Host boot ROM code vulnerability

Description of vulnerabilities and mitigation strategies

Table of Contents

1. Introduction 2
2. Relevant use cases for UFS Boot feature 2
a. Platform Booting from Boot WKLU 2
b. Boot LUN Data Protection –..... 3
3. Description of vulnerability 6
4. Proposed mitigations 7
a. Option #1 7
b. Option #2 7
c. Option #3 8
5. References 9

1. Introduction

Boot Well Known Logical Unit (Boot WKLU) is a UFS feature, first introduced to UFS standard in UFSv1.1. The feature still exists and is still widely used in the industry through UFS v4.0 standard supporting products.

2. Relevant use cases for UFS Boot feature

a. Platform Booting from Boot WKLU

The feature allows the host platform to download the system boot loader from an external non-volatile device by reading the Boot WKLU data at the system startup to access the host SoC boot code.

The Boot Well Known Logical Unit has a pre-defined, fixed by the standard number, W-LUN (30h) through which the host may read the boot code. Boot Well Known Logical Unit is a virtual reference to the actual logical unit containing the boot code, as designated by the host through bBootLunEn attribute.

Depicted in Figure a-1 is a platform with ROM code accessing the UFS device Boot WKLU to read the platform boot code. In this diagram, as the host is reading from the virtual reference W-LUN (30h), the device provides the data stored in LU A as the bBootLunEn attribute references LU A. In case the attribute would reference LU B, the device would provide data stored in Boot LU B.

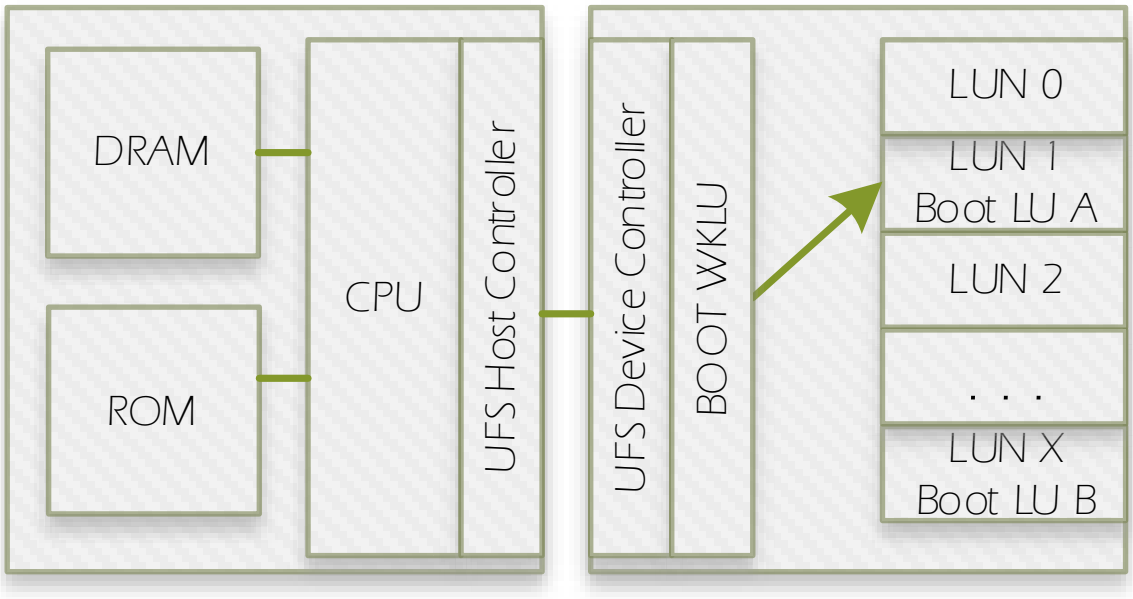

Figure a-1: UFS System Diagram

Depicted in Figure a-2 is a ROM code boot flow to access the UFS device Boot WKLU to read the platform boot code. In this diagram, before the ROM code access the Boot WKLU, it does not check that the boot feature is enabled.

