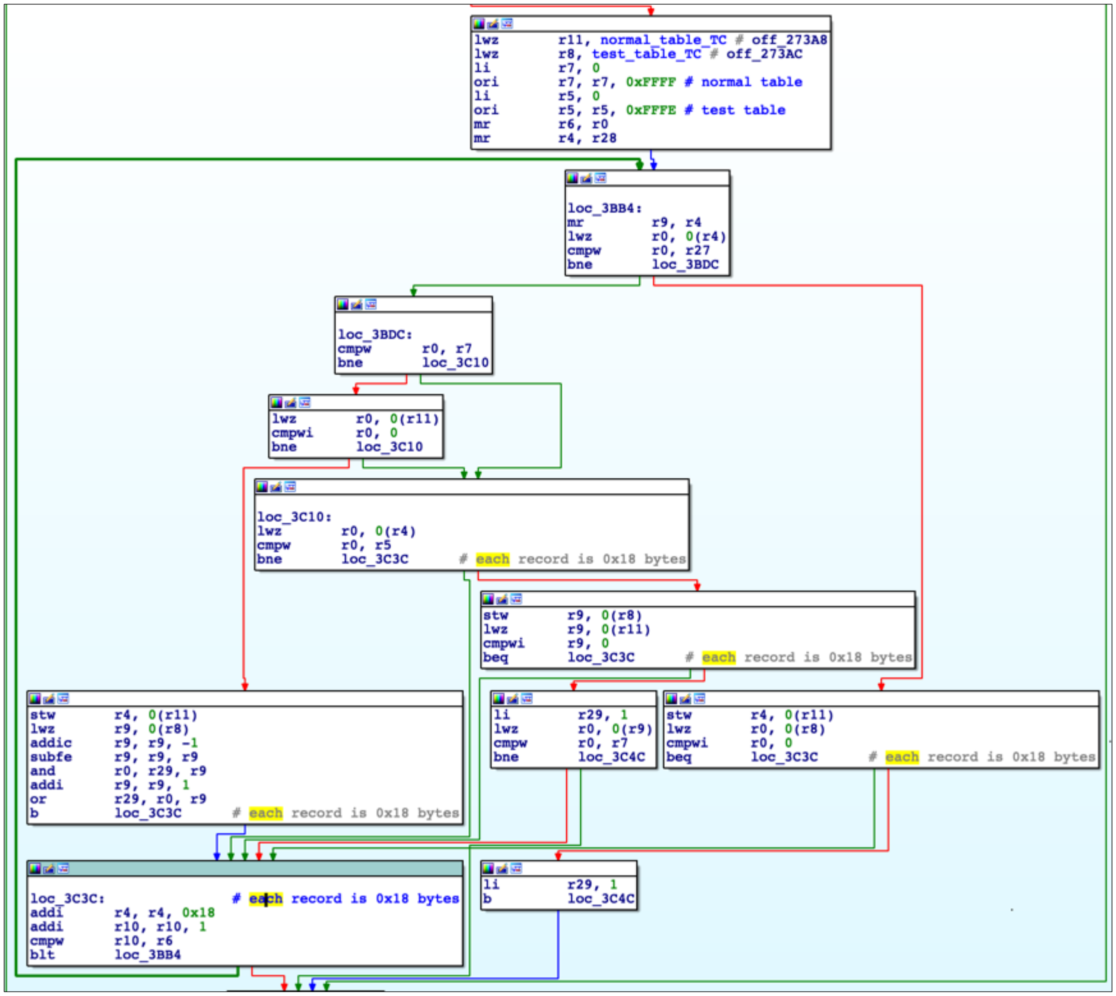


Figure 54. Code Searching for `normal_table` Record

Once the normal table has been found, a `normal_features` configuration SubEntry is allocated based on the normal table's offset to the `normal_feature` entry in `network.cfg`.

```
.text:00003C4C             loc_3C4C:                        # CODE XREF: .LoadConfigTables+154↑j
.text:00003C4C                                              # .LoadConfigTables+19C↑j ...
.text:00003C4C 2C 1D 00 01          cmpwi   r29, 1
.text:00003C50 40 82 00 C0          bne     loc_3D10
.text:00003C54 83 E2 01 E8          lwz     r31, normal_table_TC # off_273A8
.text:00003C58 81 3F 00 00          lwz     r9, 0(r31)
.text:00003C5C 7F C3 F3 78          mr      r3, r30
.text:00003C60 80 89 00 08          lwz     r4, 8(r9)
.text:00003C64 80 A9 00 04          lwz     r5, 4(r9)
.text:00003C68 80 C9 00 0C          lwz     r6, 0xC(r9)
.text:00003C6C 80 E2 01 EC          lwz     r7, normal_features_TC # dword_273B8
.text:00003C70 4B FF FC 29          bl      .AllocateConfigSubEntry
```

Figure 55. `normal_features` SubEntry

The driver then tries to find the `WSA_V0` SubEntry from the previously allocated entries.

```
.text:00003038 4F FF FB 82                 crmove    4*cr7+so, 4*cr7+so
.text:0000303C 2C 03 00 01                 cmpwi     r3, 1
.text:00003040 40 82 00 E0                 bne       loc_3120
.text:00003044 38 60 00 01                 li        r3, 1
.text:00003048 80 82 01 C4                 lwz       r4, LC..121_TC # aWsa_v0 # "WSA_V0"
.text:0000304C 38 A1 00 38                 addi      r5, r1, 0x38
.text:00003050 48 00 06 E9                 bl        .GetConfigSubEntryInfo
```

Figure 56. Searching for `WSA_V0`

The information contained into these entries provides `LoadAslConfig` with a pointer to `CnfgTblOffsets`, which contains offsets to the different configuration tables and its number of entries, as you can in Figure 57.

```
01098  0006417C 000017B8 5753415F 56300000 00000000 00000000 00000000 00000000 00000000
010BC  00000000 00064154 000017B8 000000D8 00000010 00000158 00000010 000001D8 00000012
010E0  00000418 00000002 00000420 00000000 00000420 00000000 00000420 00000010 000004A0
01104  00000018 000007A0 00000000 000007A0 00000000 000007A0 00000010 000007E0 0000000F
01128  00000DF8 00000010 00000E38 00000010 000013B8 00000010 000018B8 00000000 000018B8
0114C  00000000 000018B8 00000000 000018B8 00000031 00001A40 00000003 00001A4C 00000010
01170  00001A8C 00000007 00001B88 00000002 00001BC8 00000000 00001BC8 00000000 00001BC8
01194  00000000 00001BC8 00000000 00290000 00000000 00020000 00000000 00000000 00000000
011B8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Figure 57. `CnfgTblOffsets`

```
text:0000B6D4                         .globl .LoadAslConfig
text:0000B6D4             .LoadAslConfig:                        # CODE XREF: .afdx_aslinstall+17C↑p
text:0000B6D4                                                    # DATA XREF: .data:off_284F0↓o
text:0000B6D4
text:0000B6D4             .set sender_sp, -0x58
text:0000B6D4             .set var_20, -0x20
text:0000B6D4             .set var_1C, -0x1C
text:0000B6D4             .set var_18, -0x18
text:0000B6D4             .set var_14, -0x14
text:0000B6D4             .set var_10, -0x10
text:0000B6D4             .set var_C, -0xC
text:0000B6D4             .set var_8, -8
text:0000B6D4             .set var_4, -4
text:0000B6D4             .set sender_lr,  8
text:0000B6D4
text:0000B6D4 7C 08 02 A6             mflr     r0
text:0000B6D8 93 81 FF F0             stw      r28, var_10(r1)
text:0000B6DC 93 A1 FF F4             stw      r29, var_C(r1)
text:0000B6E0 93 C1 FF F8             stw      r30, var_8(r1)
text:0000B6E4 93 E1 FF FC             stw      r31, var_4(r1)
text:0000B6E8 90 01 00 08             stw      r0, sender_lr(r1)
text:0000B6EC 94 21 FF A8             stwu     r1, sender_sp(r1)
text:0000B6F0 3B E0 00 01             li       r31, 1
text:0000B6F4 83 C2 01 54             lwz      r30, CnfgTblOffsets_TC # CnfgTblOffsets
text:0000B6F8 90 9E 00 00             stw      r4, 0(r30)
text:0000B6FC 83 82 01 AC             lwz      r28, SckAllocCnfg_TC # SckAllocCnfg
text:0000B700 80 04 00 00             lwz      r0, 0(r4)
text:0000B704 7C 04 02 14             add      r0, r4, r0
text:0000B708 90 1C 00 00             stw      r0, 0(r28)
text:0000B70C 81 22 03 6C             lwz      r9, RxCnfgIndexTbl_TC # RxCnfgIndexTbl
text:0000B710 80 04 00 08             lwz      r0, 8(r4)
text:0000B714 7C 04 02 14             add      r0, r4, r0
text:0000B718 90 09 00 00             stw      r0, 0(r9)
text:0000B71C 81 22 03 54             lwz      r9, RxCnfgTbl_TC # RxCnfgTbl
text:0000B720 80 04 00 10             lwz      r0, 0x10(r4)
text:0000B724 7C 04 02 14             add      r0, r4, r0
text:0000B728 90 09 00 00             stw      r0, 0(r9)
text:0000B72C 81 22 03 F0             lwz      r9, McBufferCnfgTbl_TC # McBufferCnfgTbl
text:0000B730 80 04 00 18             lwz      r0, 0x18(r4)
text:0000B734 7C 04 02 14             add      r0, r4, r0
text:0000B738 90 09 00 00             stw      r0, 0(r9)
text:0000B73C 81 22 03 F4             lwz      r9, RxcRbpCnfgTbl_TC_0 # RxcRbpCnfgTbl
text:0000B740 80 04 00 20             lwz      r0, 0x20(r4)
text:0000B744 7C 04 02 14             add      r0, r4, r0
text:0000B748 90 09 00 00             stw      r0, 0(r9)
text:0000B74C 81 22 03 A8             lwz      r9, RxcComPortCnfgTbl_TC # RxcComPortCnfgTbl
text:0000B750 80 04 00 28             lwz      r0, 0x28(r4)
text:0000B754 7C 04 02 14             add      r0, r4, r0
text:0000B758 90 09 00 00             stw      r0, 0(r9)
text:0000B75C 81 22 03 70             lwz      r9, TxCnfgIndexTbl_TC # TxCnfgIndexTbl
text:0000B760 80 04 00 30             lwz      r0, 0x30(r4)
text:0000B764 7C 04 02 14             add      r0, r4, r0
text:0000B768 90 09 00 00             stw      r0, 0(r9)
text:0000B76C 81 22 03 64             lwz      r9, TxCnfgTbl_TC # TxCnfgTbl # 8
text:0000B770 80 04 00 38             lwz      r0, 0x38(r4)
text:0000B774 7C 04 02 14             add      r0, r4, r0
text:0000B778 90 09 00 00             stw      r0, 0(r9)
text:0000B77C 81 22 03 F8             lwz      r9, TxcRbpCnfgTbl_TC # TxcRbpCnfgTbl
text:0000B780 80 04 00 40             lwz      r0, 0x40(r4)
text:0000B784 7C 04 02 14             add      r0, r4, r0
text:0000B788 90 09 00 00             stw      r0, 0(r9)
text:0000B78C 81 22 03 AC             lwz      r9, TxcComPortCnfgTbl_TC # TxcComPortCnfgTbl
text:0000B790 80 04 00 48             lwz      r0, 0x48(r4)
text:0000B794 7C 04 02 14             add      r0, r4, r0
text:0000B798 90 09 00 00             stw      r0, 0(r9)
text:0000B79C 81 22 03 FC             lwz      r9, HostNameCnfgIndexTbl_TC # HostNameCnfgIndexTbl
text:0000B7A0 80 04 00 50             lwz      r0, 0x50(r4)
```

Figure 58. AFDX ASL Driver - `LoadAslConfig` Function

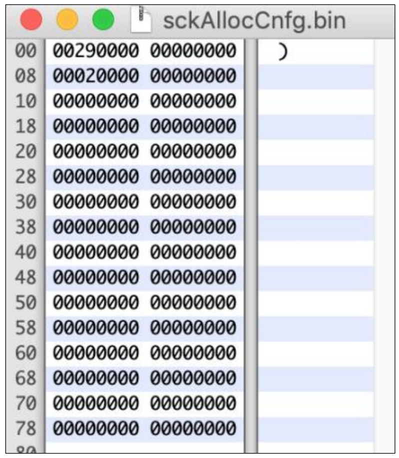


Figure 59. `sckAllocCnfg.bin`

Based on this information, we can see in Figure 57 that the first entry, which corresponds to the `SckAllocCnfg` table (see Figure 59), is at offset 0xD8 (starting at the `CnfgTblOffsets` offset) and it contains 0x10 entries of 8 bytes, one for each supported VM. The table itself contains the number of sockets a VM is allowed to allocate.

Following this logic, it was possible to identify the tables involved.

Table 3. Identified tables

Table Name	Offset	Description	Enabled
`SckAllocCfng`	0xD8	Number of allowed sockets	TRUE
`RxCnfgIndexTbl`	0x158	A VM-based index of configured Rx entries in RxCnfgTbl	TRUE
`RxCnfgTbl`	0x1D8	Incoming Sockets allowed	TRUE
`McBufferCnfgTbl`	0x418	Multicast Buffer Config	TRUE
`RxcRbpCnfgTbl`	0x420		FALSE
`RxcComPortCnfgTbl`	0x420		FALSE
`TxCnfgIndexTbl`	0x420	A VM-based index of configured Tx entries in TxCnfgTbl	TRUE
`TxCnfgTbl`	0x4a0	Outgoing Sockets allowed	TRUE
`TxcRbpCnfgTbl`	0x7A0		FALSE
`TxcComPortCnfgTbl`	0x7A0		FALSE
`HostNameCnfgIndexTbl`	0x7A0	A VM-based index of configured Hostname entries in HostNameCnfgTbl	TRUE
`HostNameCnfgTbl`	0x7E0	IP And Hostname of expected hosts.	TRUE

Table Name	Offset	Description	Enabled
`PortNameCnfgIndexTbl`	0xDF8	A VM-based index of configured port name entries in PortNameCnfgTbl	TRUE
`PortNameCnfgTbl`	0xE38	Port number and Name of the configured sockets	TRUE
`HostCnfgTbl`	0x13b8	Default hostnames for each of the supported VM	TRUE
`EdeLocalPtr`	0x18B8		FALSE
`EdeRemotePtr`	0x18B8		FALSE
`DCACnfgTbl`	0x18B8		FALSE
`_653PortCnfgTbl`	0x18B8	List of the id for the configured ARINC653 Q/S ports	TRUE
`IvmCnfgTbl`	0x1A40		TRUE
`_653PortNameCnfgIndexTbl`	0x1A4C	A VM-based index of configured ARINC653 Q/S port name entries in 653PortNameCnfg	TRUE
`_653PortNameCnfgTb`	0x1a8c	Name, id and VM associated with the configured A653 Q/S ports.	TRUE
`DeviceNameCnfgTbl`	0x1B88	Name of the supported AFDX/PCIE pseudo-devices	TRUE
`AggregatePortCnfgTbl`	0x1BC8		FALSE
`PogoeGeneralPtrlPtr`	0x1BC8		FALSE
`PogoeChannelPtr`	0x1BC8		FALSE
`StreamRBPCnfgTbl`	0x1BC8		FALSE

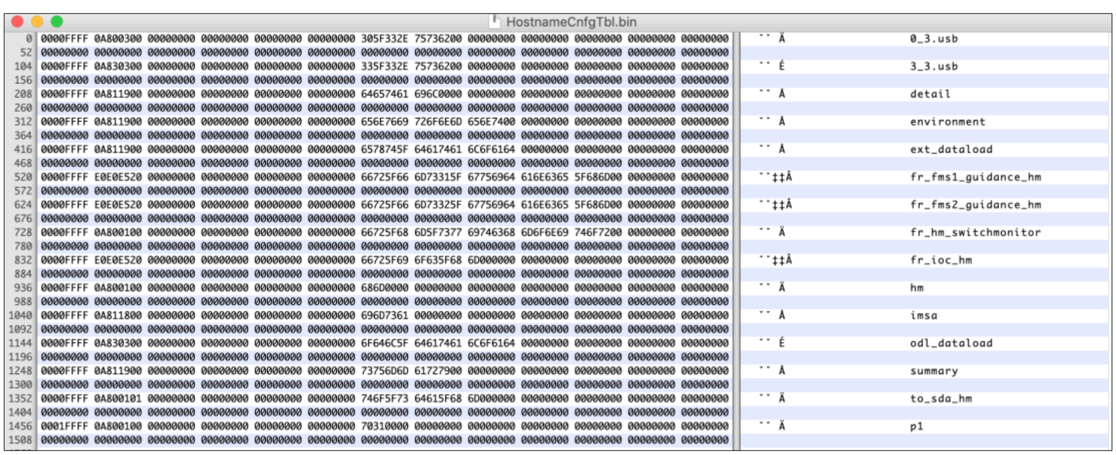


Figure 60. Hostname Table

For the `PCIE.dldd` driver, the approach was much the same.

