Ku Band Radio Frequency System (KRFS) Version 3 (K3) Migration from Concurrent RedHawk Linux to Red Hat Real-Time Linux

John Jesus

September 4, 2020

Contents

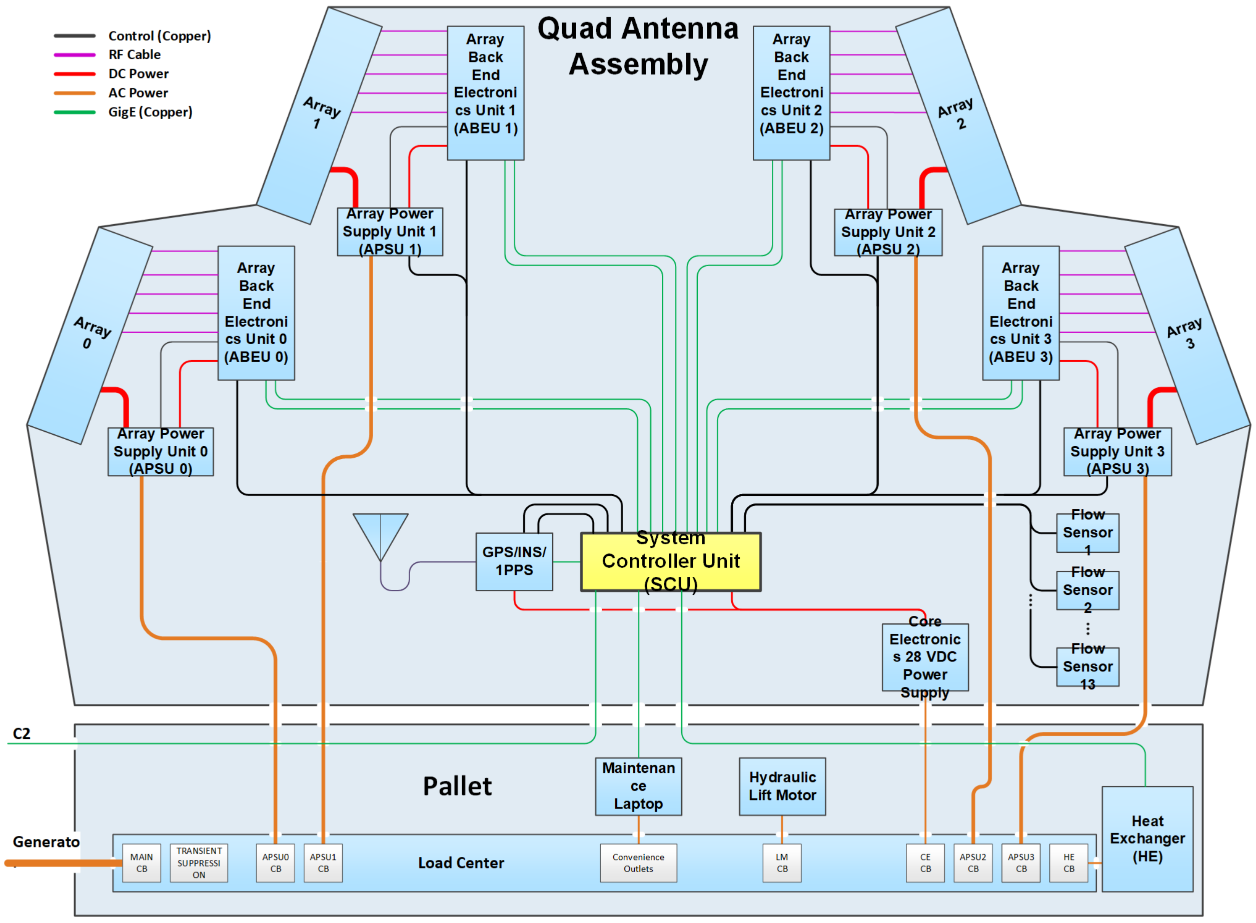
1 Scope

1.1 Identification

This document is a Raytheon internal document analyzing an opportunity to change the version of Linux operating system used in the Ku Band Radio Frequency System (KRFS) Version 3 (K3).

1.2 System overview

KRFS is a radar system that supports the counter- rocket, artillery, and mortar (CRAM) mission. KRFS has four radar apertures each operating as an Active Electronically Scanned Array (AESA). Each aperture has its own receiver/exciter and signal processing subsystem called the Array Backend Electronics Unit (ABEU). The arrays are arranged to provide 360 degree coverage in azimuth and horizon to 90-degree coverage in elevation.



Control (Copper)
RF Cable
DC Power
AC Power
GigE (Copper)
Quad Antenna Assembly
Array 0
Array 1
Array 2
Array 3
Array Back End Electronics Unit 0 (ABEU 0)
Array Back End Electronics Unit 1 (ABEU 1)
Array Back End Electronics Unit 2 (ABEU 2)
Array Back End Electronics Unit 3 (ABEU 3)
Array Power Supply Unit 0 (APSU 0)
Array Power Supply Unit 1 (APSU 1)
Array Power Supply Unit 2 (APSU 2)
Array Power Supply Unit 3 (APSU 3)
GPS/INS/ 1PPS
System Controller Unit (SCU)
Flow Sensor 1
Flow Sensor 2
Flow Sensor 13
Core Electronics 28 VDC Power Supply
C2
Pallet
Maintenance Laptop
Hydraulic Lift Motor
Heat Exchanger (HE)
Generator
MAIN CB
TRANSIENT SUPPRESSION ON
APSU0 CB
APSU1 CB
Load Center
Convenience Outlets
LM CB
CE CB
APSU2 CB
APSU3 CB
HE CB

Figure 1: K3 Radar System

The System Controller Unit (SCU) is a 3-U VPX/VME shelf with commercial-off-the-self (COTS) cards and one custom card assembly (CCA). The SCU is the central controller for the radar system, and the main control software within the SCU runs in its SBC card (Figure 2).

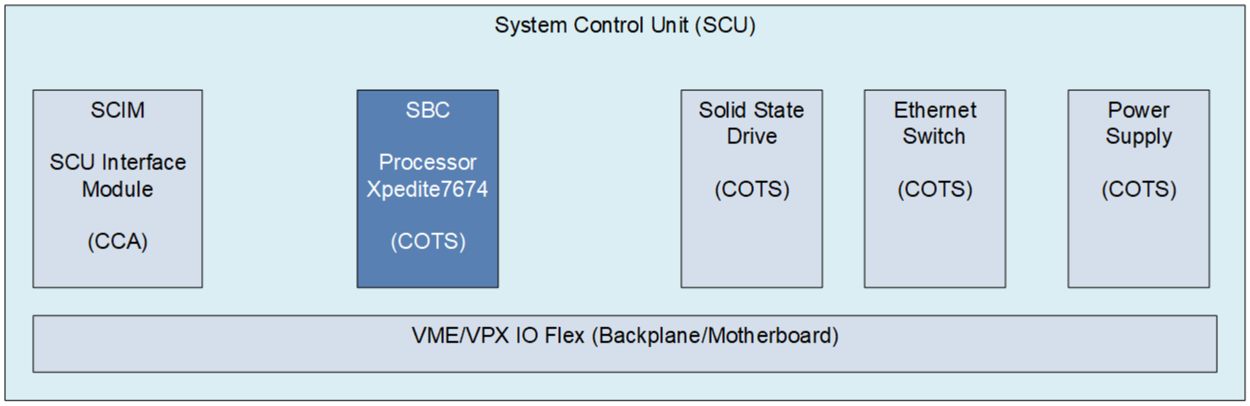


Figure 2: The SBC card in the SCU

In KRFS K2, the SBC processor is a Extreme Engineering (XES) Xpedite7672 card based on the Xeon-D System-on-a-Chip (SOC) running the Linux operating system. In KRFS K3, the SBC processor is a very similar Extreme Engineering (XES) Xpedite7674 card also based on the Xeon-D SOC and running the Linux operating system.