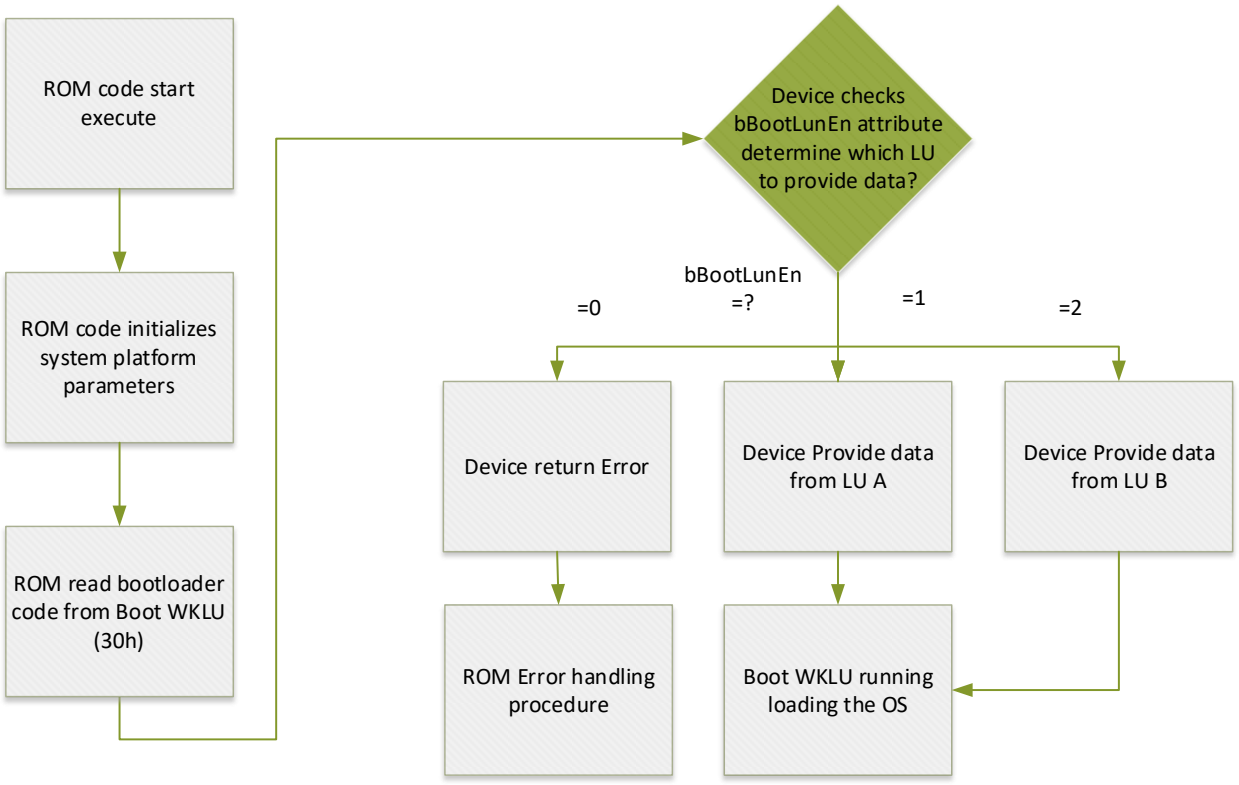

Figure a-2: Naïve Boot flow

b. Boot LUN Data Protection –

Data stored in the boot LUN (LUN A or LUN B) may be configured by the host to be protected by the UFS device in different ways: Permanent Write Protected, Power-on Write Protected or Secure Write Protected.

When the boot LU data has a write Protection mechanism enabled, normal-world applications cannot modify the data in the Boot LU.

For example, depicted in Figure b-1 are boot LUNs which are write protected by the device through permanent WP mechanism, which prevents any change to the content of the boot LU.