```
.text:00002FE8                  .globl .load_config
.text:00002FE8 .load_config:                        # CODE XREF: .pcie_driver_install+80↓p
.text:00002FE8                                      # DATA XREF: .data:load_config↓o
.text:00002FE8
.text:00002FE8 .set sender_sp, -0x48
.text:00002FE8 .set var_10, -0x10
.text:00002FE8 .set var_C, -0xC
.text:00002FE8 .set var_8, -8
.text:00002FE8 .set sender_lr,  8
.text:00002FE8
.text:00002FE8                  mflr     r0
.text:00002FEC                  stw      r0, sender_lr(r1)
.text:00002FF0                  stwu     r1, sender_sp(r1)
.text:00002FF4                  lwz      r0, in_ephemeral_tbl_count_TC # 0xD8A0
.text:00002FF8                  stw      r0, 0x38(r1)
.text:00002FFC                  lwz      r0, in_emac_table_TC # off_BEB0
.text:00003000                  stw      r0, 0x3C(r1)
.text:00003004                  lwz      r0, in_emac_tbl_count_TC # 0xD8A4
.text:00003008                  stw      r0, 0x40(r1)
.text:0000300C                  lwz      r4, in_tx_table_TC # _config.rw_c
.text:00003010                  lwz      r5, in_tx_tbl_count_TC # _config.bss_c
.text:00003014                  lwz      r6, in_eth_table_TC # off_BEA8
.text:00003018                  lwz      r7, in_eth_tbl_count_TC # 0xD89C
.text:0000301C                  lwz      r8, in_rx_table_TC # dword_BEA4
.text:00003020                  lwz      r9, in_rx_tbl_count_TC # 0xD894
.text:00003024                  lwz      r10, in_ephemeral_table_TC # off_BEAC
.text:00003028                  bl       .extract_config_data
```

Figure 61. PCIE Driver - `load_config` Function

The configured tables for the PCIE driver are the following:

- `in_tx_table`
- `in_tx_tbl_count`
- `in_eth_table`
- `in_eth_table_count`
- `in_rx_table`
- `in_rx_tbl_count`
- `in_ephemeral_table`

These tables contain expected tuples of IPs and ports involved in the ASL communications the End-System expects to see.

Having this information, we now proceed to trace a socket communication to figure out whether we can claim remote/inter-partition attacks against the `snmpd` are possible.

Following the Packets

As the previous architecture diagram showed, the entire Socket Abstraction Layer is implemented over the AFDX's IOCTL interface. In this way, user-mode applications can directly talk to the AFDX driver to request operations and receive data.

The entire communication process is transparent for user-mode applications, no matter whether they are looking to communicate with another VM or a remote device through the ASL.

Through the use of 'AFDX logical devices,' the AFDX and PCIE drivers implement the logic that handles the socket requests depending on the source and destination of the participants.

```
.text:000030B0 48 00 86 25          bl      .LoadAslConfig
.text:000030B4 4F FF FB 82          crmove  4*cr7+so, 4*cr7+so
.text:000030B8 7C 7E 1B 78          mr      r30, r3
.text:000030BC 2C 1E 00 01          cmpwi   r30, 1
.text:000030C0 40 82 00 4C          bne     loc_310C
.text:000030C4 80 02 01 CC          lwz     r0, afdx_device_reg_fntab_TC # afdx_device_reg_fntab
.text:000030C8 90 01 00 3C          stw     r0, 0x50+var_14(r1)
.text:000030CC 80 62 01 D0          lwz     r3, LC..126_TC # aAfdx_device_re # "AFDX_DEVICE_REG_FNTAB"
.text:000030D0 38 80 00 01          li      r4, 1
.text:000030D4 38 A1 00 3C          addi    r5, r1, 0x50+var_14
.text:000030D8 48 00 3A 6D          bl      .afdx_asl_ksetenv
.text:000030DC 4F FF FB 82          crmove  4*cr7+so, 4*cr7+so
.text:000030E0 2C 03 00 00          cmpwi   r3, 0
.text:000030E4 40 82 00 3C          bne     loc_3120
.text:000030E8 4B FF DF 1D          bl      .register_IVM # Register IVM Pseudo-Device to handle Inter-VM communications
.text:000030EC 7F 83 E3 78          mr      r3, r28
.text:000030F0 7F E4 FB 78          mr      r4, r31
.text:000030F4 4B FF D0 4D          bl      .register_ES  # not supported in King-Air configuration
.text:000030F8 4B FF DE 55          bl      .register_Aggregate # not supported in King-Air configuration
.text:000030FC 81 22 01 D4          lwz     r9, driver_installed_TC # driver_installed
.text:00003100 93 C9 00 00          stw     r30, 0(r9)
```

Figure 62. AFDX Driver Code

As seen in Figure 62, it first registers a kernel 'environment variable' that contains the required function pointers to register an AFDX logical device.

These function pointers are the following:

Table 4. `AFDX_DEVICE_REG_FNTAB`

Offset	Value
0	NULL
4	`afdx_device_register`
8	`unregister_device`
0xC	`enable_device`
0x10	`disable_device`
0x14	`get_device_config`
0x18	`get_device_test_config`

It proceeds to call `register_IVM`, `register_ES` (see Figure 66), and `register_Aggregate`; however, a logical device will only be successfully registered and enabled when it is present in the `DeviceNameCnfgTbl`.

```
ext:00004950                    .globl .register_device
ext:00004950          .register_device:                      # CODE XREF: .register_ES+188↑p
ext:00004950                                                 # .register_ES+1A4↑p ...
ext:00004950
ext:00004950          .set sender_sp, -0x58
ext:00004950          .set var_1C, -0x1C
ext:00004950          .set var_18, -0x18
ext:00004950          .set var_14, -0x14
ext:00004950          .set var_10, -0x10
ext:00004950          .set var_C, -0xC
ext:00004950          .set var_8, -8
ext:00004950          .set var_4, -4
ext:00004950          .set sender_lr,  8
ext:00004950
ext:00004950 7C 08 02 A6        mflr    r0
ext:00004954 93 21 FF E4        stw     r25, var_1C(r1)
ext:00004958 93 41 FF E8        stw     r26, var_18(r1)
ext:0000495C 93 61 FF EC        stw     r27, var_14(r1)
ext:00004960 93 81 FF F0        stw     r28, var_10(r1)
ext:00004964 93 A1 FF F4        stw     r29, var_C(r1)
ext:00004968 93 C1 FF F8        stw     r30, var_8(r1)
ext:0000496C 93 E1 FF FC        stw     r31, var_4(r1)
ext:00004970 90 01 00 08        stw     r0, sender_lr(r1)
ext:00004974 94 21 FF A8        stwu    r1, sender_sp(r1)
ext:00004978 7C 7B 1B 78        mr      r27, r3
ext:0000497C 7C 9A 23 78        mr      r26, r4
ext:00004980 7C BD 2B 78        mr      r29, r5
ext:00004984 7C D9 33 78        mr      r25, r6
ext:00004988 4B FF FE C9        bl      .getDeviceIndex
```

Figure 63. PCIE Driver - `register_device` Function

```
text:00004850
text:00004850                   .globl .getDeviceIndex
text:00004850         .getDeviceIndex:                  # CODE XREF: .register_device+38↓p
text:00004850                                           # DATA XREF: .data:off_27F70↓o
text:00004850
text:00004850         .set sender_sp, -0x50
text:00004850         .set var_18, -0x18
text:00004850         .set var_14, -0x14
text:00004850         .set var_10, -0x10
text:00004850         .set var_C, -0xC
text:00004850         .set var_8, -8
text:00004850         .set var_4, -4
text:00004850         .set sender_lr,  8
text:00004850
text:00004850 7C 08 02 A6       mflr    r0
text:00004854 93 41 FF E8       stw     r26, var_18(r1)
text:00004858 93 61 FF EC       stw     r27, var_14(r1)
text:0000485C 93 81 FF F0       stw     r28, var_10(r1)
text:00004860 93 A1 FF F4       stw     r29, var_C(r1)
text:00004864 93 C1 FF F8       stw     r30, var_8(r1)
text:00004868 93 E1 FF FC       stw     r31, var_4(r1)
text:0000486C 90 01 00 08       stw     r0, sender_lr(r1)
text:00004870 94 21 FF B0       stwu    r1, sender_sp(r1)
text:00004874 7C 7A 1B 78       mr      r26, r3
text:00004878 3B 80 00 00       li      r28, 0
text:0000487C 63 9C FF FF       ori     r28, r28, 0xFFFF
text:00004880 3B A0 00 00       li      r29, 0
text:00004884 81 22 01 54       lwz     r9, CnfgTblOffsets_TC # CnfgTblOffsets
text:00004888 81 29 00 00       lwz     r9, 0(r9)
text:0000488C A3 C9 00 B6       lhz     r30, 0xB6(r9) # Number of entries in the DeviceNameCnfgTbl
text:00004890 2C 1E 00 00       cmpwi   r30, 0
text:00004894 41 82 00 90       beq     loc_4924
text:00004898 7C 1D F0 10       subfc   r0, r29, r30
text:0000489C 38 00 00 00       li      r0, 0
text:000048A0 7C 00 01 14       adde    r0, r0, r0
text:000048A4 6B C9 FF FF       xori    r9, r30, 0xFFFF
text:000048A8 7D 29 00 D0       neg     r9, r9
text:000048AC 55 29 0F FE       srwi    r9, r9, 31
text:000048B0 7C 0B 48 39       and.    r11, r0, r9
text:000048B4 41 82 00 70       beq     loc_4924
text:000048B8 83 62 02 60       lwz     r27, DeviceNameCnfgTbl_TC # DeviceNameCnfgTbl
text:000048BC
text:000048BC         loc_48BC:                         # CODE XREF: .getDeviceIndex+D0↓j
text:000048BC 7C 1D F2 14       add     r0, r29, r30  # iterates the DeviceNameConfigTbl
text:000048C0 54 1F F8 7E       srwi    r31, r0, 1
text:000048C4 57 E0 28 34       slwi    r0, r31, 5
text:000048C8 80 9B 00 00       lwz     r4, 0(r27)
text:000048CC 7F 43 D3 78       mr      r3, r26
text:000048D0 7C 84 02 14       add     r4, r4, r0
text:000048D4 48 01 0D 9D       bl      .internal_stricmp # Check Device Name
text:000048D8 4F FF FB 82       crmove  4*cr7+so, 4*cr7+so
```

Figure 64. PCIE Driver - `getDeviceIndex` Function

In our current configuration there are only two entries (logical devices) in `DeviceNameCnfgTbl`: ‘`EPCI`’ and ‘`IVM`’.