1.3 Overview of Linux operating system and real-time features

The Linux operating system is a free, open-source software (FOSS) operating system originally authored by Linus Torvalds in September 1991 as a Unix system for his personal computer. Linux eventually became the operating system underlying platforms from the smallest devices (Android phones are based on Linux) to supercomputers. Although the source code for the software is freely available, some commercial vendors charge fees to provide the software pre-compiled for a particular device, board, or machine, and charge for support and additional documentation or other tools. Linux initially resembled the Unix systems of its time; it supported multiple users and multiple simultaneously running programs and used a *completely fair scheduler* (CFS) to insure equality among the running processes.

1.3.1 Real-time patch (RT Patch) for the Linux operating system

In the early 2000s, engineers at IBM and Red Hat were working on modifying the Linux kernel source code to support real-time applications. In a real-time system like a radar, some processing has deadlines, for example, transmitting pulses or processing signal returns, where instead of emphasizing fair scheduling among all processes, a real-time operating system (RTOS) must assign priorities to tasks and interrupt (preempt) not only the currently running task but even its own internal mechanisms (spinlocks, semaphores, and interrupt handling) to meet the real-time deadlines. This patch was originally named *PREEMPT_RT* or *CONFIG_PREEMPT_RT* and authored by some of the most important Linux kernel developers, Ingo Molnar and Thomas Gleixner of Red Hat and Paul McKenny of IBM. The source code for the RT Patch has been actively maintained separately from the mainline Linux source code over the last 15 years by teams led by these same developers and has been downloaded and incorporated in thousands of systems and other Linux software distributions.

Some other features of the RT Patch besides kernel-level preemption:

1. CPU shielding - this mechanism constrains the running of operating system threads to certain CPUs in a multicore/multiprocessor system. KRFS uses this feature to protect or *shield* certain CPUs from running Linux kernel lower-priority operations and lower-priority interrupts (such as disk I/O) so that the shielded CPUs can be dedicated to real-time application (radar processing) code.
2. CPU affinity - this mechanism enables application workload to be spread among multiple CPUs to exploit concurrent processing. This assignment of application processes to *cpusets* is specified in userspace (that is, the application programmer decides this partitioning). In KRFS, signal processing application threads can be distributed among shielded CPUs so dedicated and network communications can be partitioned onto other cores/CPUs.

1.3.2 Red Hat Real-Time Products

The first announcement of Red Hat using their RT Patch work in their own products was in February 2007 when IBM and Raytheon IDS of Tewksbury, MA announced at the Boston 2007 Red Hat Summit that Red Hat real-time kernels were running on IBM Blade Server computing clusters for the DDG 1000 Zumwalt Class Destroyer Program.[1]

Later in 2007, Red Hat announced their Red Hat Messaging-Real-time-Grid (MRG) product which bundled the real-time kernel with low-latency messaging components and grid computing (*grid computing* allows Linux to automatically spread workload across any available CPU in a distributed cluster of machines). In this MRG product, Red Hat seemed to be ignoring customers who were interested in real-time but did not need the enterprise cluster computing features, namely, embedded systems developers.

When Red Hat released Red Hat Enterprise Linux 7 (RHEL7) in 2014, the MRG product was deprecated, and *Red Hat for Real-Time* was introduced as a separate product. Red Hat RT includes the standard Linux kernel with the RT patch plus the Red Hat *Tuna* tuning tool and other miscellaneous tools and documentation, aiming this product at the embedded systems market.

1.3.3 SuSE Linux Enterprise Real Time

Soon after IBM and Red Hat released the original Linux RT Patch in the early 2000s, German firm *Software und System-Entwicklung* (SuSE) released *SuSE Linux Enterprise Real Time* which included the RT Patch and added some features:

1. Shield command - SuSE introduced a small python script to simplify the use of the Linux *cpuset* command for confining kernel and user processes to certain processors (this script eventually moved to the main RT Patch)

[1]https://www-03.ibm.com/press/us/en/pressrelease/21033.wss

2. Support for a Real-Time Clock and Interrupt Module (RCIM) - a PCI card that provided programmable timers and 12 input and output external interrupt lines to synchronize clocks across systems. SuSE provided Linux device drivers for this card.
3. A Frequency Based Scheduler (FBS) - SuSE provided a scheduler based on the high-resolution timer provided by the RCIM to support software that performed high-frequency short-deadline processing. SuSE also released a debugging tool providing a view of system utilization during each frequency slice of the scheduler.

Although SuSE implied that they had implemented the real-time kernel features, they were simply including the RT Patch code from Red Hat; Red Hat remarked of the SuSE offering:

> *"With their latest release, (SuSE) has moved to the Red Hat-developed real-time patch set," Che said. "If you look at it from a technology standpoint, (SuSE) has moved to a version aligned with what Red Hat is doing. It validates our approach that you have to work with the upstream Linux kernel." At the London launch of MRG, line of business VP Scott Crenshaw said that (SuSE) had used beta versions of Red Hat's code in its offering. "They haven't contributed a line of code", he said. As a result of this change of code, he argued that "all their prior users are cut off" from previous versions.*[2]

1.3.4 Concurrent Computing Corporation RedHawk Linux

In April 2004, Concurrent Computing Corporation announced a real-time Linux product, containing this excerpt:

> *Concurrent's refinements include kernel-level priority inheritance support, a Frequency Based Scheduler (FBS), process and IRQ shielding extensions, user-level real-time Hyper-Treading control for Intel Xeon platforms, user-level spin locks, significant real-time performance tuning and many additional improvements.*[3]

It seems fairly obvious that this list of so-called *Concurrent's refinements* are simply features inherited from SuSE Real-Time, and, this comment from the Linux Weekly News online was representative of many in the Linux community to Concurrent's product announcement:

> *(Parent article: Concurrent releases RedHawk Linux 2.1)*
>
> *They do not have to release source to the world, but, they certainly do have the obligation to make source available to those to whom they're distributing these customized Linux kernels. The company seems to be very quiet on these obligations, to judge from their web site. I think it's entirely appropriate that they get some scrutiny from the folks whose code they've customized.*[4]

