z/OS Validated Boot

Date: May 17, 2023

Kevin McKenzie
Validated Boot Test Architect
kmckenzi@us.ibm.com

The current version of this document can be found at https://ibm.biz/zosValidatedBoot

What is Validated Boot?

Supply chain attacks are cybersecurity threats that do not directly attack a system, but instead tamper with the system during the build and maintenance of the system software. These build systems are often easier to exploit than production systems, and can inject software vulnerabilities that are far harder to detect. The NIAP Protection Profile for General Purpose Operating Systems is intended to address supply chain risks, among other things, and requires that as an operating system starts, it is possible for the system to validate that it has not been altered by a threat actor.

With z/OS Validated Boot, IBM is providing the capability for clients to sign parts of the operating system with a private key. The corresponding public certificate is then distributed via the HMC to supported CPCs, and associated with one or more LPARs. A new IPL type, the List Directed IPL (LD IPL) was created to support Validated Boot. During IPL, a new bootloader will validate that the IPL text used to start the system has been signed by one of the designated private keys, and as the system starts, each part of the system will validate that the next part of the system being loaded has also been signed, up through and including modules loaded into LPA.

Validated Boot Overview

Before going into any detail, we want to go through the expected high level behavior of Validated Boot. First, Validated Boot should be transparent to a non-Validated Boot system. If you have a test system or something else that won't be brought up in Validated Boot mode, or not all of your hardware supports Validated Boot, you don't need to do anything, including installing toleration service. Those systems won't notice any changes.

Next, there are essentially two types of systems involved with Validated Boot: there's the system that you are enabling for Validated Boot, and there's the system that is doing the building. The system that is doing the building is known as the driving system, and the system image that is being built is known as the target system.

While we have designated one set of PTFs as for the driving system, and another set as the full solution, you need to have the full set of PTFs installed on the target system driver before you begin signing anything. You can do the RACF setup without any of the required PTFs, but until the full set of PTFs is installed on the target system, you will not be able to build a system capable of Validated Boot. Until you have the all the z/OS PTFs and the appropriate firmware driver installed on your CPC, you will not be able to perform a Validated Boot.

The building system, which is also referred to as the clean room system, is used to build or make any necessary changes to the target operating system image. Once all changes are made, the clean room system signs the operating system image, and the target system image is then

copied from the clean room to the external world via pack to pack copies. The private keys are only accessible on the clean room system, and changes to the protected operating system's protected modules are only made on the clean room system.

There are a huge number of ways that changes can be made to data on DASD. The expectation is that any time a change is made to a signed load module, that load module will need to be resigned. You may or may not get a warning message saying that a signature has been changed, or dropped, or something else, but you need to assume that any time a signed load module is altered, it will need to be re-signed. Similarly, anything that alters signed IPL text will require the IPL text to be regenerated and re-signed. Even something like zapping a load module and then undoing the zap can invalidate the signature, depending on what data is touched.

Validated Boot is referred to as Secure Boot on the HMC. When IPLing with Validated Boot, you can come up in one of two modes, Audit Mode or Enforce Mode. In Audit Mode, the system will keep track of any exceptions during IPL, and after the IPL is complete, you can use the reporting tool to get a list of exceptions. In Enforce Mode, any exception to a validation request will result in the system wait stating.

Setting up Validated Boot

There are a number of steps that need to be performed to set up and IPL a Validated Boot system. In the chart below, each column may be done separately until they intersect in a joined cell.

Driving System Steps	**Target System Steps**	**Target Hardware Steps**
RACF Setup		Make sure you have VFM installed
Install driving system PTFs	Configure SCM on z/OS	Install the hardware support
	Identify all modules in LPA and Nucleus that need to be signed	Import the Signing Certificate to the HMC
	Install Target System PTFs	
Sign the Target System		
Deploy the Target System		
LD-IPL in Validated Boot Mode		

Planning

Before getting started, you will need to start thinking about what your signing infrastructure should look like, and what you're trying to protect against. Questions to ask include

1. Do you want to use a self-signed certificate, or a certificate from an external certificate authority?
2. Where do you want the private key to live?
3. Do you want to have one signing key per person, per driver, or something else?
4. How will you handle key distribution?
5. Are there any shared resources (load libraries, etc) that need to be protected?
6. How will you handle certificate revocation?
7. How concerned are you about the risk of data corruption during signing?

Samples

You can copy and paste samples from this white paper; however, be very careful about quotation marks, as Microsoft Word keeps enabling “smart quotes.”