```
DeviceNameCnfgTbl.bin
 0 45504349 00000000 00000000 00000000  EPCI
16 00000000 00000000 00000000 00000000
32 49564D00 00000000 00000000 00000000  IVM
48 00000000 00000000 00000000 00000000
```

Figure 65. `DeviceNameCnfgTbl.bin`

Thus, `register_ES` and `register_Aggregate` will fail as they are trying to register ‘`ES_0`’ and ‘`POGOE_ES_0`’, which are not supported in the current configuration.

```
text:000002B0 80 41 00 14    lwz     r2, 0x2F8+saved_toc(r1)
text:000002B4 3B A1 00 D8    addi    r29, r1, 0xD8
text:000002B8 80 62 00 FC    lwz     r3, es_asic_string_TC # _afdxdrvr.rw_c_0 # "ES_0"
text:000002BC 38 81 01 28    addi    r4, r1, 0x128
text:000002C0 7F A5 EB 78    mr      r5, r29
text:000002C4 80 DC 00 00    lwz     r6, 0(r28)
text:000002C8 48 00 46 89    bl      .register_device
text:000002CC 4F FF FB 82    crmove  4*cr7+so, 4*cr7+so
text:000002D0 7C 7E 1B 78    mr      r30, r3
text:000002D4 80 62 01 04    lwz     r3, pogoe_string_TC # aPogoe_es_0 # "POGOE_ES_0"
text:000002D8 38 81 00 F0    addi    r4, r1, 0xF0
text:000002DC 7F A5 EB 78    mr      r5, r29
text:000002E0 80 DC 00 00    lwz     r6, 0(r28)
text:000002E4 48 00 46 6D    bl      .register_device
```

Figure 66. `register_ES`

```
ext:00000FAC 83 E2 01 74    lwz     r31, aggregate_device_index_TC # aggregate_device_index
ext:00000FB0 80 62 01 78    lwz     r3, aggregate_name_TC # aAggregate # "AGGREGATE"
ext:00000FB4 38 81 00 38    addi    r4, r1, 0x38
ext:00000FB8 38 A0 00 00    li      r5, 0
ext:00000FBC 38 C0 00 00    li      r6, 0
ext:00000FC0 38 E0 00 00    li      r7, 0
ext:00000FC4 48 00 3D 85    bl      .afdx_device_register
```

Figure 67. `register_Aggregate`

On the other hand, as ‘`IVM`‘ is present in the `DeviceNameCnfgTbl` configuration, `register_IVM` (see Figure 68) will be able to register its logical device, which implements the ARINC653 Queuing/Sampling ports for inter-VM communication.

```
text:00001004
text:00001004                 .register_IVM:                              # CODE XREF: .afdx_aslinstall+1B4↓p
text:00001004                                                             # DATA XREF: .data:off_279C0↓o
text:00001004
text:00001004                 .set sender_sp, -0x78
text:00001004                 .set var_40, -0x40
text:00001004                 .set var_3C, -0x3C
text:00001004                 .set var_38, -0x38
text:00001004                 .set var_34, -0x34
text:00001004                 .set var_30, -0x30
text:00001004                 .set var_2C, -0x2C
text:00001004                 .set var_28, -0x28
text:00001004                 .set var_24, -0x24
text:00001004                 .set var_20, -0x20
text:00001004                 .set var_1C, -0x1C
text:00001004                 .set var_18, -0x18
text:00001004                 .set var_14, -0x14
text:00001004                 .set var_10, -0x10
text:00001004                 .set var_C, -0xC
text:00001004                 .set var_4, -4
text:00001004                 .set sender_lr,  8
text:00001004
text:00001004 7C 08 02 A6          mflr    r0
text:00001008 93 E1 FF FC          stw     r31, var_4(r1)
text:0000100C 90 01 00 08          stw     r0, sender_lr(r1)
text:00001010 94 21 FF 88          stwu    r1, sender_sp(r1)
text:00001014 80 02 01 7C          lwz     r0, Create_QueuingPort_Ivm_TC # off_285E0
text:00001018 90 01 00 38          stw     r0, 0x38(r1)
text:0000101C 80 02 01 80          lwz     r0, Send_Queuing_Ivm_TC # off_28600
text:00001020 90 01 00 3C          stw     r0, 0x3C(r1)
text:00001024 80 02 01 84          lwz     r0, Receive_Queuing_Ivm_TC # off_28620
text:00001028 90 01 00 40          stw     r0, 0x40(r1)
text:0000102C 80 02 01 88          lwz     r0, GetPortStatus_Queuing_Ivm_TC # off_28650
text:00001030 90 01 00 44          stw     r0, 0x44(r1)
text:00001034 80 02 01 8C          lwz     r0, Purge_QueuingPort_Ivm_TC # off_285F0
text:00001038 90 01 00 48          stw     r0, 0x48(r1)
text:0000103C 80 02 01 90          lwz     r0, Create_SamplingPort_Ivm_TC # off_285D0
text:00001040 90 01 00 4C          stw     r0, 0x4C(r1)
text:00001044 80 02 01 94          lwz     r0, Write_Sampling_Ivm_TC # off_28610
text:00001048 90 01 00 50          stw     r0, 0x50(r1)
text:0000104C 80 02 01 98          lwz     r0, Read_Sampling_Ivm_TC # off_28630
text:00001050 90 01 00 54          stw     r0, 0x54(r1)
text:00001054 80 02 01 9C          lwz     r0, GetPortStatus_Sampling_Ivm_TC # off_28640
text:00001058 90 01 00 58          stw     r0, 0x58(r1)
text:0000105C 80 02 00 C4          lwz     r0, CreateStreamingPort_RBP_TC # off_283F0
text:00001060 90 01 00 5C          stw     r0, 0x5C(r1)
text:00001064 80 02 00 C8          lwz     r0, SendStreamingData_RBP_TC # off_28410
text:00001068 90 01 00 60          stw     r0, 0x60(r1)
text:0000106C 80 02 00 CC          lwz     r0, ReceiveStreamingData_RBP_TC # off_28400
text:00001070 90 01 00 64          stw     r0, 0x64(r1)
text:00001074 80 02 00 D0          lwz     r0, GetStreamingPortStatus_RBP_TC # off_28420
text:00001078 90 01 00 68          stw     r0, 0x68(r1)
text:0000107C 80 02 00 D4          lwz     r0, ResetStreamingPort_RBP_TC # off_28430
text:00001080 90 01 00 6C          stw     r0, 0x6C(r1)
text:00001084 83 E2 01 A0          lwz     r31, ivm_device_index_TC # ivm_device_index
text:00001088 80 62 01 A4          lwz     r3, ivm_string_TC # aIvm # "IVM"
text:0000108C 38 81 00 38          addi    r4, r1, 0x38
text:00001090 38 A0 00 00          li      r5, 0
text:00001094 38 C0 00 00          li      r6, 0
text:00001098 48 00 38 B9          bl      .register_device
text:0000109C 4F FF FB 82          crmove  4*cr7+so, 4*cr7+so
text:000010A0 90 7F 00 00          stw     r3, 0(r31)
text:000010A4 38 00 00 00          li      r0, 0
text:000010A8 60 00 FF FF          ori     r0, r0, 0xFFFF
text:000010AC 7C 03 00 00          cmpw    r3, r0
text:000010B0 41 82 00 14          beq     loc_10C4
text:000010B4 A0 7F 00 02          lhz     r3, 2(r31)
text:000010B8 38 80 00 00          li      r4, 0
text:000010BC 48 00 3B 2D          bl      .enable_device
text:000010C0 4F FF FB 82          crmove  4*cr7+so, 4*cr7+so
```

Figure 68. `register_IVM`[43]

[43] Reliable Burst Protocol (RBP) is a proprietary protocol developed by Rockwell Collins with similarities to TCP. There is almost no public information on RBP. The AFDR-3700 supports this protocol. https://ieeexplore.ieee.org/document/5655316

The PCIE driver operates in the same way to register its 'EPCI' device. It gets the AFDX_DEVICE_REG_FNTAB pointer and proceeds to register the device with the required functions to handle those ARINC653 Queuing/Sampling ports that require communication over the AFDX network (ASL).

```
.text:0000AB5C                 .globl .afdx_device_install
.text:0000AB5C .afdx_device_install:                  # CODE XREF: .pcie_install+1D4↑p
.text:0000AB5C                                        # DATA XREF: .data:afdx_device_install↓o
.text:0000AB5C
.text:0000AB5C .set sender_sp, -0x48
.text:0000AB5C .set saved_toc, -0x34
.text:0000AB5C .set var_10, -0x10
.text:0000AB5C .set var_C, -0xC
.text:0000AB5C .set var_8, -8
.text:0000AB5C .set var_4, -4
.text:0000AB5C .set sender_lr,  8
.text:0000AB5C
.text:0000AB5C                 mflr      r0
.text:0000AB60                 stw       r28, var_10(r1)
.text:0000AB64                 stw       r29, var_C(r1)
.text:0000AB68                 stw       r30, var_8(r1)
.text:0000AB6C                 stw       r31, var_4(r1)
.text:0000AB70                 stw       r0, sender_lr(r1)
.text:0000AB74                 stwu      r1, sender_sp(r1)
.text:0000AB78                 mr        r29, r3
.text:0000AB7C                 lwz       r3, LC..2_TC # aAfdx_device_re # "AFDX_DEVICE_REG_FNTAB"
.text:0000AB80                 lwz       r4, device_reg_fntab_TC # _afdxentrypoints.rw_c
.text:0000AB84                 li        r5, 4
.text:0000AB88                 bl        .kgetenv
.text:0000AB8C                 lwz       r2, 0x48+saved_toc(r1)
.text:0000AB90                 cmpwi     r3, 0
.text:0000AB94                 bne       loc_ACCC
.text:0000AB98                 lwz       r28, device_reg_fntab_TC # _afdxentrypoints.rw_c
.text:0000AB9C                 lwz       r11, 0(r28)
.text:0000ABA0                 lwz       r9, 0(r11)
.text:0000ABA4                 cmpwi     r9, 0
.text:0000ABA8                 bne       loc_ACCC
.text:0000ABAC                 lwz       r4, device_fntab_TC # _afdxentrypoints.bss_c
.text:0000ABB0                 lwz       r0, pcie_create_port_TC # pcie_create_port
.text:0000ABB4                 stw       r0, 0(r4)
.text:0000ABB8                 lwz       r0, pcie_send_message_TC # pcie_send_message
.text:0000ABBC                 stw       r0, 4(r4)
.text:0000ABC0                 lwz       r0, pcie_recv_message_TC # pcie_recv_message
.text:0000ABC4                 stw       r0, 8(r4)
.text:0000ABC8                 lwz       r0, pcie_get_status_TC # pcie_get_status
.text:0000ABCC                 stw       r0, 0xC(r4)
.text:0000ABD0                 stw       r9, 0x10(r4)
.text:0000ABD4                 stw       r9, 0x14(r4)
.text:0000ABD8                 stw       r9, 0x18(r4)
.text:0000ABDC                 stw       r9, 0x1C(r4)
.text:0000ABE0                 stw       r9, 0x20(r4)
.text:0000ABE4                 lwz       r0, 4(r11)
.text:0000ABE8                 lwz       r3, LC..10_TC # aEpci # "EPCI"
.text:0000ABEC                 lwz       r5, device_config_TC # 0xD910
.text:0000ABF0                 li        r6, 0
.text:0000ABF4                 li        r7, 1
.text:0000ABF8                 mr        r8, r0
.text:0000ABFC                 stw       r2, 0x48+saved_toc(r1)
.text:0000AC00                 lwz       r10, 0(r8)
.text:0000AC04                 lwz       r2, 4(r8)
.text:0000AC08                 mtlr      r10
.text:0000AC0C                 lwz       r11, 8(r8)
.text:0000AC10                 blrl
```

Figure 69. PCIE Driver

Finding the Path to snmpd

Both the AFDX and PCIE drivers have the ARINC 653 Queuing/Sampling ports logic implemented, but as seen in the diagram below, the Socket Abstraction Layer is implemented on top of this layer in the AFDX driver.

The entire sequence required to reach the snmpd daemon from both inter-partition and the Avionics System LAN perspective follows.

WSAStartup

As with a Windows process, when any of the AFDR-3700 applications wants to use 'Winsock API version 2.2' it has to first initialize it by calling `WSAStartup`.