At the same time as its repackaging of RT Patch, Concurrent released a set of tools called *NightStar* that are especially good in tracing the processing of data as it travels among boards and over networks:

[2] `http://www.internetnews.com/dev-news/article.php/3714366`

[3] `https://www.businesswire.com/news/home/20040427005101/en/Concurrent-Announces-Version-2.1-RedHawk-Real-Time-Linux`

[4] `https://lwn.net/Articles/82589/`

> *The RedHawk NightStar Tools allow users on an iHawk system running RedHawk Linux to schedule, monitor, debug and analyze the run time behavior of their real-time applications as well as the RedHawk Linux operating system kernel. The RedHawk NightStar Tools consist of the NightTrace event analyzer; the NightSim frequency-based scheduler; the NightProbe data monitoring tool; the NightView symbolic debugger; the NightTune system and application tuner; shmdefine, a shared memory configuration aid tool; and the Data Monitoring API (Datamon). NightTrace is a graphical tool for analyzing the dynamic behavior of single and multiprocessor applications. NightTrace can log application data events from simultaneous processes executing on multiple CPUs or even multiple systems. NightTrace combines application events with RedHawk Linux kernel events, presenting a synchronized view of the entire system. Many of the tools include a small run-time agent that executes on the RedHawk Linux target system in a non-intrusive manner, preserving the deterministic characteristics of the application.*[5]

Though the excerpt above was over fifteen years ago, the RedHawk product features have not changed, the product instead progressing by moving along with the mainline kernel version and RT Patch and porting to new boards/platforms.

1.3.5 Merge of RT Patch to Linux mainline in July 2019

In July 2019, Linux Torvalds merged the RT Patch into the Linux mainline and the latest Linux kernel can now be made real-time using a configuration option.[6] This merge means that anyone can download the latest Linux kernel source and build it to include all of the real-time features.

1.3.6 Summary of Linux real-time options

1. Each of the real-time kernels offered from Red Hat, RedHawk, and SuSE is the same as the others, merged from the RT Patch originally contributed and currently maintained by Red Hat developers
2. Though it has not changed its Red Hawk or NightStar feature set in 15 years, Concurrent's NightStar tools are very useful for distributed real-time systems
3. Red Hat has always had a real-time kernel product, but, it was obscured by the enterprise cluster features with which it was packaged within Red Hat's MRG product, effectively hiding Red Hat's real-time kernel from consideration by embedded systems developers.
4. When Red Hat released Red Hat Enterprise Linux 7 (RHEL7) in 2014, the MRG product was deprecated, and *Red Hat for Real-Time* was introduced as a separate product. Red Hat RT includes the standard Linux kernel with the RT patch plus the Red Hat *Tuna* tuning tool and other miscellaneous tools and documentation, aiming this product at the embedded systems market.
5. The RT Patch was merged into the Linux Kernel mainline in July 2019 and thus available to anyone who downloads and builds the kernel source code

[5] *RedHawk NightStar Tools Version 2.2 Release Notes March 2005 0898008-2.2*, March 2005

[6] Linus Torvald's commit message is included in its entirety in Appendix A

1.4 Document purpose

This document enables a software engineer or architect to reason about migrating the KRFS Linux operating system to Red Hat Real-Time.

1.5 Document overview

The structure of this document:

1. Document scope (this section)
2. References
3. Real-time features from RedHawk used in current KRFS software
4. Creating the Linux operating system image
5. Hardening the Linux operating system
6. Costs, readiness, and opportunities

2 References

1. *RedHawk Linux Users Guide 0898004-780* (Concurrent Computer Corporation, March 2016)
2. *RedHawk Architect Users Guide 0898601-7.2-1* (Concurrent Computer Corporation, December 2016)
3. *NightStar RT Tutorial Version 4.4 0898009-090* (Concurrent Computer Corporation, May 2014)
4. *Red Hat Enterprise Linux for Real Time 7 Reference Guide* (Red Hat Corporation, November 6, 2015)
5. *Red Hat Enterprise Linux for Real Time 7 Installation Guide* (Red Hat Corporation, November 6, 2015)
6. *Preboot Execution Environment (PXE) Specification Version 2.1* (Intel Corporation, September 20, 1999)
7. *Packaging software for Red Hat Enterprise Linux 8* online document at `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/packaging_and_distributing_software/packaging-software_packaging-and-distributing-software`
8. *Red Hat Enterprise Linux 7 (RHEL 7) Security Technical Implementation Guide (STIG) Version 2, Release 8* (Defense Information Systems Agency (DISA), July 24, 2020)
9. *Ansible role for Red Hat 7 STIG Baseline* (MindPoint Group, June 21, 2020) at `https://github.com/ansible-lockdown/RHEL7-STIG`
10. Source for Red Hawk costs in Table 4 was Karl Weis, Global Supply Chain Program Lead, Raytheon
11. Source for Red Hat Real-Time costs in Table 4 was Lynn Bonenfant, IT Program Lead, Raytheon
12. *Pull CONFIG_PREEMPT_RT stub config from Thomas Gleixner* commit message from the web view of the Linux git repository at `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=70e6e1b971e46f5c1c2d72217ba62401a2edc22b`

3 Real-time features from RedHawk in KRFS

KRFS presently is on the Concurrent RedHawk Linux platform. Table 1 and Table 2 list the Concurrent RedHawk features in the order they appear in Reference 1 even though not all of the features are actually real-time Linux kernel features; for example, hyper-threading is a hardware setting in the Intel processor, the RCIM card is a piece of hardware and the XFS filesystem originally from SGI is technically outside the kernel. All of the real-time kernel features are inherited from the RT Patch described in Section 1.3.1 except for those based on the RCIM card (device driver, scheduler, process accounting).

Only the first two features from the list are explicitly used by KRFS source code.

Table 1: RedHawk real-time feature list (Part 1)

Feature	**Used in KRFS**	**Description**
Processor Shielding	**Yes**	KRFS currently uses the old syntax from SuSE and Concurrent *shield* instead of the now-standard Linux command *cset*
Processor Affinity	**Yes**	KRFS code uses the POSIX thread interface to set processor affinity
User-level Preemption Control	No	User control of preemption of userspace synchronization mechanisms like POSIX semaphores using a system function call *resched_cntl*. Makes kernel space preemption mechanism invocable from userspace.
Fast Block/Wake Services	No	A preemptible semaphore in the kernel available from RT Patch since 2007.
RCIM Driver	No	Concurrent sells a PCI card called the Real-time Clock Interface Module (RCIM). The card can provide a 400ns clock signal to the operating system through an interrupt handler to enable a high-resolution timer feature.
Frequency-Based Scheduler (FBS)	No	Source code for the FBS provided by Concurrent is actually copied without attribution from the original SuSE Linux Real-Time Extensions released in 2006.
/proc Modifications	No	This feature is part of the *usermap* feature (see below) that can be used by NightStar or even regular users.
Kernel Trace Facility	No	This feature is combined with the standard kernel debuggers to support enhanced kernel debugging features using NightStar
ptrace Extensions	No	This feature is combined with the *usermap* feature (below) to support NightStar debugging)