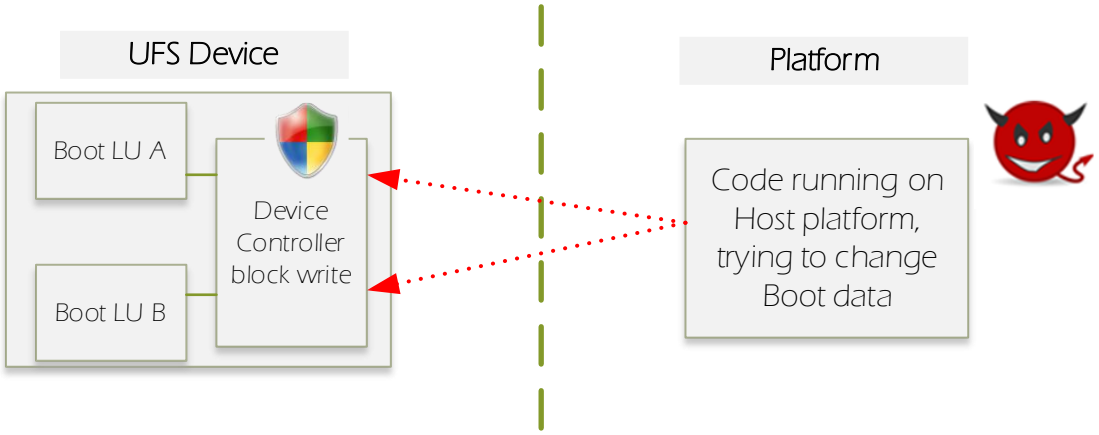

Figure b-1:Device Permanent Write Protection boot LUNs

Other types of Write Protection of the Boot LUN also exist, which would protect the LUN from being changed only by entities authorized by the TEE (through RPMB, secure WP) or by entities after the boot code is read (through Power-on WP).

Depicted in Figure b-2 are boot LUNs which are protected using the Power-on WP mechanism. In case the host platform is not initiating a BOOT LU update, it sets the fPowerOnWPEn flag to block any update to the Boot LUNs (which are configured with bLUWriteProtect = 1).

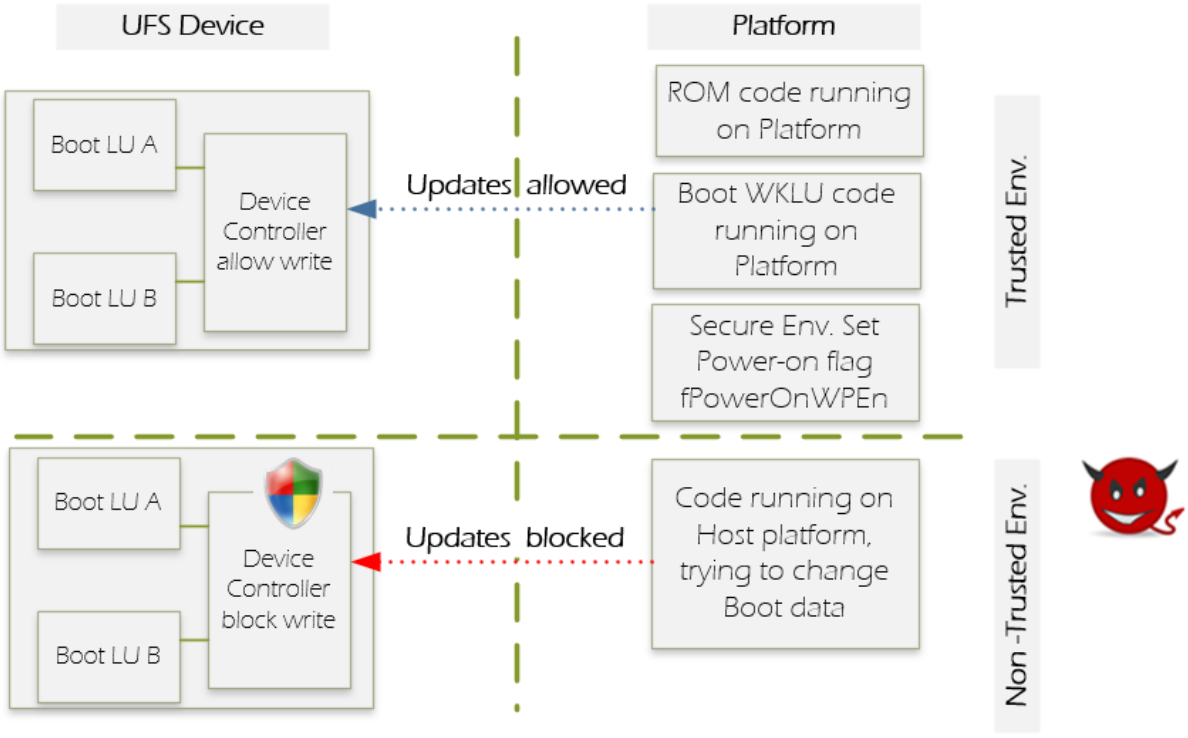

Figure b-2 Device Power-on Write Protection boot LUNs

Depicted in Figure b-3 are boot LUNs which are protected using the Secure WP mechanism. When the host is updating the Boot LU from a secure environment, it sets the RPMB Configuration block to allow writing to the Boot LUNs and when finish updating the LUNs it close back the Boot LUN for Write using the Secure WP mechanism.