```
text:000263C8                           .csect _spistart.rw_c[RO]
text:000263C8 52 6F 63 6B+_spistart.rw_c: .string "Rockwell-Collins CoRE - AFDX_ASL WinSock API Version 2.2\n"
text:000263C8 77 65 6C 6C+                                       # DATA XREF: .WSPStartup+160↑o
text:000263C8 2D 43 6F 6C+                                       # .data:LC..5_TC_0↓o
text:000263C8 6C 69 6E 73+              .byte 0
text:00026402 00 00                     .short 0
text:00026404 41 46 44 58+aAfdx_aslDriver:.string "AFDX_ASL Driver ID: "
text:00026404 5F 41 53 4C+                                       # DATA XREF: .WSPStartup+180↑o
text:00026404 20 44 72 69+                                       # .data:LC..7_TC_0↓o
text:00026404 76 65 72 20+              .byte 0
text:00026419 00 00 00                  .byte 0, 0, 0
text:0002641C 31 2E 36 00 al_6:         .string "1.6"            # DATA XREF: .WSPStartup+1A0↑o
```

Figure 70. `WSAStartup`

Here we find the first check, as previously mentioned, `WSAStartup` checks whether the VM invoking the function is allowed to even create a socket.

```
text:0000DE28
text:0000DE28             loc_DE28:                      # CODE XREF: .WSPStartup+5C↑j
text:0000DE28 81 22 01 A8         lwz     r9, curr_vm_TC # curr_vm
text:0000DE2C 81 69 00 00         lwz     r11, 0(r9)
text:0000DE30 81 22 01 AC         lwz     r9, SckAllocCnfg_TC # SckAllocCnfg
text:0000DE34 81 29 00 00         lwz     r9, 0(r9)
text:0000DE38 55 6B 18 38         slwi    r11, r11, 3
text:0000DE3C 7D 4B 4A 14         add     r10, r11, r9
text:0000DE40 7C 0B 4A 2E         lhzx    r0, r11, r9
text:0000DE44 2C 00 00 00         cmpwi   r0, 0
text:0000DE48 40 82 00 10         bne     loc_DE58
text:0000DE4C A0 0A 00 02         lhz     r0, 2(r10)
text:0000DE50 2C 00 00 00         cmpwi   r0, 0
text:0000DE54 41 82 00 34         beq     loc_DE88
```

Figure 71. `SckAllocCnfg`

According to `sckAllocCnfg` (each entry is 8 bytes) only VM0 (0x29 sockets) and VM1 (2 sockets) will be able to allocate sockets.

```
sckAllocCnfg.bin
00  00290000 00000000   )
08  00020000 00000000
10  00000000 00000000
18  00000000 00000000
20  00000000 00000000
28  00000000 00000000
30  00000000 00000000
38  00000000 00000000
40  00000000 00000000
48  00000000 00000000
50  00000000 00000000
58  00000000 00000000
60  00000000 00000000
68  00000000 00000000
70  00000000 00000000
78  00000000 00000000
```

Figure 72. `sckAllocCnfg.bin`

The following functions comprise the AFDX_ASL Winsock2 API (see Figure 73), which are available through the AFDX's driver IOCTL entry point.

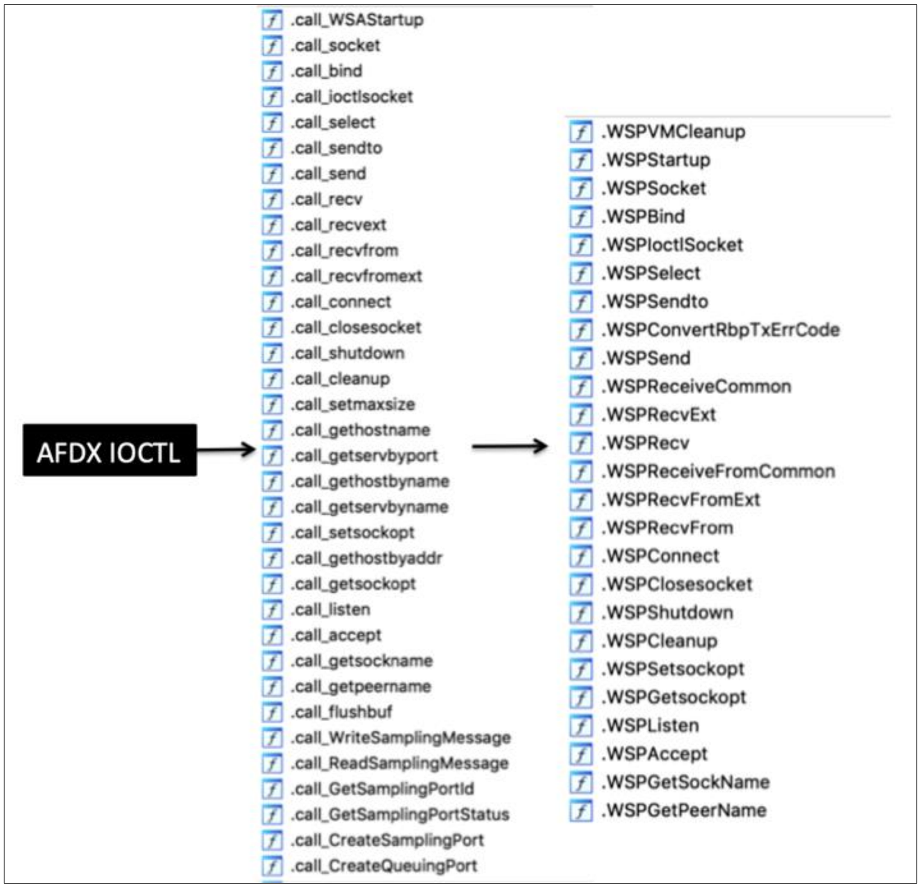


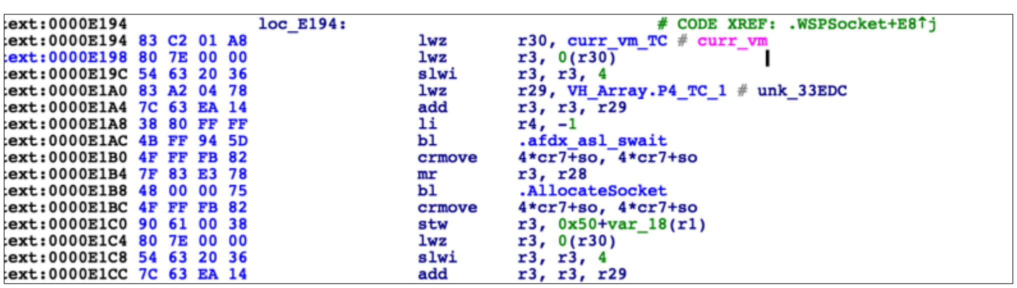
Figure 73. AFDX_ASL Winsock2 API Functions

Create Socket

After calling `WSAStartup`, `snmpd` will try to open a socket at the port 161 to attend SNMP requests. This ends up invoking `WSPSocket` (see Figure 74) which checks:

- If the Socket layer has been initialized for the VM
- The kind of socket the application is trying to create (either a UDP or RBP socket)

If everything is fine, it creates the socket, which is added to a global array of sockets.



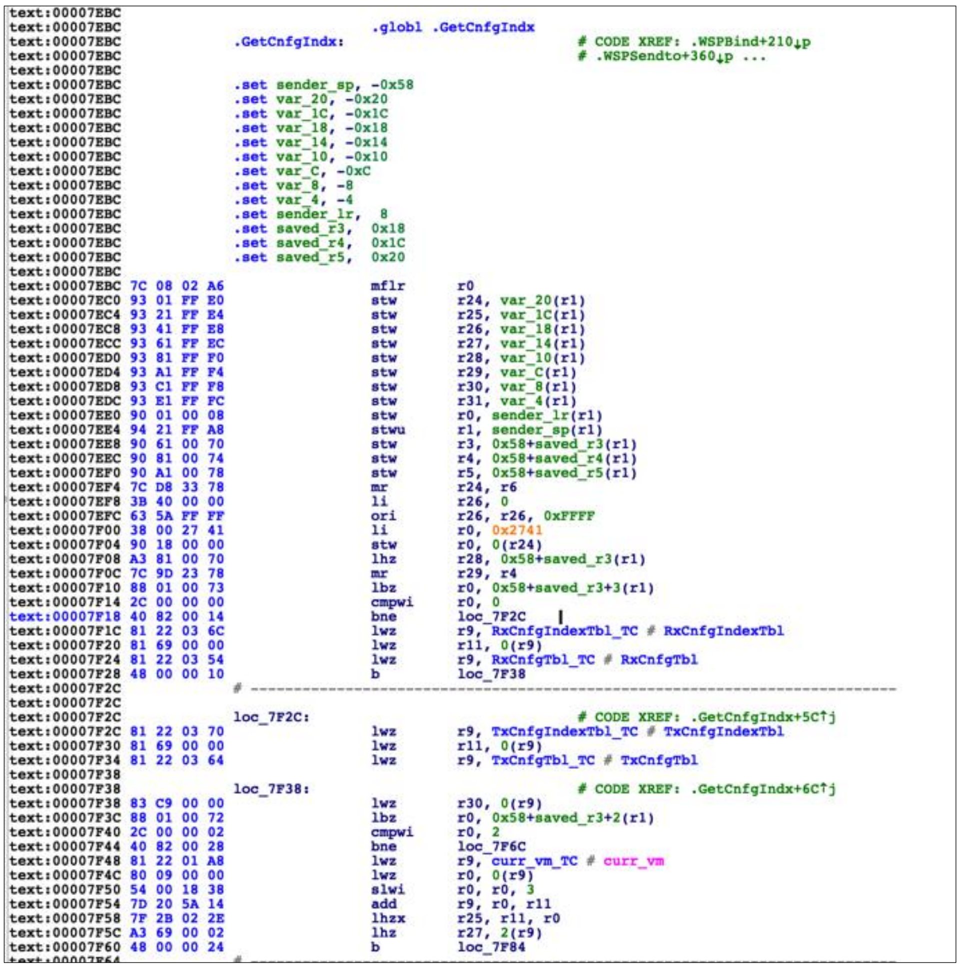
```
ext:0000E194             loc_E194:                    # CODE XREF: .WSPSocket+E8↑j
ext:0000E194 83 C2 01 A8          lwz      r30, curr_vm_TC # curr_vm
ext:0000E198 80 7E 00 00          lwz      r3, 0(r30)
ext:0000E19C 54 63 20 36          slwi     r3, r3, 4
ext:0000E1A0 83 A2 04 78          lwz      r29, VH_Array.P4_TC_1 # unk_33EDC
ext:0000E1A4 7C 63 EA 14          add      r3, r3, r29
ext:0000E1A8 38 80 FF FF          li       r4, -1
ext:0000E1AC 4B FF 94 5D          bl       .afdx_asl_swait
ext:0000E1B0 4F FF FB 82          crmove   4*cr7+so, 4*cr7+so
ext:0000E1B4 7F 83 E3 78          mr       r3, r28
ext:0000E1B8 48 00 00 75          bl       .AllocateSocket
ext:0000E1BC 4F FF FB 82          crmove   4*cr7+so, 4*cr7+so
ext:0000E1C0 90 61 00 38          stw      r3, 0x50+var_18(r1)
ext:0000E1C4 80 7E 00 00          lwz      r3, 0(r30)
ext:0000E1C8 54 63 20 36          slwi     r3, r3, 4
ext:0000E1CC 7C 63 EA 14          add      r3, r3, r29
```

Figure 74. `WSPSocket`

Bind Socket

As expected, `WSPBind` needs to perform several verifications according to the network configuration tables before letting the application bind a socket.

1. `GetCnfgIndx` uses the VM ID (0 in this case), looks into `RxCnfgIndexTbl`, and checks for the allowed range of entries the VM owns in `RxCnfgTbl`. In this current configuration, the operation that VM0 is requesting is checked against the first 0x10 entries. For VM1, the only available entry would be the last one.