Table 2: RedHawk real-time feature list (Part 2)

Feature	**Used in KRFS**	**Description**
Kernel Preemption	(Yes,built-in)	Kernel Preemption is from the RT Patch
Real-Time Scheduler	(Yes,built-in)	Alternatives to the Linux default Completely Fair Scheduler (CFS) like priority-based SCHED_FIFO and SCHED_RR (round-robin) have been part of the RT Patch since 2007. SCHED_DEADLINE is available in Linux 3.6 (Red Hat 7.6) and later.
Low Latency Enhancements	(Yes,built-in)	Some kernel tuning configuration have been compiled in as defaults.
Priority Inheritance	(Yes,built-in)	From the RT Patch.
High Resolution Process Accounting	No	This feature follows from the use of the RCIM card and driver (above).
Capabilities Support	No	Permission files (*capabilities* files) for features like the *usermap* and */proc mmap* features (below).
Kernel Debuggers	(Yes,built-in)	The built-in kernel debuggers *kdb* and *kgdb* can be configured to work with NightStar.
Kernel Core Dumps/Crash and Live Analysis	(Yes,built-in)	This feature supports NightStar, allowing it to directly map and introspect kernel memory.
User-level Spin Locks	No	User control of preemption of spinlocks in the kernel provided by a special system function call *resched_cntl*. Makes kernel space preemption mechanism invocable from userspace.
usermap and /proc mmap	No	This feature supports NightStar, allowing it to directly map and introspect any user process.
Hyper-threading	(Yes,built-in)	This is a setting in the CPU hardware that is controlled in the BIOS or bootloader that allows in-core multi-processing. Support in mainline Linux for hyper-threading and symmetric multiprocessing (SMP) has been since 2004.
XFS Journaling File System	(Yes,built-in)	The XFS filesystem was invented at Silicon Graphics (SGI) in 1996 and has been available for mainline Linux since 2001.

3.1 KRFS source code changes for a non-RedHawk Linux

3.1.1 Shielding CPUs from low-priority system tasks

CPUs can be shielded by using the *isolcpus* kernel boot parameter to specify one or more CPUs to isolate from the general Linux kernel scheduler algorithms (Listing 1).

Listing 1: Shielding CPUs at at kernel boot command-line

```
# Typical kernel boot command-line specifying isolated CPUs 1-6
#
acpi_enforce_resources=lax intel_iommu=on cma=800M nousb isolcpus=1-6
```

To shield CPUs anytime after kernel boot the *cset* command can be used (Listing 2).

Listing 2: Shielding CPUs at command-line

```
# Create a cpuset for signal processing named "sigproc" on CPU 1-6
#
$ cset set --cpu=1-6 --set=sigproc


# Create a cpuset named "system" on CPU 0, 7, 8
#
$ cset set --cpu=0,7-8 --set=system


# Assign operating system tasks to "system"
#
$ cset proc --move --fromset=root --toset=system
```

3.1.2 CPU affinity for processes and threads

To pin a process onto a particular cpu or cpuset, instead of using the RedHawk *run* command, use the standard *taskset* command.

Listing 3: Use taskset instead of run

```
# In RedHawk, run a program on CPU 5 or 6 (-b stands for "bias")
#
$ run -b 5,6 ./mycommand


# The equivalent of above in conventional Linux
# This command will run the /bin/my-app application on CPU 5 or 6:
#
$ taskset -c 5,6 /bin/my-app
```

KRFS source code already uses the POSIX *pthreads* to set affinity for threads, either in the thread attributes object used at thread creation or using the functions *pthread_setaffinity_np()* or *pthread_attr_setaffinity_np()*.[7]

[7] https://man7.org/linux/man-pages/man3/pthread_attr_setaffinity_np.3.html

4 Creating the Linux operating system image

4.1 RedHawk Boot in K2

4.1.1 Normal Boot with RedHawk in K2

The normal boot of RedHawk Linux in K2 is depicted in Figure 3. On reset, the Xeon-D jumps into the boot flash (bottom of diagram) where two components called CoreBoot and SeaBIOS are installed; both CoreBoot and SeaBIOS come not from RedHawk but from the board vendor X-ES. The Xeon-D begins booting CoreBoot (Item 1) and CoreBoot subsequently loads its payload called SeaBIOS which is a small piece of software that decides the Linux boot method (Item 2).

The RedHawk Linux kernel is in the local SSD flash (upper-right of diagram) on the 7672 with a Linux target filesystem. SeaBIOS accesses the Master Boot Record (MBR) in the local SSD flash (upper-right) and checks for a magic number at the end which was previously inserted by KRFS software (Item 3). The magic number is an indicator that the local SSD contains the normal boot contents, and, SeaBIOS continues to boot from the local SSD if it finds the magic number there; if the magic number is not found, SeaBIOS will advance to the next boot option, which is a network boot over Ethernet (Item 4).

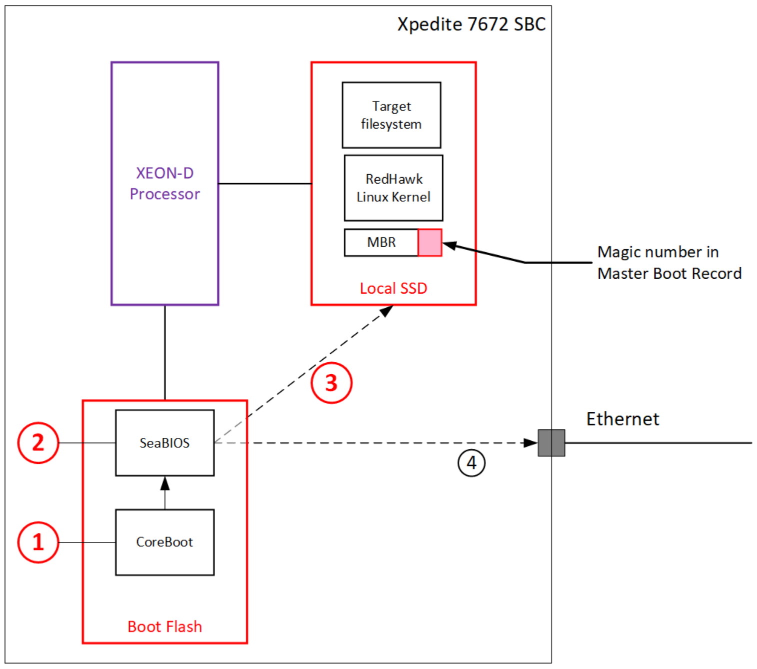


Figure 3: Normal boot of SBC with RedHawk in K2

4.1.2 PXE Boot to Install RedHawk in K2

To install RedHawk onto the local SSD flash, a technique depicted in Figure 4 combines a network boot (called *PXE Boot*) with an embedded installation script. A full description of PXE Boot is in Reference 6.

