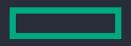
From 1.1.1 to 3.5: Modernizing OpenSSL for Enterprise Networking, FIPS, and Crypto Agility

William Bellingrath, Software Engineer

2026-10-15

Agenda

01	The environment
02	The big jump: 1.1.1 -> 3.0
03	The small hops: 3.0 -> 3.5
04	Where we are
05	Crypto agility: what’s next



Introduction

Who are we?

— Cybersecurity R&D @ HPE, formerly Juniper Networks

— Developers of security infrastructure used throughout Junos OS and Junos OS Evolved

— Responsible for keeping that infrastructure secure, compliant and maintainable

What do we work on?

— Enterprise grade routers, switches and firewalls

— FreeBSD and Linux, across multiple hardware architectures

— OpenSSL, OpenSSH, FIPS and other core security components

What did we do?

— Upgraded Junos OS OpenSSL from 1.1.1 to 3.5 for (almost) all consumers

— Modernized the build, testing, and integration model along the way

— Positioned Junos to adopt future OpenSSL releases rapidly

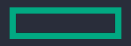
HPE Juniper Networks Portfolio

Junos OS

- FreeBSD-based operating system
- Chassis-level FIPS boundary
- Build targets across FreeBSD, Linux and multiple architectures
- Broad product portfolio, including performance sensitive SRX firewalls
- Firewall PFE remains on OpenSSL 1.1.1 due to performance requirements
- PQC deadlines provide a forcing function for modernization

Junos OS Evolved

- Linux-based operating system
- Modular FIPS boundary
- Linux-focused, limited architecture built with the Yocto Project
- Narrower product portfolio
- Migrated to 3.5 at the time of release
- OpenSSL 3.5 (base and FIPS providers)



“Journey before destination”

Getting to 3.0.x

3.0.x upgrade methodology

Complexity in Junos OS – The challenge

OpenSSL's footprint across Junos

— On the order of one hundred consumers

— Compounded across multiple OS and arch – none can break

Compliance requirements

— FIPS, CC, "Cannot break fips mode."

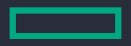
Specialized consumers such as SRX SSL Proxy and SRX PFE

— Not enough to "build and ship" – real world performance goals

Starting approach "Monolithic commit"

— Jump everyone. At once.

— Everyone moves, or nothing moves



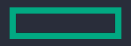
3.0.x upgrade methodology

Complexity in Junos OS - The problems, circa 2023

Legacy build
— FreeBSD 6 artifacts
— Static opensslconf.h with only 3-4 copies covering **everyone**
— Statically generated files

Dependency problems
— ”Header games”
— FreeBSD kernel depends on OpenSSL
— UI CLI relies on the kernel and OpenSSL
— Full Junos build relies on everything

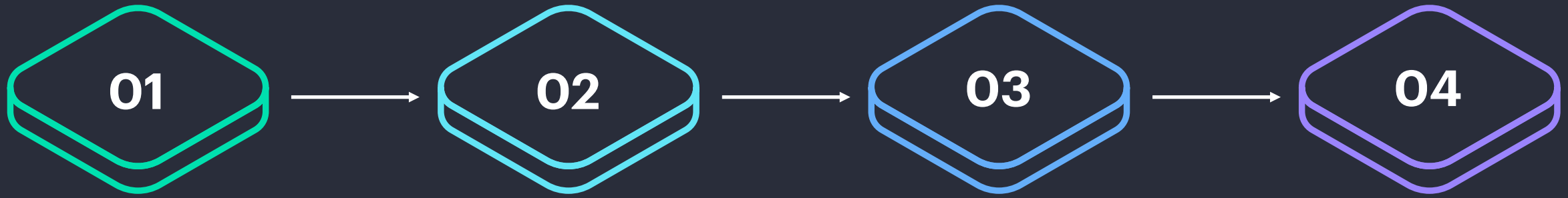
Upgrade friction
— -Werror means the build breaks on deprecation warnings
— Swapping library version from .so.111 to .so.30 means packaging and pointer churn for each test build



3.0.x upgrade methodology

Complexity in Junos OS

We need to dynamically generate assembly...



Static assembly

Static script to generate assembly. Run manually and committed.

Why can't we do this dynamically like everyone else?

Dynamic Configure

Now we have a dynamic configuration.h that *actually* matches our OS and Arch.

Dynamic headerfile generation

With our matching configuration we can generate headers per OS/Arch as well

Dynamic assembly generation!

Supported platforms now get assembly dynamically built from source

Build OpenSSL like OpenSSL builds OpenSSL

3.0.x upgrade methodology

“Header games”: whack-a-mole

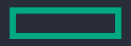
Prolific internal header usage

— Some is excusable, some is due to laziness

Custom sys/ headers for some platforms

— Build breaks from a system header brought in by OpenSSL
  - #define OPENSSL_NO_FOO to prevent that header from getting pulled in

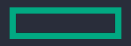
```
...
#include <openssl/foo.h>
/* pthread.h breaks the build here */
#define OPENSSL_NO_THREADS
#include <openssl/bar.h>
...
```



3.0.x upgrade methodology

“Header games”: whack-a-mole

```
...
/* Conflicts with our header definitions.
 * #define these so we don’t pull in sys/time.h or
 * sys/types.h */
#ifndef _SYS_TIME_H_
#define _SYS_TIME_H_
#endif
#ifndef _SYS_TYPES_H_
#define _SYS_TYPES_H_
#include <openssl/foo.h>
...
```



3.0.x upgrade methodology

Complexity in Junos OS – Align with upstream!