```
text:00007EBC
text:00007EBC                           .globl .GetCnfgIndx
text:00007EBC            .GetCnfgIndx:                            # CODE XREF: .WSPBind+210↓p
text:00007EBC                                                     # .WSPSendto+360↓p ...
text:00007EBC
text:00007EBC            .set sender_sp, -0x58
text:00007EBC            .set var_20, -0x20
text:00007EBC            .set var_1C, -0x1C
text:00007EBC            .set var_18, -0x18
text:00007EBC            .set var_14, -0x14
text:00007EBC            .set var_10, -0x10
text:00007EBC            .set var_C, -0xC
text:00007EBC            .set var_8, -8
text:00007EBC            .set var_4, -4
text:00007EBC            .set sender_lr,  8
text:00007EBC            .set saved_r3,  0x18
text:00007EBC            .set saved_r4,  0x1C
text:00007EBC            .set saved_r5,  0x20
text:00007EBC
text:00007EBC 7C 08 02 A6               mflr    r0
text:00007EC0 93 01 FF E0               stw     r24, var_20(r1)
text:00007EC4 93 21 FF E4               stw     r25, var_1C(r1)
text:00007EC8 93 41 FF E8               stw     r26, var_18(r1)
text:00007ECC 93 61 FF EC               stw     r27, var_14(r1)
text:00007ED0 93 81 FF F0               stw     r28, var_10(r1)
text:00007ED4 93 A1 FF F4               stw     r29, var_C(r1)
text:00007ED8 93 C1 FF F8               stw     r30, var_8(r1)
text:00007EDC 93 E1 FF FC               stw     r31, var_4(r1)
text:00007EE0 90 01 00 08               stw     r0, sender_lr(r1)
text:00007EE4 94 21 FF A8               stwu    r1, sender_sp(r1)
text:00007EE8 90 61 00 70               stw     r3, 0x58+saved_r3(r1)
text:00007EEC 90 81 00 74               stw     r4, 0x58+saved_r4(r1)
text:00007EF0 90 A1 00 78               stw     r5, 0x58+saved_r5(r1)
text:00007EF4 7C D8 33 78               mr      r24, r6
text:00007EF8 3B 40 00 00               li      r26, 0
text:00007EFC 63 5A FF FF               ori     r26, r26, 0xFFFF
text:00007F00 38 00 27 41               li      r0, 0x2741
text:00007F04 90 18 00 00               stw     r0, 0(r24)
text:00007F08 A3 81 00 70               lhz     r28, 0x58+saved_r3(r1)
text:00007F0C 7C 9D 23 78               mr      r29, r4
text:00007F10 88 01 00 73               lbz     r0, 0x58+saved_r3+3(r1)
text:00007F14 2C 00 00 00               cmpwi   r0, 0
text:00007F18 40 82 00 14               bne     loc_7F2C
text:00007F1C 81 22 03 6C               lwz     r9, RxCnfgIndexTbl_TC # RxCnfgIndexTbl
text:00007F20 81 69 00 00               lwz     r11, 0(r9)
text:00007F24 81 22 03 54               lwz     r9, RxCnfgTbl_TC # RxCnfgTbl
text:00007F28 48 00 00 10               b       loc_7F38
text:00007F2C            # ---------------------------------------------------------------
text:00007F2C
text:00007F2C            loc_7F2C:                                # CODE XREF: .GetCnfgIndx+5C↑j
text:00007F2C 81 22 03 70               lwz     r9, TxCnfgIndexTbl_TC # TxCnfgIndexTbl
text:00007F30 81 69 00 00               lwz     r11, 0(r9)
text:00007F34 81 22 03 64               lwz     r9, TxCnfgTbl_TC # TxCnfgTbl
text:00007F38
text:00007F38            loc_7F38:                                # CODE XREF: .GetCnfgIndx+6C↑j
text:00007F38 83 C9 00 00               lwz     r30, 0(r9)
text:00007F3C 88 01 00 72               lbz     r0, 0x58+saved_r3+2(r1)
text:00007F40 2C 00 00 02               cmpwi   r0, 2
text:00007F44 40 82 00 28               bne     loc_7F6C
text:00007F48 81 22 01 A8               lwz     r9, curr_vm_TC # curr_vm
text:00007F4C 80 09 00 00               lwz     r0, 0(r9)
text:00007F50 54 00 18 38               slwi    r0, r0, 3
text:00007F54 7D 20 5A 14               add     r9, r0, r11
text:00007F58 7F 2B 02 2E               lhzx    r25, r11, r0
text:00007F5C A3 69 00 02               lhz     r27, 2(r9)
text:00007F60 48 00 00 24               b       loc_7F84
text:00007F64            # ---------------------------------------------------------------
```

Figure 75. Get Configuration Index

2. `TestAndClaimConfigIndex` will check the requested parameters (IPs, ports) to verify that specific socket operation matches the entries in corresponding configuration table (either `RxCnfgTable` or `TxCnfgTable`).

3. If all the checks passed, the request will be pushed down to the ARINC653 layer described previously.

```
text:0000E648                # ---------------------------------------------------------------------------
text:0000E648
text:0000E648                loc_E648:                                # CODE XREF: .WSPBind+23C↑j
text:0000E648 7F A3 EB 78                mr      r3, r29
text:0000E64C 38 80 00 02                li      r4, 2
text:0000E650 4B FF AE A5                bl      .createRxPorts
text:0000E654 4F FF FB 82                crmove  4*cr7+so, 4*cr7+so
text:0000E658 2C 03 00 00                cmpwi   r3, 0
text:0000E65C 41 82 FF 54                beq     loc_E5B0
text:0000E660 7F 83 E3 78                mr      r3, r28
text:0000E664 7F A4 EB 78                mr      r4, r29
text:0000E668 4B FF 9F 1D                bl      .NameSocket
text:0000E66C 4F FF FB 82                crmove  4*cr7+so, 4*cr7+so
text:0000E670 81 3B 00 00                lwz     r9, 0(r27)
text:0000E674 7D 3F 4A 14                add     r9, r31, r9
text:0000E678 A3 49 00 16                lhz     r26, 0x16(r9)
text:0000E67C 7C 1A F0 00                cmpw    r26, r30
text:0000E680 41 82 00 50                beq     loc_E6D0
text:0000E684 81 22 04 80                lwz     r9, TxCnfgTbl_TC_0 # TxCnfgTbl
text:0000E688 80 09 00 00                lwz     r0, 0(r9)
text:0000E68C 57 49 28 34                slwi    r9, r26, 5
text:0000E690 7D 29 02 14                add     r9, r9, r0
text:0000E694 80 69 00 08                lwz     r3, 8(r9)
text:0000E698 A0 89 00 1E                lhz     r4, 0x1E(r9)
text:0000E69C 38 A0 00 00                li      r5, 0
text:0000E6A0 38 C1 00 4C                addi    r6, r1, 0x78+var_2C
text:0000E6A4 48 00 61 A9                bl      .CreateQueuingPort_WinSock
text:0000E6A8 4F FF FB 82                crmove  4*cr7+so, 4*cr7+so
text:0000E6AC 80 61 00 4C                lwz     r3, 0x78+var_2C(r1)
text:0000E6B0 4B FF AF 59                bl      .portCreatedSuccessfully
text:0000E6B4 4F FF FB 82                crmove  4*cr7+so, 4*cr7+so
text:0000E6B8 2C 03 00 00                cmpwi   r3, 0
text:0000E6BC 41 82 FE F4                beq     loc_E5B0
text:0000E6C0 7F 83 E3 78                mr      r3, r28
text:0000E6C4 7F 44 D3 78                mr      r4, r26
text:0000E6C8 4B FF 9F 2D                bl      .AssocSocket
text:0000E6CC 4F FF FB 82                crmove  4*cr7+so, 4*cr7+so
```

Figure 76. AFDX ASL Driver - `WSPBind` Function

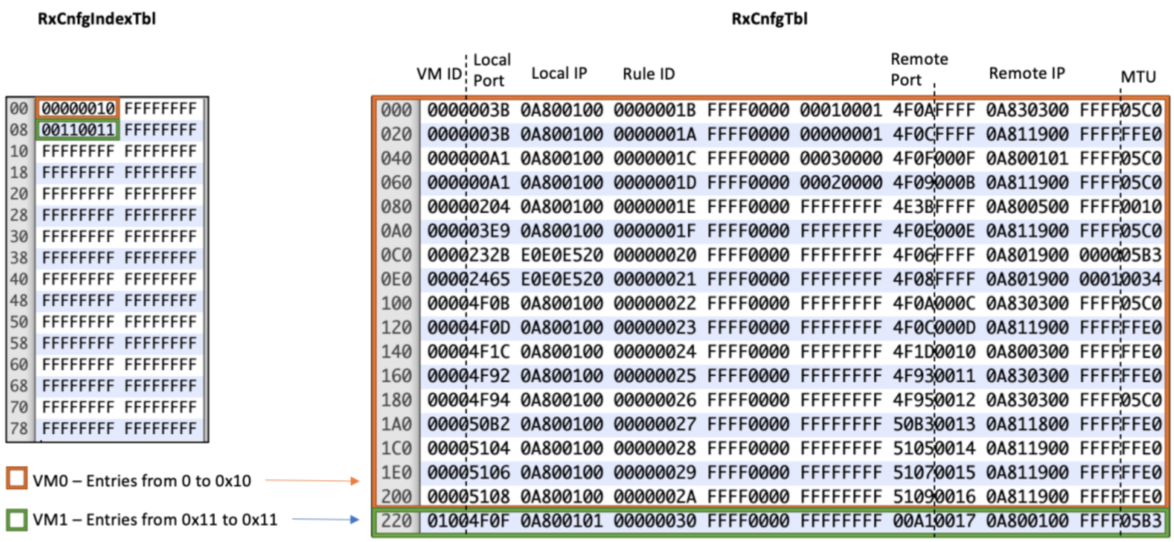
Recvfrom

`snmpd` is now ready to receive data from the authorized clients. When `recvfrom` is invoked, `WSPReceiveCommon` will eventually invoke `ReadQueuingMessage_WinSock`, which will receive the data from the required logical device as previously mentioned, based on the `653PortCnfgTbl` configuration (see Figure 77).