To force a PXE boot, the magic number in the Master Boot Record of the local SSD (upper-right of diagram) is invalidated. Erasing the SSD will also invalidate the MBR as will using a special KRFS tool to erase the magic number. With the MBR magic number invalidated, when the system is rebooted, SeaBIOS will detect that local SSD is not bootable (Item 3) and will advance to the next boot option which is a PXE Boot (Item 4). SeaBIOS will broadcast PXE boot requests on the Ethernet which will be detected and serviced by a maintenance laptop plugged into the same Ethernet network (Item 5). The laptop provides a small Linux kernel image (called *vmlinuz*) and a tiny root file system (called an initial ramdisk or *initrd.img*) over the network which SeaBIOS loads into RAM and then has the Xeon-D jump into this Linux kernel in RAM (Item 6). Embedded within the initial ramdisk is an installation script, which pulls a tarball called *image.tgz* from the laptop and installs it onto the local SSD (Item 7). This tarball contains the tactical RedHawk Linux Kernel and target filesystem. After a successful install, the installation script inserts the magic number into the Master Boot Record and the local SSD is ready to be booted. The next time the system boots, SeaBIOS will detect that a good image is installed in the local SSD and boot from it.

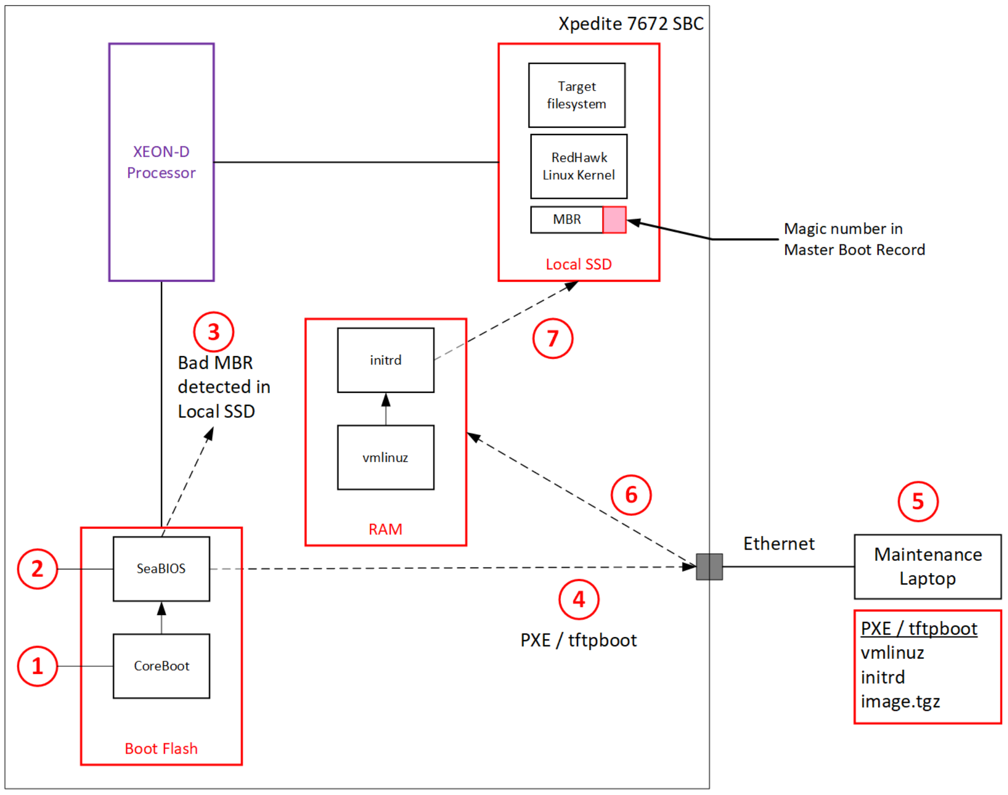


Figure 4: PXE Install of SBC with RedHawk in K2

4.2 RedHawk Architect kernel module build in K2

RedHawk provides a build system called Architect to build the artifacts of Figure 4. RedHawk Architect requires an entire Red Hat Linux machine dedicated to host it as depicted in Figure 5. A dedicated host is required because as stated in Chapter 11 of Reference 1:

> *It is often necessary to build driver modules for use with either one of the pre-existing kernels supplied by Concurrent Real-Time or a custom kernel. To build driver modules for a kernel, the following conditions must be met:*
>
> ***The desired kernel must be the currently running kernel.***
>
> *The -c option to ccur-config can be used to ensure that the kernel source directory is properly configured. This option automatically detects the running kernel and configures the source tree to properly match the running kernel.* ***Driver modules can then be properly compiled for use with the running kernel.***

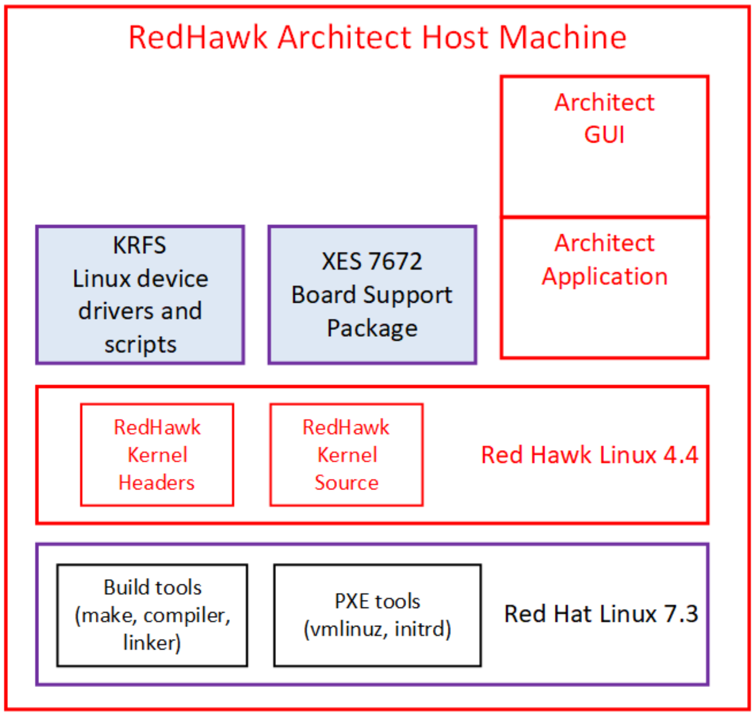

RedHawk Architect Host Machine
Architect GUI
Architect Application
KRFS Linux device drivers and scripts
XES 7672 Board Support Package
RedHawk Kernel Headers
RedHawk Kernel Source
Red Hawk Linux 4.4
Build tools (make, compiler, linker)
PXE tools (vmlinuz, initrd)
Red Hat Linux 7.3

Figure 5: RedHawk Architect host machine

The KRFS Linux device drivers and scripts and the XES Board Support Package (BSP) modules must be copied onto this machine and can only be built for the RedHawk kernel headers and libraries currently running on that machine.