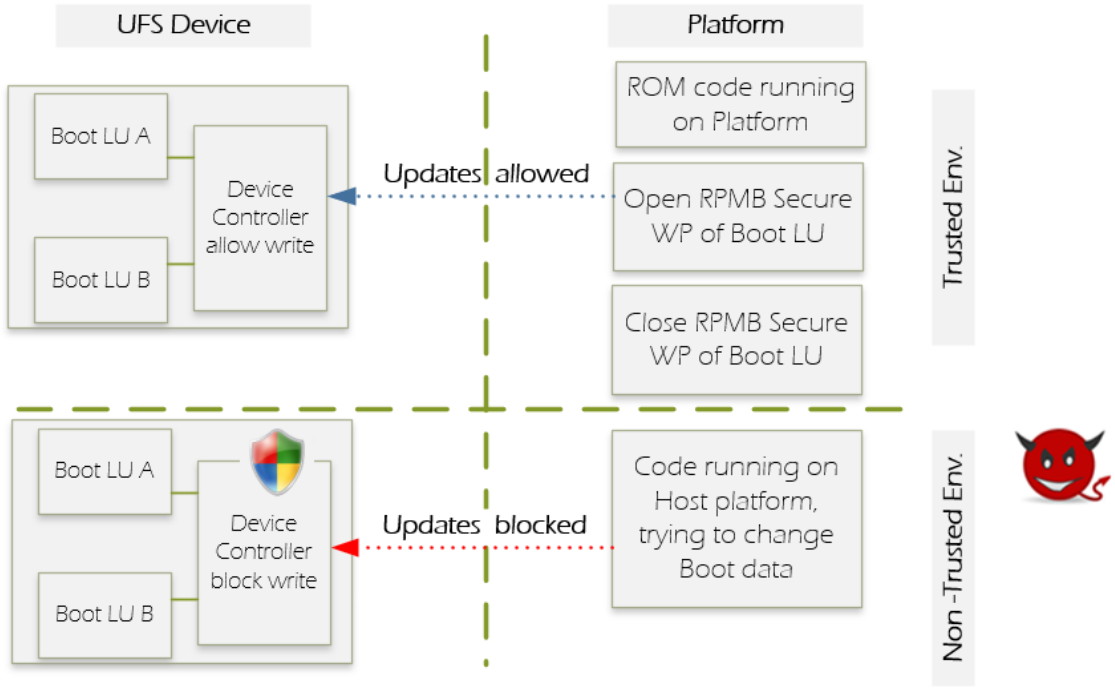

Figure b-3 Device Secure WP for boot LUNs

3. Description of vulnerability

The vulnerability identified may take advantage of a naïve ROM boot code implementation to allow an adversary to make a host platform unusable by failing to complete its boot process.

The adversary may disable the UFS Boot feature by setting bBootLunEn attribute to 0x0.

Depicted in Figure 3-1 is an event flow which describes how an adversary can disable the normal boot flow. As a result, the host boot ROM code is not able to handle the unexpected situation and does not complete the platform boot process.

*Another variant of the vulnerability is that the adversary sets bBootLunEn to the alternate Boot LU causing the host platform to boot with a potentially old version of the boot code (Downgrade attack).