```
ext:0000FBE0
ext:0000FBE0                loc_FBE0:                               # CODE XREF: .WSPReceiveCommon+22C↑j
ext:0000FBE0 7F 83 E3 78                mr      r3, r28
ext:0000FBE4 4B FF 9A 3D                bl      .selectNextRxIndex
ext:0000FBE8 4F FF FB 82                crmove  4*cr7+so, 4*cr7+so
ext:0000FBEC 7C 7D 1B 78                mr      r29, r3
ext:0000FBF0 7C 1D F8 00                cmpw    r29, r31
ext:0000FBF4 41 82 01 A8                beq     loc_FD9C
ext:0000FBF8 A0 1C 00 9A                lhz     r0, 0x9A(r28)
ext:0000FBFC 7C 00 F8 00                cmpw    r0, r31
ext:0000FC00 40 82 01 38                bne     loc_FD38
ext:0000FC04 93 41 00 64                stw     r26, 0x98+var_34(r1)
ext:0000FC08 7C 1D EA 14                add     r0, r29, r29
ext:0000FC0C 39 3C 00 50                addi    r9, r28, 0x50
ext:0000FC10 7C 69 02 2E                lhzx    r3, r9, r0
ext:0000FC14 38 81 00 38                addi    r4, r1, 0x98+var_60
ext:0000FC18 38 A1 00 60                addi    r5, r1, 0x98+var_38
ext:0000FC1C 38 C1 00 64                addi    r6, r1, 0x98+var_34
ext:0000FC20 38 E0 00 01                li      r7, 1
ext:0000FC24 39 01 00 68                addi    r8, r1, 0x98+var_30
ext:0000FC28 48 00 4D 89                bl      .ReadQueuingMessage_WinSock
```

Figure 77. `WSPReceiveCommon`

Taking into account the previous information, we are now in a position to analyze `RxCnfgTbl` in order to discover from where `snmpd` is reachable.



RxCnfgIndexTbl

00	00000010	FFFFFFFF
08	00110011	FFFFFFFF
10	FFFFFFFF	FFFFFFFF
18	FFFFFFFF	FFFFFFFF
20	FFFFFFFF	FFFFFFFF
28	FFFFFFFF	FFFFFFFF
30	FFFFFFFF	FFFFFFFF
38	FFFFFFFF	FFFFFFFF
40	FFFFFFFF	FFFFFFFF
48	FFFFFFFF	FFFFFFFF
50	FFFFFFFF	FFFFFFFF
58	FFFFFFFF	FFFFFFFF
60	FFFFFFFF	FFFFFFFF
68	FFFFFFFF	FFFFFFFF
70	FFFFFFFF	FFFFFFFF
78	FFFFFFFF	FFFFFFFF

RxCnfgTbl

000	0000003B	0A800100	0000001B	FFFF0000	00010001	4F0AFFFF	0A830300	FFFF05C0
020	0000003B	0A800100	0000001A	FFFF0000	00000001	4F0CFFFF	0A811900	FFFFFFE0
040	000000A1	0A800100	0000001C	FFFF0000	00030000	4F0F000F	0A800101	FFFF05C0
060	000000A1	0A800100	0000001D	FFFF0000	00020000	4F09000B	0A811900	FFFF05C0
080	00000204	0A800100	0000001E	FFFF0000	FFFFFFFF	4E3BFFFF	0A800500	FFFF0010
0A0	000003E9	0A800100	0000001F	FFFF0000	FFFFFFFF	4F0E000E	0A811900	FFFF05C0
0C0	0000232B	E0E0E520	00000020	FFFF0000	FFFFFFFF	4F06FFFF	0A801900	000005B3
0E0	00002465	E0E0E520	00000021	FFFF0000	FFFFFFFF	4F08FFFF	0A801900	00010034
100	00004F0B	0A800100	00000022	FFFF0000	FFFFFFFF	4F0A000C	0A830300	FFFF05C0
120	00004F0D	0A800100	00000023	FFFF0000	FFFFFFFF	4F0C000D	0A811900	FFFFFFE0
140	00004F1C	0A800100	00000024	FFFF0000	FFFFFFFF	4F1D0010	0A800300	FFFFFFE0
160	00004F92	0A800100	00000025	FFFF0000	FFFFFFFF	4F930011	0A830300	FFFFFFE0
180	00004F94	0A800100	00000026	FFFF0000	FFFFFFFF	4F950012	0A830300	FFFF05C0
1A0	000050B2	0A800100	00000027	FFFF0000	FFFFFFFF	50B30013	0A811800	FFFFFFE0
1C0	00005104	0A800100	00000028	FFFF0000	FFFFFFFF	51050014	0A811900	FFFFFFE0
1E0	00005106	0A800100	00000029	FFFF0000	FFFFFFFF	51070015	0A811900	FFFFFFE0
200	00005108	0A800100	0000002A	FFFF0000	FFFFFFFF	51090016	0A811900	FFFFFFE0
220	01004F0F	0A800101	00000030	FFFF0000	FFFFFFFF	00A10017	0A800100	FFFF05B3

Figure 78. Rx Configuration Index Table and Rx Configuration Table

According to the Rx configuration tables shown in Figure 78, the vulnerable `snmpd` can be reached both from the VM1 and from a remote node through the Avionics System LAN.

1. **Inter-Partition**

   Rule ID: 0x1C
   Local IP: 10.128.1.0 (0xA800100)
   Local Port: 161/UDP (0xA1)
   Local Host: VM0

   Remote IP: 10.128.1.1 (0xA800101)
   Remote Port: 0x4F0F
   Remote Host: VM1

   The blue arrow in Figure 78 points to Rule ID 0x30, which is the VM1 rule for the SNMP inter-partition communication between VM0 and VM1.

   This entry basically contains the same parameters seen in VM0's Rule 0x1C, but in the opposite direction, as from the VM1 perspective, it is now receiving the response from the `snmpd` server in VM0.

2. **Remote Node (Avionics System LAN)**

   Rule ID: 0x1D
   Local IP: 10.128.1.0 (0xA800100)
   Local Port: 0xA1 (161/UDP) SNMP
   Local Host: VM0

   Remote IP: 10.129.25.0 (0xA811900)
   Remote Port: 20233/UDP

Following the verification process, we find that, as expected, `TxCnfgTbl` contains the complementary rules perfectly matching the ones described above.

TxCnfgTbl

VM ID | Remote Port | Remote IP | Rule ID | Local Port | Local IP | MTU

000	0000003B	0A811900	00000001	FFFF0000	00000016	51080010	0A800100	000005C0
020	0000003B	0A811900	00000000	FFFF0000	00000015	5106000F	0A800100	000005C0
040	0000003B	0A830300	00000002	FFFF0000	00000012	4F94000C	0A800100	000005C0
060	00000045	0A800300	00000003	FFFF0000	00000010	4F1C000A	0A800100	000005C0
080	00000045	0A811800	00000004	FFFF0000	00000013	50B2000D	0A800100	000005C0
0A0	00000045	0A811900	00000005	FFFF0000	00000014	5104000E	0A800100	00008000
0C0	00000045	0A830300	00000006	FFFF0000	00000011	4F92000B	0A800100	000005C0
0E0	0000233C	0A811900	00000008	FFFF0000	0000FFFF	4E21FFFF	0A800100	000005B3
100	0000233D	0A811900	00000009	FFFF0000	0000FFFF	4E24FFFF	0A800100	000005B3
120	000023F2	0A811900	0000000A	FFFF0000	0000FFFF	4E22FFFF	0A800100	00000100
140	00002580	0A800101	0000000B	FFFF0000	0000FFFF	4E25FFFF	0A800100	000005B3
160	00004F09	0A811900	0000000C	FFFF0000	0000000F	00A10003	0A800100	000005B3
180	00004F0A	0A830300	0000000D	FFFF0000	0000FFFF	4F0B0008	0A800100	00001CB3
1A0	00004F0C	0A811900	0000000E	FFFF0000	0000FFFF	4F0D0009	0A800100	00001CB3
1C0	00004F0E	0A811900	0000000F	FFFF0000	0000FFFF	03E90005	0A800100	00000200
1E0	00004F0F	0A800101	00000010	FFFF0000	0000000B	00A10002	0A800100	000005B3
200	00004F1D	0A800300	00000011	FFFF0000	00000003	4F1C000A	0A800100	000005C0
220	00004F93	0A830300	00000012	FFFF0000	00000006	4F92000B	0A800100	000005C0
240	00004F95	0A830300	00000013	FFFF0000	00000002	4F94000C	0A800100	000005C0
260	000050B3	0A811800	00000014	FFFF0000	00000004	50B2000D	0A800100	000005C0
280	00005105	0A811900	00000015	FFFF0000	00000005	5104000E	0A800100	00008000
2A0	00005107	0A811900	00000016	FFFF0000	00000001	5106000F	0A800100	000005C0
2C0	00005109	0A811900	00000017	FFFF0000	00000000	51080010	0A800100	000005C0
2E0	010000A1	0A800100	00000018	FFFF0000	0000FFFF	4F0F0011	0A800101	000005C0

Figure 79. Tx Configuration Table

The last verification step corresponds to PCIE's `in_rx_table`, which is checked by the EPCI logical device before routing the received message from the ASL.

```
text:000096F0                        .globl .read_message
text:000096F0 .read_message:                                  # CODE XREF: .pcie_recv_message+70↓p
text:000096F0                                                 # DATA XREF: .data:read_message↓o
text:000096F0
text:000096F0 .set sender_sp, -0xA8
text:000096F0 .set var_70, -0x70
text:000096F0 .set var_6C, -0x6C
text:000096F0 .set var_68, -0x68
text:000096F0 .set var_64, -0x64
text:000096F0 .set var_60, -0x60
text:000096F0 .set var_5C, -0x5C
text:000096F0 .set var_50, -0x50
text:000096F0 .set var_4C, -0x4C
text:000096F0 .set var_48, -0x48
text:000096F0 .set var_47, -0x47
text:000096F0 .set var_46, -0x46
text:000096F0 .set var_40, -0x40
text:000096F0 .set var_3C, -0x3C
text:000096F0 .set var_34, -0x34
text:000096F0 .set var_30, -0x30
text:000096F0 .set var_2C, -0x2C
text:000096F0 .set var_28, -0x28
text:000096F0 .set var_24, -0x24
text:000096F0 .set var_20, -0x20
text:000096F0 .set var_1C, -0x1C
text:000096F0 .set var_18, -0x18
text:000096F0 .set var_14, -0x14
text:000096F0 .set var_10, -0x10
text:000096F0 .set var_C, -0xC
text:000096F0 .set var_8, -8
text:000096F0 .set var_4, -4
text:000096F0 .set sender_lr,  8
text:000096F0
text:000096F0                        mflr     r0
text:000096F4                        stw      r19, var_34(r1)
text:000096F8                        stw      r20, var_30(r1)
text:000096FC                        stw      r21, var_2C(r1)
text:00009700                        stw      r22, var_28(r1)
text:00009704                        stw      r23, var_24(r1)
text:00009708                        stw      r24, var_20(r1)
text:0000970C                        stw      r25, var_1C(r1)
text:00009710                        stw      r26, var_18(r1)
text:00009714                        stw      r27, var_14(r1)
text:00009718                        stw      r28, var_10(r1)
text:0000971C                        stw      r29, var_C(r1)
text:00009720                        stw      r30, var_8(r1)
text:00009724                        stw      r31, var_4(r1)
text:00009728                        stw      r0, sender_lr(r1)
text:0000972C                        stwu     r1, sender_sp(r1)
text:00009730                        mr       r19, r4
text:00009734                        mr       r20, r5
text:00009738                        mr       r30, r6
text:0000973C                        mr       r24, r7
text:00009740                        li       r26, 0
text:00009744                        li       r29, 0
text:00009748                        li       r23, 0
text:0000974C                        li       r25, 0
text:00009750                        li       r22, 0
text:00009754                        li       r31, 0
text:00009758                        stw      r22, 0xA8+var_40(r1)
text:0000975C                        stw      r22, 0xA8+var_3C(r1)
text:00009760                        bl       .get_rx_table_entry
```

Figure 80. PCIE Driver - `read_message` Function

Within `in_rx_table` is the highlighted entry that matches the incoming `snmpd` rule we analyzed in the AFDX configuration tables.