4.2.1 RedHawk build method is unusual and outdated

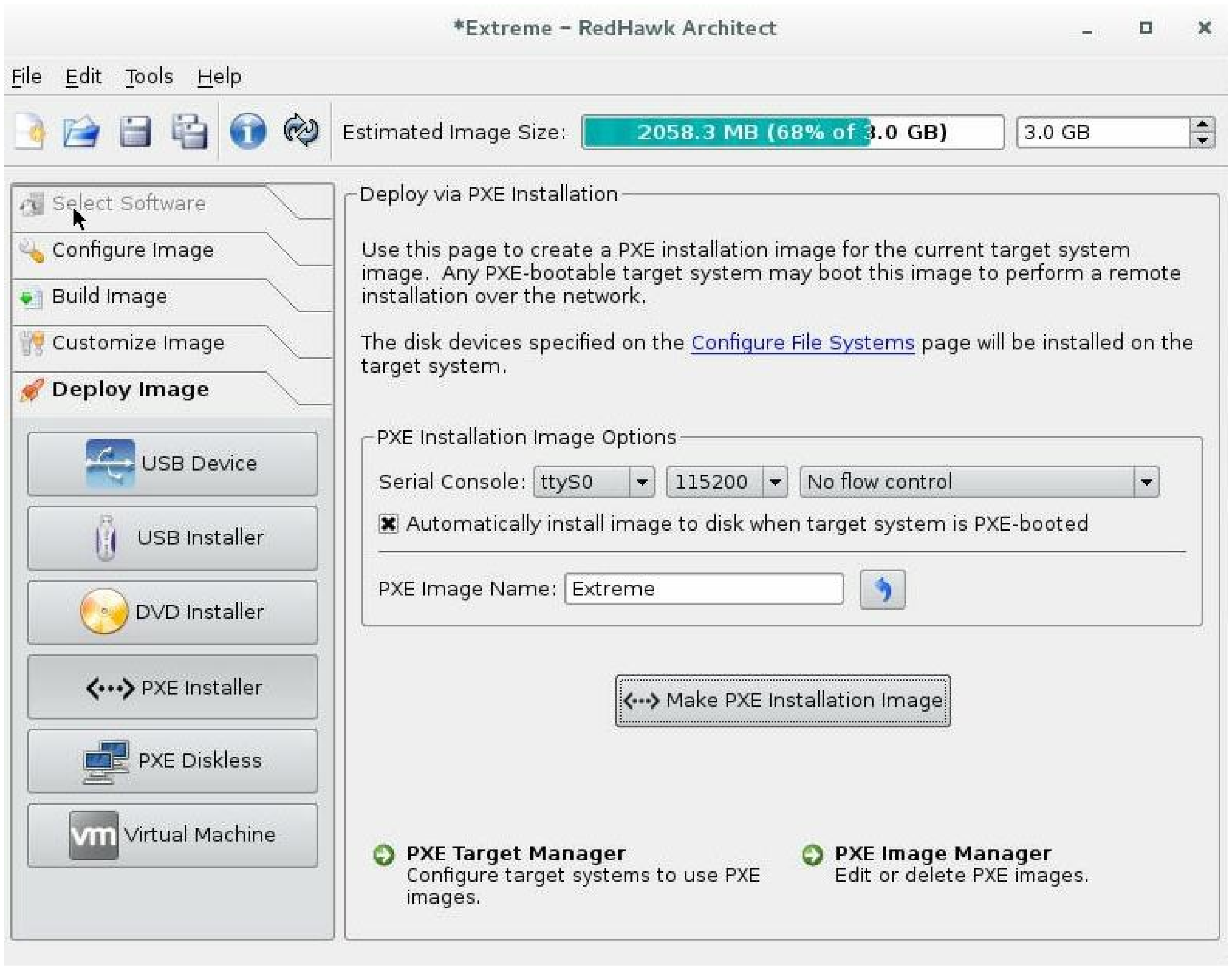
That a kernel module for RedHawk can be built only against the running kernel is an unusual and outdated requirement from the early days of Linux development. Developers typically build RPMs (Red Hat Package Manager) from their home directory against kernel development packages that are also installed in their home directory.[8] These kernel development

[8]Many documents cover building RPM and kernel modules, the Red Hat procedure for Red Hat Enterprise Linux 8 is Reference 7.

packages can be for a different version from the running kernel or even for a different machine architecture (can cross-compile ARM on x86).

4.2.2 Artifacts of RedHawk Architect build

Building for RedHawk can only be performed by a privileged user from the Architect GUI (Figure 6) to produce the build artifacts specified in Table 3.



Estimated Image Size: 2058.3 MB (68% of 3.0 GB) 3.0 GB

Deploy via PXE Installation

Use this page to create a PXE installation image for the current target system image. Any PXE-bootable target system may boot this image to perform a remote installation over the network.

The disk devices specified on the Configure File Systems page will be installed on the target system.

PXE Installation Image Options

Serial Console: ttyS0 115200 No flow control

[✓] Automatically install image to disk when target system is PXE-booted

PXE Image Name: Extreme

PXE Target Manager
Configure target systems to use PXE images.

PXE Image Manager
Edit or delete PXE images.

Figure 6: Screenshot from the RedHawk Architect Tool

Table 3: Artifacts created with RedHawk Architect

	Artifact Name	Size (MB)
PXE Boot files		
	vmlinuz	5.3
	initrd.img	7.0
	other PXE config files	< 1.0
Target system image		
	image.tgz	700.0
	partition info, fstab, other config files	< 1.0

The artifacts are then deployed into the tftpboot area of the Maintenance Laptop for installation on the target using the PXE Boot mechanism of Figure 4.

4.3 Red Hat RT Build

With Red Hat RT, each individual developer or the Continuous Integration (CI) / Continuous Deployment (CD) server (*e.g.*, Jenkins) can install its own copy of the kernel source code or kernel development tools and build a local copy of the kernel, BSP, or device drivers (Figure 7. The open-source Red Hat LiveMedia Creator package can be used for creating the target Linux filesystem and the PXE boot tools.

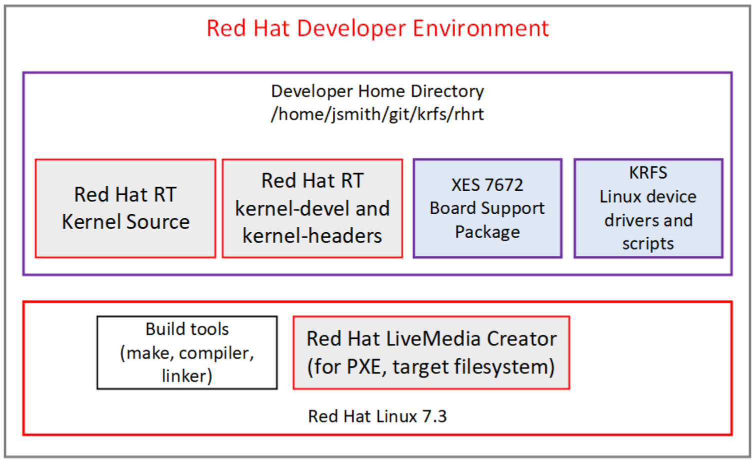

Red Hat Developer Environment
Developer Home Directory
/home/jsmith/git/krfs/rhrt
Red Hat RT Kernel Source
Red Hat RT kernel-devel and kernel-headers
XES 7672 Board Support Package
KRFS Linux device drivers and scripts
Build tools (make, compiler, linker)
Red Hat LiveMedia Creator (for PXE, target filesystem)
Red Hat Linux 7.3

Figure 7: Red Hat RT Developer Environment

4.4 DevSecOps Linux in K3

The K3 DevSecOps Red Hat RT Linux and target filesystem are created and deployed using a DevSecOps facility as depicted in Figure 8.