Reworked Makefiles and our Configure to accept select ‘no-foo’ options based on build variables

— In OpenSSL 1.1.1 we could remove the crypto/foo directory from build – with the provider layer this doesn’t work

Complete audit of all code delta to upstream

— “Why do we need this” -> carrying legacy changes against 0.9.8

— “Does anyone still need this” -> carrying legacy SSL ciphers by default

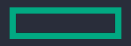
— Downstream delta reduced from ~2,000 lines in 1.1.1 to ~1,000 lines in 3.0

— The goal isn’t zero delta, it’s zero unexplained delta

TODOs

— We keep our FIPS changes for now (PCTs, KATs, zeroization)

— We plan to shave more from the upstream delta with each release



3.0.x upgrade methodology

Complexity in Junos OS – The 11th hour

SRX PFE cannot take the performance hit, we cannot make the monolithic jump

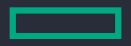
Bifurcate OpenSSL and maintain OpenSSL 1.1.1 and 3.0

— For performance, stay on 1.1.1 extended support

Why didn’t we do this all along?

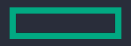
— Previous upgrade (1.0.2 -> 1.1.1) method was piecemeal

— Maintain dual libraries and migrate consumers over



OpenSSL 3.0 -> 3.5 sprint

Can we make upgrades boring?



OpenSSL 3.0 -> 3.5 sprint

Managing risk – Never repeat the 3.0 upgrade

Monolithic jump?

— This is what we just tried and took 2 years (not unlike 1.0.2 -> 1.1.1)

— Any problems? Stuck on 3.0.x

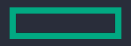
Small bites

— 3.0 -> 3.1 -> 3.2 -> 3.3 -> 3.4 -> 3.5

— Newfound Agility, “I want to go fast”

OpenSSL 3.1 Proof-of-concept

— Upgraded, committed and pushed to everyone in ~1 week



OpenSSL 3.0 -> 3.5 sprint

Managing risk – Increase confidence in upgrades

Validate OpenSSL on the actual Junos image

Problems we face

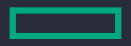
— No Perl on-box
— Junos hooks and features prevent a simple ‘make test’
— Currently build the test utilities and run them on-box manually

Improvements made

— “Why are we doing it this way” -> build libtestutil.a; “Follow upstream” and build all test utilities

Future automation?

— On-box can’t ‘make test’ but host builds can
— Upstream perl tests need build files to match source tree? (we don’t build libdefault, libcommon)



OpenSSL 3.0 -> 3.5 sprint

Managing risk – Pain points

Clean up and check FIPS at every crank!

OpenSSL 3.2

Compatibility

- SSL_CONNECTION
- Internal structure access
- Threading/header problems
- QUIC disabled for ease of upgrade

OpenSSL 3.3

Diff noise, difficult to review

- Upstream reformat
- Deliberately step from 3.3.5 to 3.3.6

OpenSSL 3.4

Staging the focus on 3.5

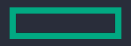
- Minimize the disruption
- Isolate remaining 3.5 delivery and PQC
- Overhaul on-box unittests

OpenSSL 3.5

Boring

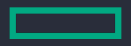
- ~1 week
- Added PQC tests

After an intense 3 months, boring is good.



Where are we now?

So... did it work?



Where are we now?

I’m tired boss

Migrating SRX performance from 1.1.1 to 3.5 (SRX 4700)

— High-end data center SRX

— TLS 1.2 ~11% degradation

— TLS 1.3 ~15-37% degradation

— Algorithm fetching our likely culprit

- Patches pulled from 3.6, 4.0 and 4.1 put us at +/- 5% of 1.1.1

— Guess who has a busy few months ahead?

Remaining technical debt and open questions

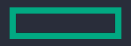
— FIPS alignment

- Can we use the upstream KATs?
- Migrating off our custom PCT, zeroization, key checks
- Can we use the FIPS provider in a chassis FIPS boundary?

— Actual Cleanup - “We’ll do this bit after the big push”

- Down to ~700 lines of known features against upstream
- Found changes we meant to upstream
- Purging internal headers and improper macro usage
- Enable QUIC? Unit test improvements?

— Align with upstream



Where are we going?

I’m tired boss – Chasing upstream? But we *just* got to 3.5.

Performance gains in 4.x and the next LTS drive us onwards and upwards

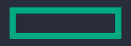
Can we get a test build with 4.0? Major version jumps are hard

- ~2 years 1.1.1 -> 3.0
- 1 day to build all crypto components against OpenSSL 4.0
- 2 days to begin building against all of Junos OS

New upstream release cadence

- Agility is now mandatory
- Improve experimental ways to build against new major version cranks
- The goal wasn’t OpenSSL 3.5, the goal is making the next upgrade easier

This is our newfound crypto agility in support of an older system



Thank You

william.bellingrath@hpe.com

© 2026 Hewlett Packard Enterprise Development LP