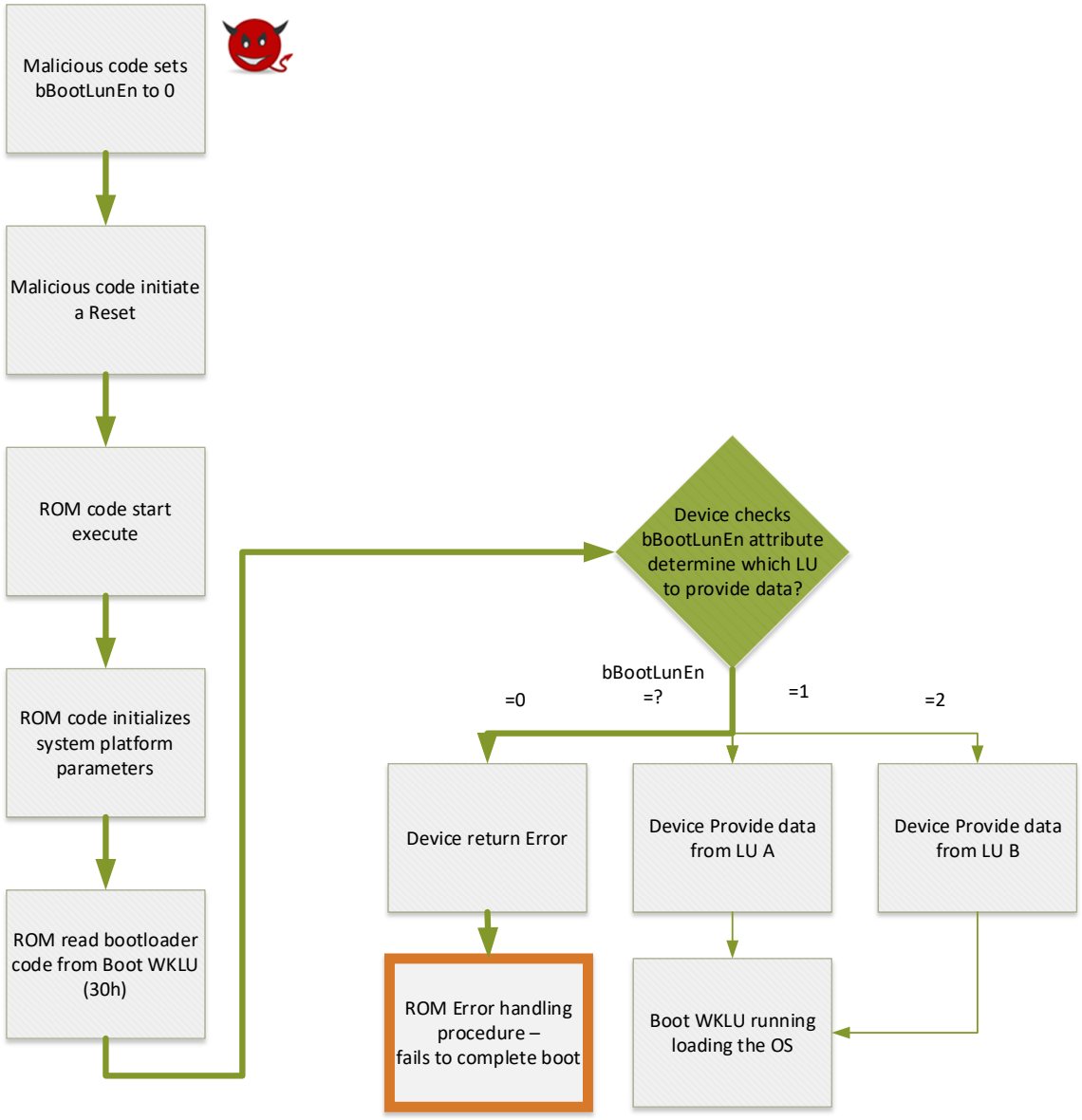

Figure 3-1:Exploiting a naïve ROM boot code

4. Proposed mitigations

a. Option #1

If the ROM code is not able to complete the boot process and enter the Error handling procedure as described in Figure 3-1, it should enable the bBootLunEn to either LUN A or LUN B and try to read the Boot WKLU