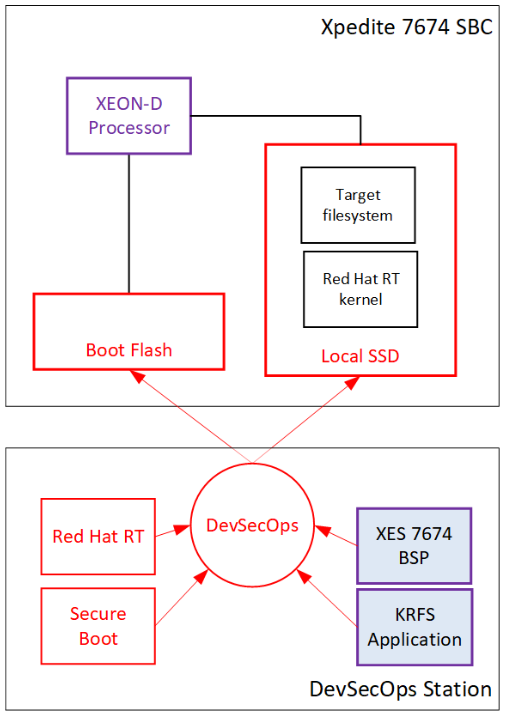


Figure 8: DevSecOps Linux deployment in K3

The DevSecOps facility combines Red Hat RT Linux and target filesystem with the KRFS application and the XES BSP and deploys directly to the bootflash and local SSD of the SBC.

5 Hardening the Linux operating system

To harden an operating system is to minimize vulnerabilities to malicious outside access; for example, requiring a password for login is a basic hardening mechanism.

The Defense Information Systems Agency (DISA) publishes guidance for hardening specific operating systems called Security Technology Implementation Guidance (STIG). For Linux distributions, DISA publishes a STIG for only these Linux Distributions:

1. Red Hat Linux
2. Oracle Linux
3. SuSE Linux

The STIG is published in an unconventional format: as an XML file. An excerpt from the *Red Hat Enterprise Linux 7 STIG* is in Appendix B. The XML file contains hundreds of elements each specifying a security item to be checked and even contains instructions for remedying the item. This XML file can be parsed by tools that generate reports or maintain a vulnerability database.

5.1 Sample STIG XML record for *telnet-server*

Excerpts from the STIG XML record for *telnet-server* illustrate how the STIG can be used in tools to report and fix vulnerabilities. From Listing 4, the *title* identifies the item to be detected and the *rule* specifies the severity.

Listing 4: STIG record *title* and *rule*

```
<title>
  The Red Hat Enterprise Linux operating system must not
  have the telnet-server package installed.
</title>
<Rule id="SV-86701r2\_rule"
      severity="high"
      weight="10.0">
```

The STIG record specifies how to detect the item and how to remedy it in the *check-content* and *fixtext* elements (Listing 5).

Listing 5: STIG record *check-content* and *fixtext*

```
<check-content>
    o o o
    Check to see if the telnet-server package is
    installed with the following command:
        # yum list installed telnet-server
    If the telnet-server package is installed,
    this is a finding.
</check-content>

<fixtext fixref="F-78429r1\_fix">
    Configure the operating system to
    disable non-essential capabilities by removing the
    telnet-server package from the system with the
    following command:
    # yum remove telnet-server
</fixtext>
```

A different government agency, National Institute of Standards and Technology (NIST), re-publishes the DISA STIG in a slightly different XML format called Security Content Automation Protocol (SCAP) that provides a better interface for security tools that check systems and generate reports or even automate the remedies. The most popular open-source SCAP tool is OpenSCAP and the most used SCAP tool in the defense community is the SCAP Compliance Checker (SCC) from Space and Naval Warfare Systems Center (SPAWAR).

5.2 Hardening with home-grown bash scripts in K2

On KRFS K2, the SPAWAR SCC tool is installed onto an SBC card running an initial Linux image released from KRFS. The SCC tool is invoked to produce a report listing any vulnerabilities and assigning a score to the Linux image on that card. KRFS source code includes bash scripts whose content is extracted from the *fixtext* of the STIG or SCAP files, and, these bash scripts are run on the SBC card to execute the STIG-prescribed remedies. The SPAWAR SCC tool is run again to produce another report and score, and, this process repeats until an acceptable security posture is achieved. This hardened Linux filesystem image is then captured and becomes the new master Linux image.

5.3 Open-source DevSecOps instead of home-grown tools in K3

For the Linux image produced for K3, instead of using the home-grown bash scripts to apply the STIG remedies, a DevSecOps tool (Reference 9) based on Ansible[9] (now part of Red Hat) will be used to automatically harden the operating system as part of the DevSecOps build-deploy process.

[9]https://www.ansible.com/

6 Costs, readiness, and opportunities

6.1 Development and maintenance license costs

Table 4 summarizes license costs (see References 10 and 11).

Table 4: License costs

	Red Hawk	Red Hat RT
Product Maintenance (yearly)	281,000	0
Development License (yearly)	20,796	2,700
Run-time License (perpetual, per-instance)	300	0
Run-time License (yearly, per-instance)	400	2,700

6.2 Technology readiness

6.2.1 Limited source code changes required to migrate K3 to Red Hat RT

Because so much of the Concurrent RedHawk product is derived from the RT Patch and none of the RedHawk proprietary features are used (Section 3), the only source code changes are to use the standard Linux syntax for CPU affinity instead of the SuSE/RedHawk shield syntax (Section 3.1). Similarly, the build method will change to the more conventional open-source Red Hat tools (kernel development packages and LiveMedia Creator of Section 4.3).

6.2.2 Development cost to switch DevSecOps build to RedHawk

The DevSecOps process planned for K3 is based on Red Hat RT on a similar XES SBC as described in Section 4.4 and Figure 8. To switch the DevSecOps build to RedHawk requires development of a new platform and process as depicted in Figure 9.