```
00000001 00000000 0A802100 0A811900 003B4F0C 005A05C8
00010001 00000000 0A802100 0A836300 003B4F0A 000205C0
00020001 00000000 0A802100 0A811900 00A14F09 000205C0
```

Figure 81. `PCIE.dldd in_rx_table`

Another important fact the analysis of `in_rx_table` and `in_tx_table` revealed is that there are similar entries for multiple ASL IPs, which denotes `snmpd` rules are also implemented for other systems different than the AFD, thus opening the door to explore additional attack vectors. It is assumed the same vulnerable 'snmpd' is used in those additional LynxOS-178-based systems (See Figure 8).

Attack Vectors for snmpd

We have two attack vectors that can be used to trigger the vulnerability during any phase of the flight: VM1 and a remote node in the Avionics System LAN (10.129.25.0).

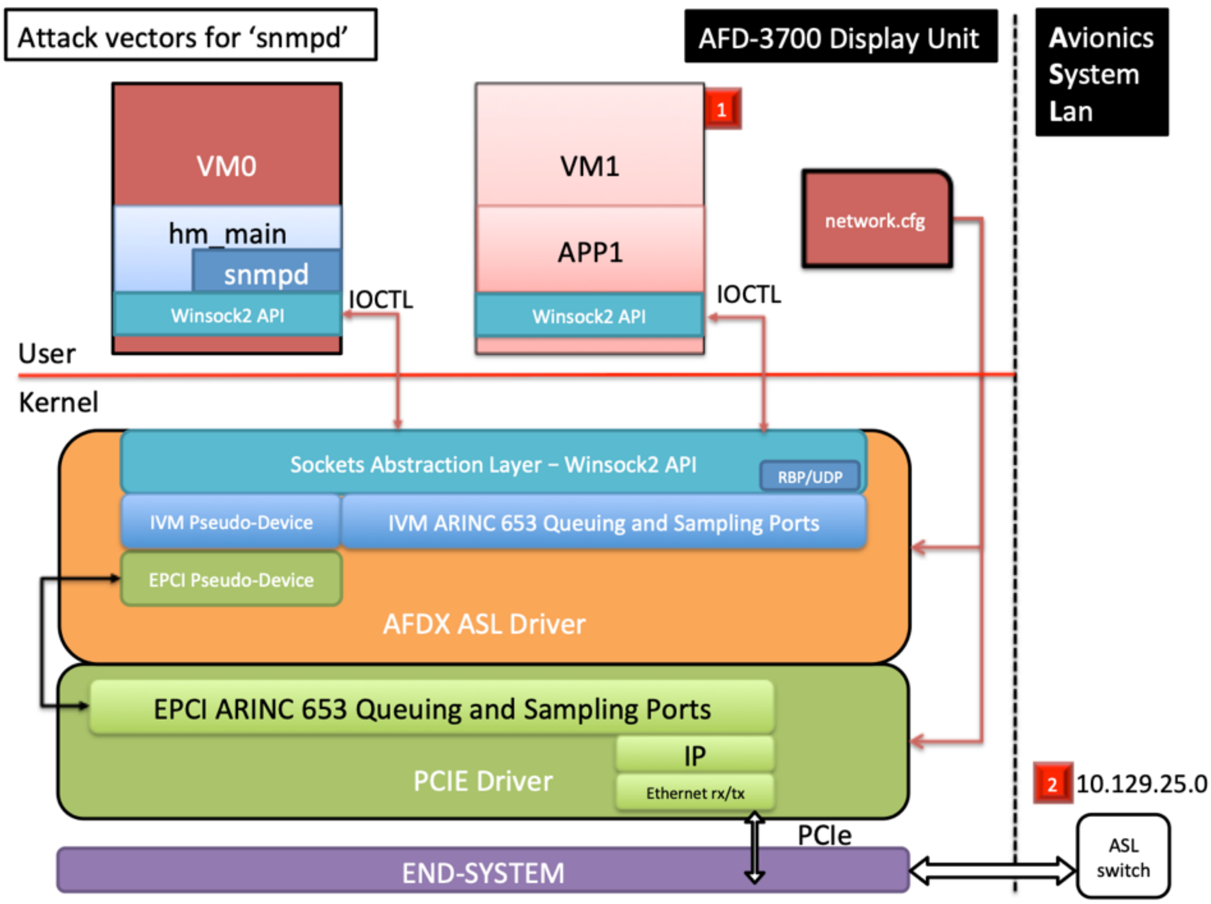


Figure 82. Attack Vectors

1. VM1

The reason for this configured `snmpd` communication channel between VM0 and VM1 is the Simple Display Application (SDA, see Figure 83), which runs in VM1 only when a certain system mode is activated (to perform a data load operation using a USB drive). During 'Normal' system mode, VM1 is assigned to a functional application, such as the ATF-3500 or the FDSA-6500.

This fact is interesting because it leads to a significant logic vulnerability: from a network configuration perspective the system mode is not taken into account, so actually VM1 can launch an attack against VM0 regardless of the application running in VM1. As a result, if a malicious actor compromises the VM1 through methods not covered in this paper, it would be possible to launch an attack against the VM0 by leveraging a deterministic network rule intended for a different system mode.

Figure 83. SDA

2. Avionics System LAN: 10.129.25.0 in the ASL

`HostNameCnfgTbl` can be used to resolve the IP of the potentially offending node 10.129.25.0 (0x0A811900).

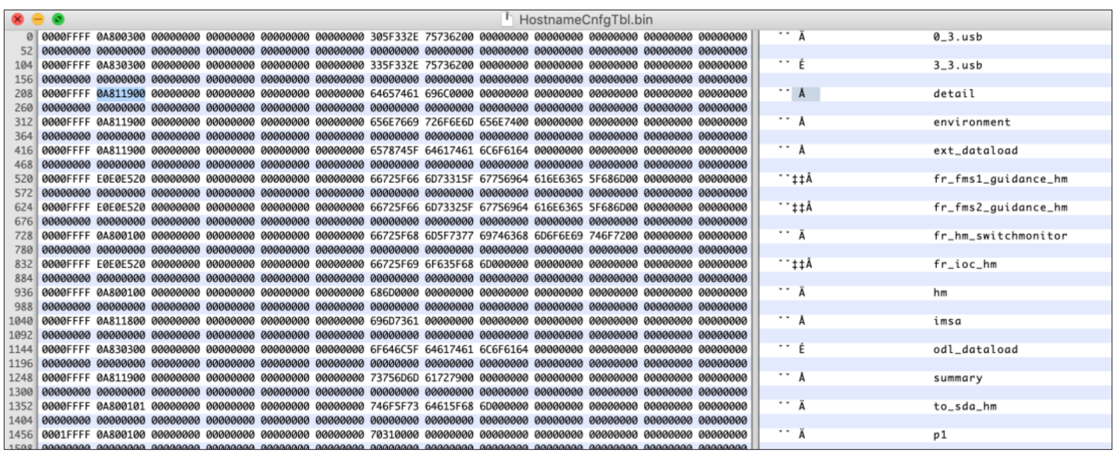


Figure 84. Hostname Configuration Table

It turns out the same IP resolves to four different hostnames:

- `detail`
- `environment`
- `ext_dataload`
- `summary`

This information is quite interesting as the hostname `ext_dataload` may give some clues.

This same device is also performing TFTP operations (see either rule 0x1A in the `RxCnfgTable` or rule 0 in `in_rx_table`), so it seems reasonable to guess we are talking about an ‘External Data Loader’, or a Data Loading Avionics Gateway, such as the Collins’ Information Management System (IMS)[44].

The IMS may be controlled over a WiFi connection.

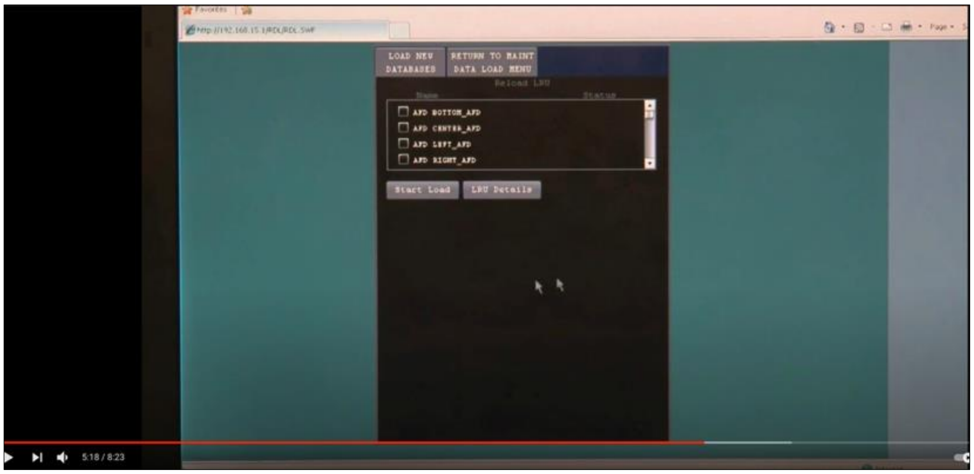
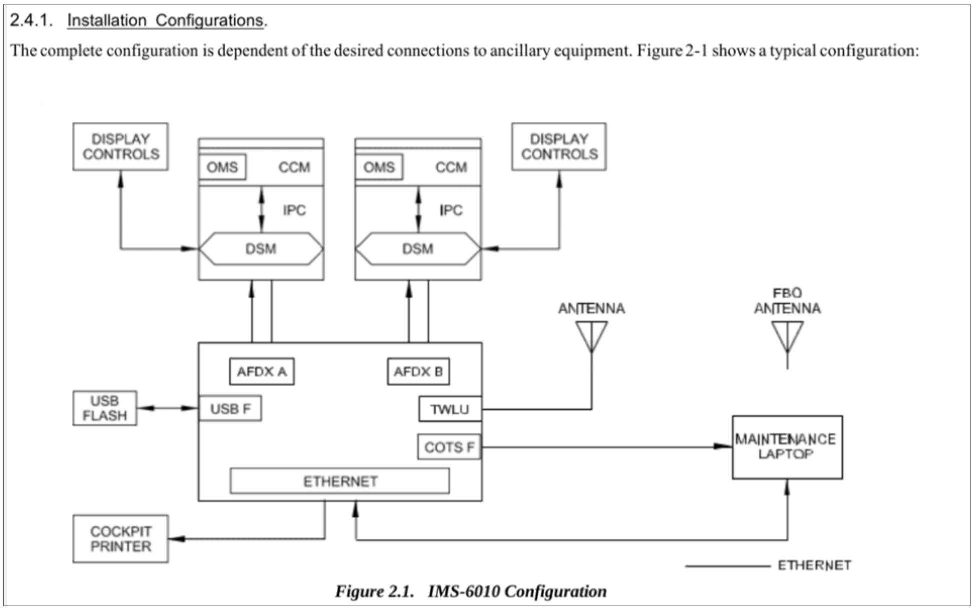


Figure 85. Data Loading over WiFi [45]

Figure 86. WiFi Enabled for IMS Maintenance Operations[46]

[44] https://fccid.io/AJK8223132/User-Manual/Manual-2621284
[45] https://www.youtube.com/watch?v=s20Xjq4HnEQ
[46] https://www.youtube.com/watch?v=9vNRoFKcIB0


2.4.1. Installation Configurations.

The complete configuration is dependent of the desired connections to ancillary equipment. Figure 2-1 shows a typical configuration:

Figure 2.1. IMS-6010 Configuration

Figure 86. IMS-6010 Installation Manual[47]

The installation manual for the IMS-6010 provides a diagram for a typical configuration that also matches the network traffic flows we just analyzed (see Figure 86.)

It is important to clarify that the IMS is just one of the potential attack vectors, which initially depends on the 'on-ground' discrete. Unfortunately, the exposed materials that enabled this research are not enough to explore the remaining attack vectors coming from the ASL.

As a result, a generic approach to reach the ASL from either external/adjacent networks or other compromised components within the network is beyond the scope of this research. The lack of access to a live target forces us to assume that there is no generic way to accomplish this required step for the different aircraft potentially affected, so those scenarios should be addressed on a case-by-case basis.

[47] https://fccid.io/AJK8223132/Users-Manual/Manual-2621284

Attacking AFDR-3700 Drivers

We have been describing the functionality implemented by some of the drivers without assessing the attack vectors they may pose. As we have seen, these drivers may also expose part of their functionality to user-mode through their IOCTL interfaces.

When analyzing the VCTs, we find that some of these drivers are configured without restrictive permissions. Thus, without any additional checks in the ‘open’ entry point, any VM would be able to communicate with the driver.

The following two vulnerabilities are used to illustrate the fact that these drivers are also prone to the same kind of vulnerabilities usually present in drivers from regular Operating Systems.

Exploiting the following vulnerabilities may allow an unprivileged VM to execute code with kernel privileges, thus gaining the ability to compromise the entire LynxOS-178 deployment. In case of a failed exploitation attempt, the attack will leave the LynxOS-178 kernel in an unstable state.

PCIE.dldd: RESET_MIB_DATA IOCTL Double Fetch

The driver fails to declare as ‘volatile’ an attacker-controlled variable that is used in a switch statement. As a result, internally the compiler optimizes the code in such a way that a race condition is created between 0x21B4 and 0x21C4, that can be leveraged to bypass the ‘jumptable’ index check at 0x21BC (see Figure 87). If the malicious threads in the offending partition win the race, it will be possible to jump to an arbitrary memory address, thus potentially executing arbitrary code within the kernel context. It is important to note that LynxOS-178 implements a deterministic scheduler, which facilitates the exploitation of these issues.

```
.text:000021A0
.text:000021A0 loc_21A0:                                 # CODE XREF: .pcie_ioctl+A0↑j
.text:000021A0                                           # DATA XREF: .pcie_ioctl:jpt_1934↑o
.text:000021A0                 mr        r3, r30         # jumptable 00001934 case 146
.text:000021A4                 li        r4, 4
.text:000021A8                 lwz       r5, LC..60_TC # aReset_mib_data # "RESET_MIB_DATA"
.text:000021AC                 bl        .check_read
.text:000021B0                 cmpwi     r3, 0
.text:000021B4                 bne       loc_1C38
.text:000021B8                 lwz       r0, 0(r30)      # first fetch
.text:000021BC                 cmplwi    r0, 6           # switch 7 cases
.text:000021C0                 bgt       def_21DC        # jumptable 00001934 default case
.text:000021C4                 lwz       r0, 0(r30)      # #second fetch
.text:000021C8                 lwz       r9, L..233_TC # jpt_21DC
.text:000021CC                 slwi      r0, r0, 2
.text:000021D0                 lwzx      r0, r9, r0
.text:000021D4                 add       r0, r0, r9
.text:000021D8                 mtctr     r0
.text:000021DC                 bctr                      # switch jump
```

Figure 87. Race Condition

The permissions applied to the driver's device (see Figure 88) leaves the attack open for any VM.