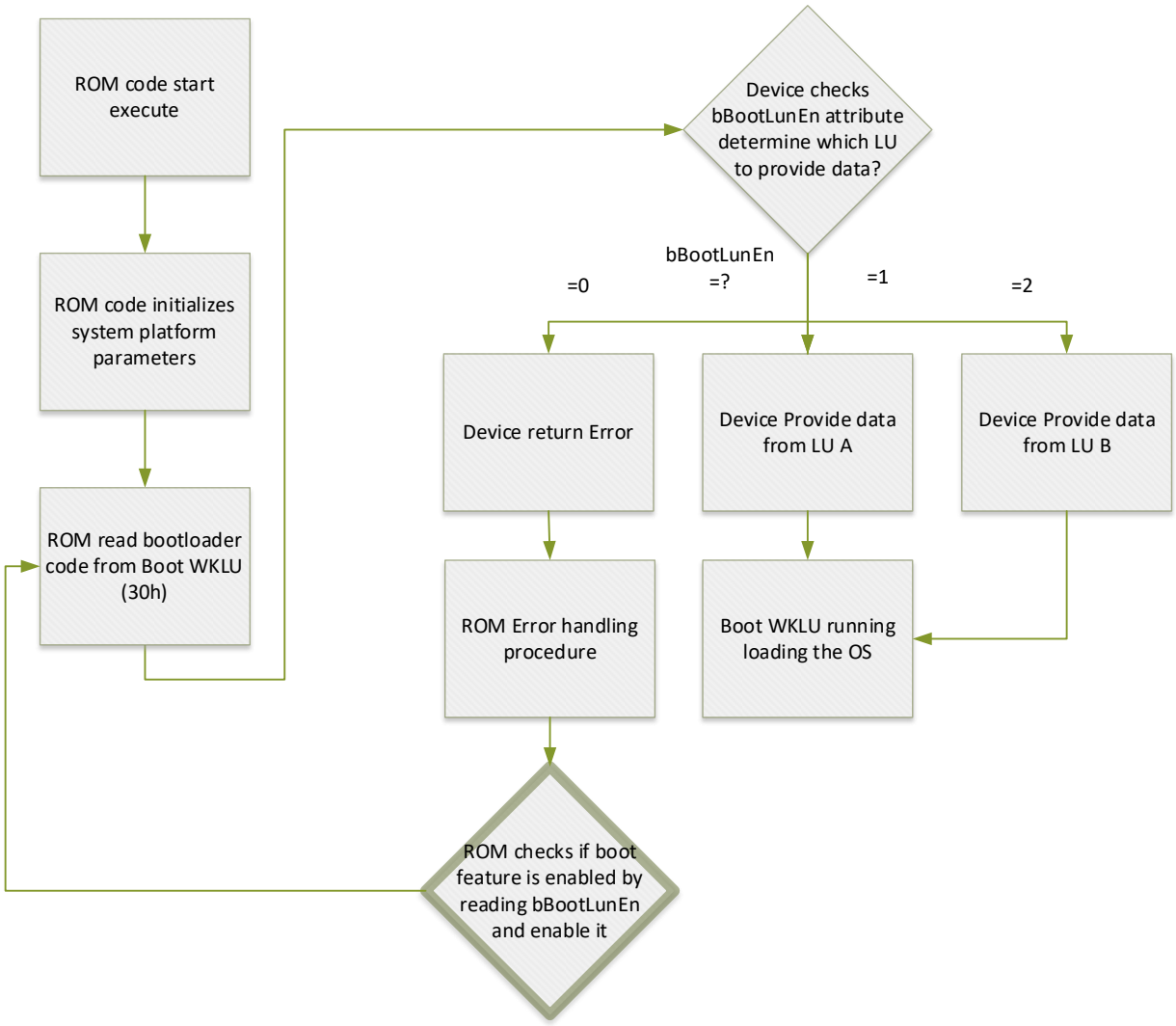

Figure a-1: Host boot ROM code implementation checking boot feature is enabled

b. Option #2

Before the ROM code starts reading the Boot WKLU, it sets bBootLunEn to a valid LUN (either LUN A or LUN B) from which the boot data is available.