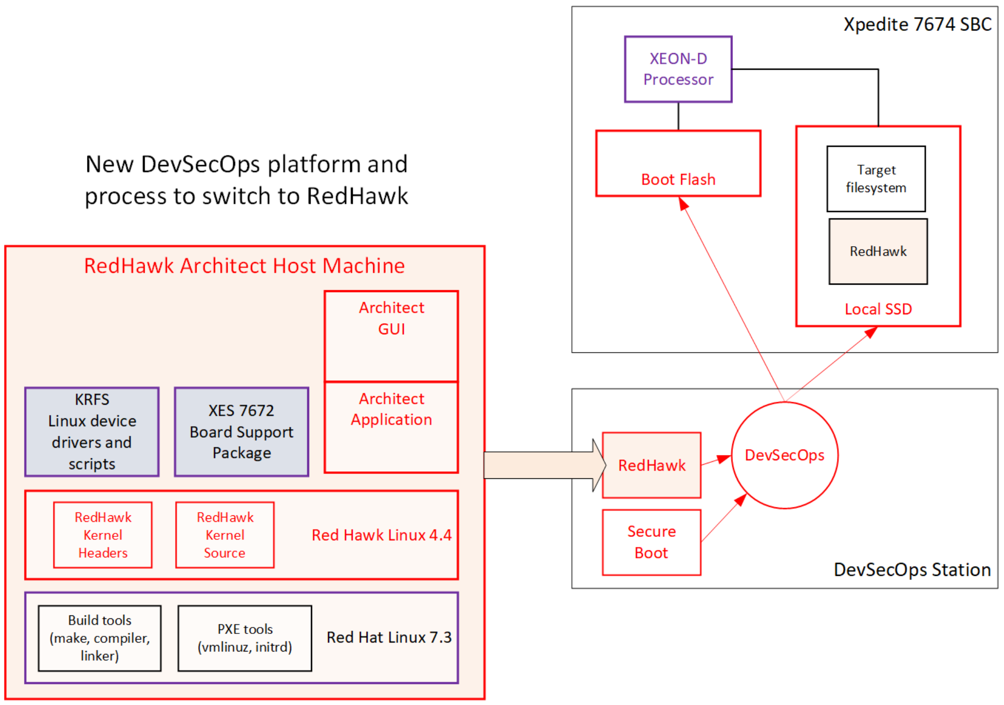


Figure 9: New DevSecOps platform and process required to switch to RedHawk

6.3 Leveraging other Raytheon and open-source resources

6.3.1 Reusing Raytheon DevSecOps with Red Hat RT on XES SBC

The DevSecOps infrastructure from another Raytheon group is based on Red Hat RT running on a similar XES card. This infrastructure can be leveraged by KRFS when running Red Hat RT on the KRFS SBC.

6.3.2 Leverage conventional builds for developers and CI/CD

By eliminating the proprietary RedHawk Architect build requirements, individual developers can turn through their build/test/deploy cycles more frequently and using conventional build system can be integrated into CI/CD to enable frequent and rapid deployment.

6.3.3 DevSecOps with open-source Ansible and STIG Playbook

The Raytheon DevSecOps infrastructure includes hardening tools based on Ansible and an open source Ansible STIG playbook whose future development and ongoing maintenance by the open-source community can be leveraged by KRFS.

6.3.4 Red Hat LiveMedia Creator

The Red Hat LiveMedia Creator can be used to create PXE-boot images and DVD installation images directly from source code by any developer or CI/CD machine. The Raytheon

DevSecOps includes tools and a process for using LiveMedia Creator that can be leveraged for KRFS.

7 Appendix A: July 2019 Merge of RT Patch to Linux mainline

The git commit message from Linus Torvalds merging the real-time patch into the Linux mainline on July 20, 2019 is below (Reference 12).

```
Pull CONFIG_PREEMPT_RT stub config from Thomas Gleixner:
 "The real-time preemption patch set exists for almost 15 years now and
  while the vast majority of infrastructure and enhancements have found
  their way into the mainline kernel, the final integration of RT is
  still missing.

  Over the course of the last few years, we have worked on reducing the
  intrusivenness of the RT patches by refactoring kernel infrastructure
  to be more real-time friendly. Almost all of these changes were
  benefitial to the mainline kernel on their own, so there was no
  objection to integrate them.

  Though except for the still ongoing printk refactoring, the remaining
  changes which are required to make RT a first class mainline citizen
  are not longer arguable as immediately beneficial for the mainline
  kernel. Most of them are either reordering code flows or adding RT
  specific functionality.

  But this now has hit a wall and turned into a classic hen and egg
  problem:

     Maintainers are rightfully wary vs. these changes as they make only
     sense if the final integration of RT into the mainline kernel takes
     place.

  Adding CONFIG_PREEMPT_RT aims to solve this as a clear sign that RT
  will be fully integrated into the mainline kernel. The final
  integration of the missing bits and pieces will be of course done with
  the same careful approach as we have used in the past.

  While I'm aware that you are not entirely enthusiastic about that, I
  think that RT should receive the same treatment as any other widely
  used out of tree functionality, which we have accepted into mainline
  over the years.

  RT has become the de-facto standard real-time enhancement and is
  shipped by enterprise, embedded and community distros. It's in use
  throughout a wide range of industries: telecommunications, industrial
  automation, professional audio, medical devices, data acquisition,
```

automotive - just to name a few major use cases.

RT development is backed by a Linuxfoundation project which is
supported by major stakeholders of this technology. The funding will
continue over the actual inclusion into mainline to make sure that the
functionality is neither introducing regressions, regressing itself,
nor becomes subject to bitrot. There is also a lifely user community
around RT as well, so contrary to the grim situation 5 years ago, it's
a healthy project.

As RT is still a good vehicle to exercise rarely used code paths and
to detect hard to trigger issues, you could at least view it as a QA
tool if nothing else"

8 Appendix B: STIG sample for telnet-server

The STIG record for *telnet-server* is excerpted from Reference 8.

Listing 6: STIG XML sample

```
<Group id="V-72077">
    <title>SRG-OS-000095-GPOS-00049</title>
    <description>&lt;GroupDescription&gt;&lt;/GroupDescription&gt;</description>
    <Rule id="SV-86701r2\_rule" severity="high" weight="10.0">
        <version>RHEL-07-021710</version>
        <title>The Red Hat Enterprise Linux operating system must not have the
telnet-server package installed.</title>
        <description> ooo </description>
        <reference>
            <dc:title>DPMS Target Red Hat 7</dc:title>
            <dc:publisher>DISA</dc:publisher>
            <dc:type>DPMS Target</dc:type>
            <dc:subject>Red Hat 7</dc:subject>
            <dc:identifier>2777</dc:identifier>
        </reference>
        <ident system="http://iase.disa.mil/cci">CCI-000381</ident>
        <fixtext fixref="F-78429r1\_fix">Configure the operating system to
disable non-essential capabilities by removing the telnet-server package from
the system with the following command:

# yum remove telnet-server</fixtext>
        <fix id="F-78429r1\_fix" />
        <check system="C-72309r2\_chk">
            <check-content-ref name="M"
                href="DPMS\_XCCDF\_Benchmark\_RHEL\_7\_STIG.xml" />
            <check-content>Verify the operating system is configured to disable
non-essential capabilities. The most secure way of ensuring a non-essential
capability is disabled is to not have the capability installed.

The telnet service provides an unencrypted remote access service that does not
provide for the confidentiality and integrity of user passwords or the remote
session.

If a privileged user were to log on using this service, the privileged user
password could be compromised.

Check to see if the telnet-server package is installed with the following
command:

# yum list installed telnet-server

If the telnet-server package is installed, this is a finding.</check-content>
        </check>
    </Rule>
</Group>
```