```
<DDD1>                                                      // VCT499
Type=c;                                                     // VCT200
DriverId=;                                                  // VCT201
ObjectFname=/usr/bin/pcie.dldd;                             // VCT202
InfoFname=/usr/etc/pcieinfo_policing_on_autoneg.info;       // VCT203
NumOfMinorDevs=0;                                           // VCT204
BaseCharNodeFname=/dev/ddev/pcie;                           // VCT205
BaseBlockNodeFname=;                                        // VCT206
OwnerId=0;                                                  // VCT207
GroupId=0;                                                  // VCT208
Permissions=0666;                                           // VCT209
</DDD1>
```

Figure 88. Driver Permissions

MERGE.dldd: Memory Corruption Due to Integer Overflow

This driver implements two different IOCTLs (0x96 and 0x97) to perform a memory copy operation from driver's internal structure to user-mode memory and vice versa. While validating the IOCTL parameters received from user-mode, the driver fails to properly verify the length, thus leading to a memory corruption scenario that may be potentially leveraged to escalate privileges (see Figure 89).

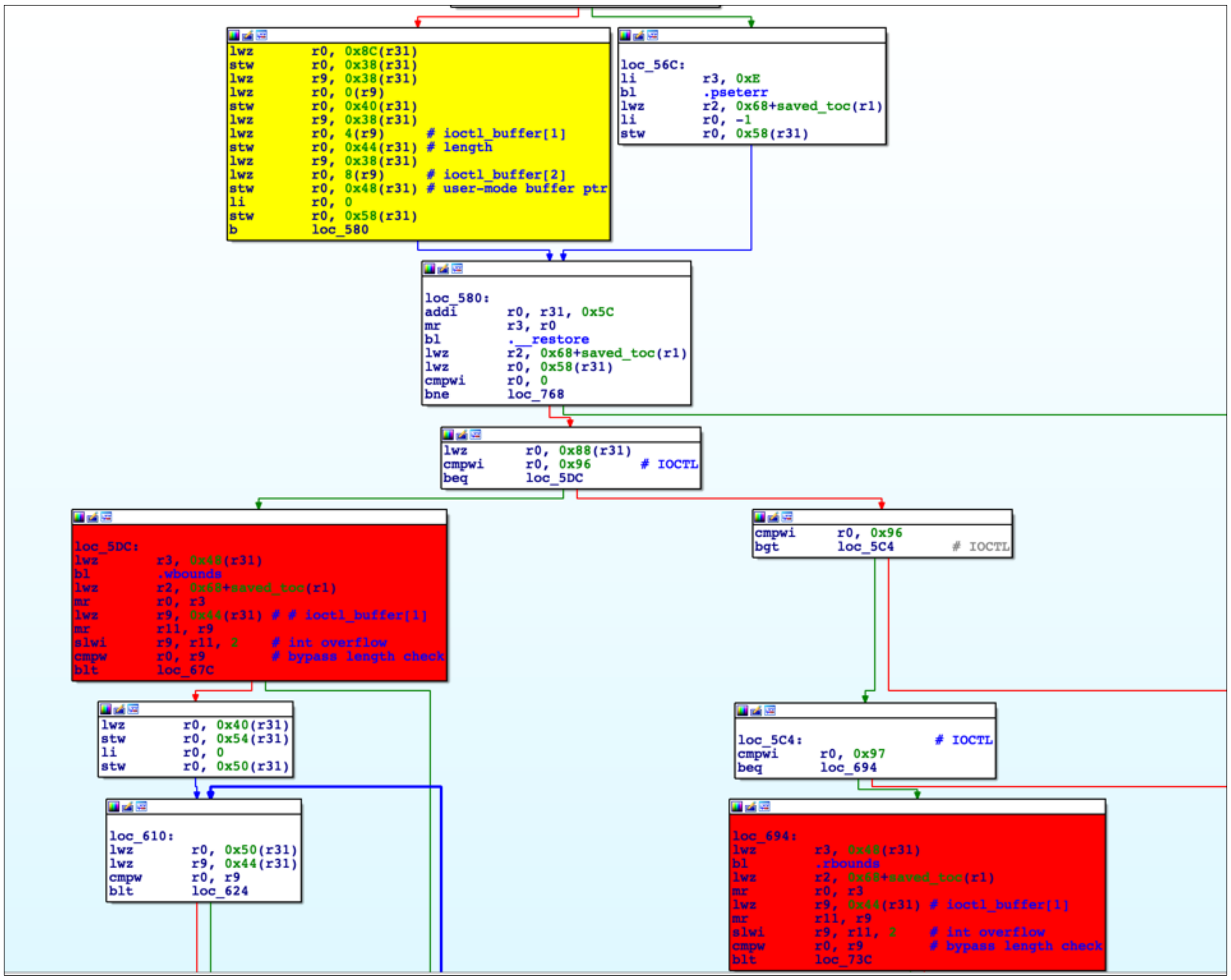


Figure 89. `Merge.dldd` *Vulnerabilities*

Conclusions

This paper has illustrated how the AFDR-3700 software plays a key role in the proper functioning of the following critical devices:

- Primary Flight Display (PFD)
- Multi-Function Display (MFD)

It has also elaborated on the fact that the integrity of functional applications that sustain safety-critical functionality, running under a compromised AFDR-3700, cannot be guaranteed.

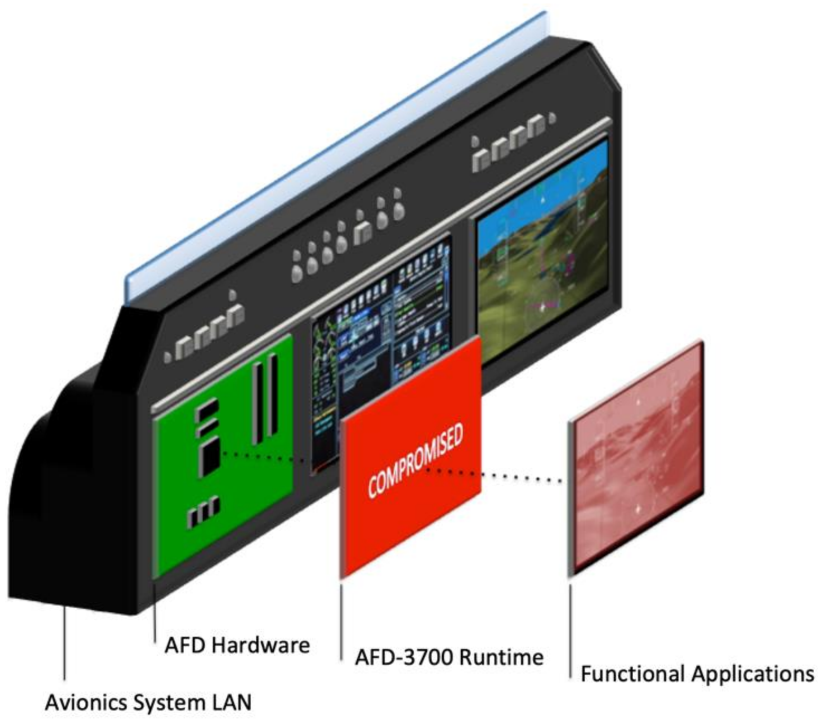


Figure 90. Scenario for a Compromised AFDR-3700

This essentially means that a successful attack may enable the attackers to perform the following actions.

1. Display malicious information to the pilots

This maliciously generated misleading information may include data that does not actually represent the external conditions nor the internal state under which the aircraft is operating.

Disputed statement 2

Collins Aerospace explicitly communicated to IOActive in a letter dated April 7, 2022 that the 'defects identified by IOActive cannot be used or manipulated to cause misleading information to be displayed', also requesting this statement to be deleted from the paper, without providing any further information or technical details.

IOActive is not removing this potential attack scenario mainly due to the following reasons:

1. Among other things, a compromised AFDR-3700 grants the attacker a direct access to low-level graphic resources and video memory in the DU.
2. To facilitate further investigations on this matter.

If any additional information is received, that clearly demonstrates this initial assessment is not aligned to a correct technical analysis, IOActive will proceed to delete this scenario and publicly rectify if required.

2. *Perform a destructive attack that prevents pilots from properly using the PFD/MFD*

A destructive payload may be triggered at certain times, under specific conditions.

The scenarios where destructive attacks can be performed may vary, depending on whether the target is a military or a commercial aircraft.

It is worth mentioning that even in a case where the PFD/MFD may be rendered inoperable, pilots should still be able to rely on the Standby Display, which is intended to operate independently, in addition to electromechanical instruments.

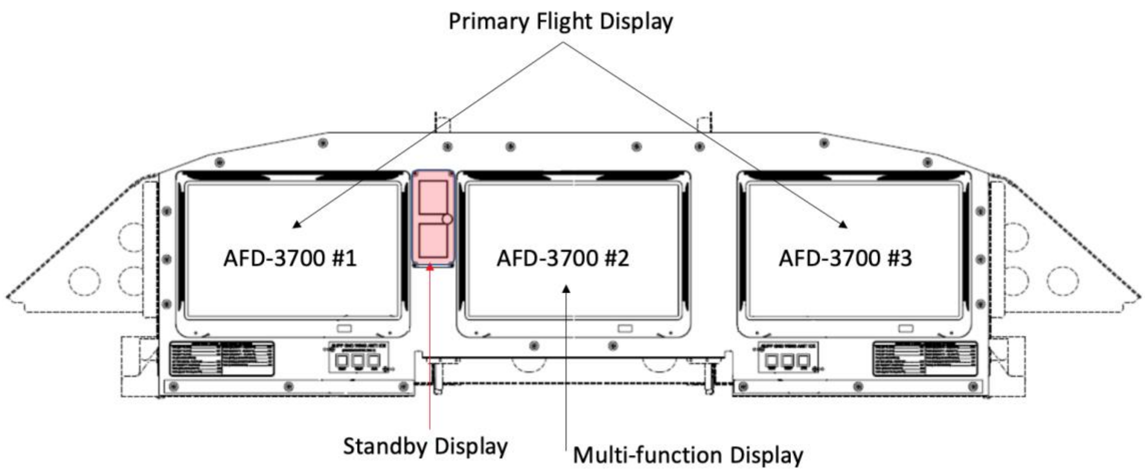


Figure 91 Standby Display

Potential safety implications

The impact of these post-exploitation scenarios will be amplified if the attacks are carried out when the weather conditions force the crew to operate the aircraft according to the instrument flight rules.

As a result, it is IOActive's considered opinion that if the vulnerabilities herein described are successfully exploited, this situation may cause certain potentially unsafe conditions for the aircraft, crew, and passengers.

Disputed statement 3

Collins Aerospace explicitly communicated to IOActive in a letter dated April 7, 2022 that “contrary to the finding in your paper, after significant analysis, testing, and review, Collins has determined that the defects described do not adversely impact operational safety. Consistent with other aerospace research IOActive has undertaken, there are mitigations installed elsewhere in the aircraft architecture that ensure the defects described cannot be activated in a way that would compromise the safety of the aircraft.”

We appreciate the efforts Collins Aerospace dedicated to properly assess these issues. However, it is worth clarifying that IOActive has not been provided with any visibility on these efforts; we know nothing about the methodology, the scope of the analysis or the implemented techniques. We do not know either, where those mitigations are implemented, nor the technical details behind them.

We also consider important to note that Collins’ response is also consistent with previous responses we have received, always pointing to unspecified mitigations, which have been never fully elaborated. Those mitigations are not mapped to specific vulnerabilities or attack scenarios, but proposed as a generic, abstract, concept able to foil any attack. When our previous aerospace research has covered non-certified airborne software, the mitigations were apparently in the certified avionics. Now that we are covering certified avionics, the mitigations are elsewhere.

That said, we have no reasons to not assume that those mitigations are actually in place, and working as expected. However, any serious security research initiative requires a healthy dose of questioning vague statements and paradigms, in order to confront them with reproducible, independently verifiable and consistent technical details.

If any additional information is received, which clearly demonstrates that our initial safety assessment is not aligned to a correct technical analysis, IOActive will proceed to update the paper and publicly rectify if required.

It is not the intention of this research to speculate on complete attack scenarios that may lead to a successful exploitation nor on the composition of post-exploitation payloads. That approach would require extensive information on a variety of both airborne and ground systems as well as technical details of multiple commercial, military, and business aircraft models. As IOActive does not have access to all of the information required for such conclusions, the right thing to do would be to refrain from speculating on these potential scenarios, although we have internally assessed them.

However, it also seems reasonable to raise questions around this situation. In IOActive's experience, the responses we receive from the affected entities usually suggest that these vulnerabilities do not represent an actual risk, due to how the systems are implemented, allegedly following a multilayered protection design. Although these entities do not provide further details on those additional security controls, it is usually expected that the "multiple layers" of defense before reaching the vulnerable component may include physical access control systems within highly secured facilities such as airports[48], as well as non-certified/COTS software and network devices.

The obvious concern we see is that if it were possible to discover the kind of vulnerabilities, presented in this document, in safety-critical avionics software that has been certified according to the highest level of software safety requirements, it would be difficult to assume any greater reliability in the remaining components of these multilayered systems.

Also, these conclusions do not weigh whether real-world attacks against aviation targets are a current trend, even in the current geopolitical situation. In general terms, the threats against safety-critical assets should be evaluated from the perspective that an adversary's capabilities remain consistent, but their intentions may change overnight.

It is important to point out that the extent of this research's conclusions is dictated by its inherent limitations: despite the evidence pointing toward certain scenarios, we will not claim what we cannot publicly demonstrate. On the other hand, in response to the questions this research may generate, we will certainly hope to see technically grounded answers from those who actually have those capabilities.

Finally, the technical details presented herein should be seen as a way to move past the point where "unbreakability" is still claimed for certified avionics that sustain safety-critical operations.

[48] Some of the affected aircraft, such as King Air, can be found also in local aerodromes, which are far behind in terms of physical security compared to commercial airports.

Acknowledgements

We want to thank the following external reviewers, also those who wish to remain anonymous, for their commitment to disinterestedly review this research, as well as for their valuable remarks:

- Peter Lemme

  Aviation Expert

  https://www.linkedin.com/in/satcomguru

- Inbar Raz

  Aviation security researcher, VP of Research at Hunters

  https://il.linkedin.com/in/inbarraz

- Noam Menscher

  Security Researcher, Former Head of Aviation R&D at Argus Cyber Security

  https://il.linkedin.com/in/noam-menscher-233a35134

- Eric S. Johnson

  Pilot and Adjunct Instructor Computer Science, Florida International University

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 https://ioactive.com for more information. Read the IOActive Labs Research Blog: https://labs.ioactive.com. Follow IOActive on Twitter: https://twitter.com/ioactive.