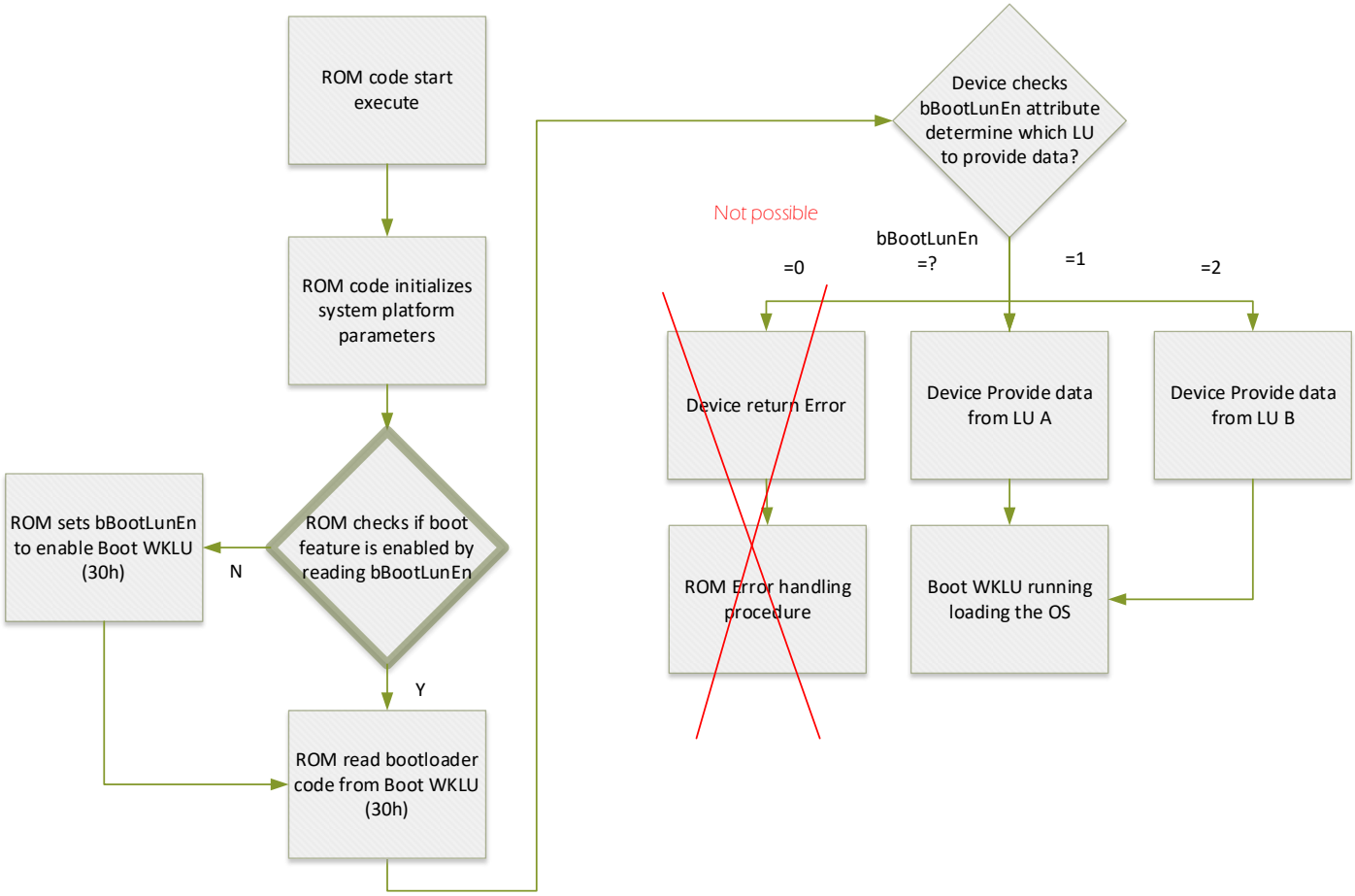

Figure b-1 Host boot ROM code implementation checking boot feature is enabled following a faliure to boot

c. Option #3

ROM code may access the boot LU directly and not through the WKLU mechanism. This could be achieved in two methods:

1) Throughout the UFS device configuration process the boot LUNs would be assigned to the LU Number which the ROM code access to and by that the ROM code would avoid using the bBootLunEn attribute.
2) Alternatively, the ROM boot code may scan the Unit Descriptors and identify the LU Numbers which has bBootLunID sets to A or B, avoiding the use of bBootLunEn attribute.

Other options also exist, depending on host implementation.

5. References

JESD220E - Universal Flash Storage (UFS) Version 3.1, JANUARY 2020