Driving System Setup

Clean Room

The system used for signing should be considered a clean room environment, so it should not be one used for general production, test, or sand box purposes. It should few defined users and be as isolated from the rest of your environment as possible. You should only have the software installed and running there that is required for signing and copying the driver (including all the Validated Boot support).

Some of the clean room setup can be done prior to any PTFs being installed. In order to create signed IPL text and to sign load modules, the Validated Boot PTFs need to be installed on the clean room system.

Prereqs

Validated Boot is part of the z16 GA1.5 deliverable, so all associated PTFs will also be included in the IBM.Device.Server.z16-3931.Exploitation FIXCAT. There is also a FIXCAT that is specific to Validated Boot Functionality, IBM.Function.ValidatedBoot, and the PTFs required on the driving system will also be included in the already existing IBM.DrivingSystem-RequiredService FIXCAT. Validated Boot is only supported on z/OS V2R5 and above; lower level systems will tolerate the signatures that are added for validated boot, but not take advantage of it, nor be used as a driving system to create a Validated Boot enabled system.

RACF setup

Even before you install the prereqs for Validated Boot, you can do all the RACF setup required. The general setup only needs to be done once per userid; however, certificate management may need to be done more frequently than that, depending on how often you need to renew certificates.

The RACF signing support is documented here: https://www.ibm.com/docs/en/zos/2.5.0?topic=verification-ipl-data-signing-validated-boot-zos. Note that while the setup is similar, you cannot use the same certificates for signing programs and for validated boot.

In this paper, we will give the TSO commands to do the RACF setup, but other methods exist. To issue these commands, you will need to have the appropriate RACF authorities. Some commands may not be required, depending on your userid's settings. You also need authority to update the load libraries that you are signing, and to write IPL text to the target DASD. This is the same authority you'd need to build or maintain a system image.

In the example below, userid XYZBLD is going to be doing the signing. User XYZBLD will be creating a self-signed certificate, owned by XYZBLD, and only to be used by user XYZBLD. It is also possible to use certificates provided by a certificate authority, and to share signing key rings between users and groups, but that won't be covered in this white paper. We're assuming that the required ICSF PKDS infrastructure has already been set up. If you do not have crypto cards installed, you can still use Validated Boot, but you will have to store the certificates in the RACF database, not the PKDS/TKDS, and adjust the commands below appropriately.

Make sure, if you're issuing the commands manually, to use upper case characters everywhere when creating or specifying the name of the key ring. Some RACF commands will automatically upper case everything, while others preserve case.

The steps involved in the RACF setup are:

1. Create a code-signing certificate

   RACDCERT ID(XYZBLD) GENCERT SUBJECTSDN(CN('XYZ Code Signing') O('XYZ Corp') C('US')) SIZE(521) NISTECC(PKDS) WITHLABEL('XYZ CODE SIGNING') KEYUSAGE(HANDSHAKE DOCSIGN) NOTAFTER(DATE(2023-12-31))

   This will generate a new certificate, along with a new public/private key pair. The private key will be stored in the ICSF PKDS, secured by the master key of your cryptographic processor. You can certainly add more information to the SUBJECTSDN options if you want, and should follow your company/installation guidelines to choose an expiration date.

2. Define a key ring to use for signing the load modules.

   RACDCERT ID(XYZBLD) ADDRING(VALIDATED.BOOT.SIGNING.KEYRING)

   This will create a new key ring for user XYZBLD, VALIDATED.BOOT.SIGNING.KEYRING.

3. Add the certificate(s) to the keyring

   RACDCERT ID(XYZBLD) CONNECT(RING(VALIDATED.BOOT.SIGNING.KEYRING) LABEL('XYZ CODE SIGNING') DEFAULT)

   This will connect certificate labeled 'XYZ CODE SIGNING' to keyring VALIDATED.BOOT.SIGNING.KEYRING, and makes it the default certificate for the keyring.

4. Permit the user to use the keyring

   RDEFINE RDATALIB XYZBLD.VALIDATED.BOOT.SIGNING.KEYRING.LST

   This defines a new profile in the RDATALIB class called XYZBLD.VALIDATED.BOOT.SIGNING.KEYRING.LST, which controls READ access to the XYZBLD.VALIDATED.BOOT.SIGNING.KEYRING.

   PE XYZBLD.VALIDATED.BOOT.SIGNING.KEYRING.LST CLASS(RDATALIB) ID(XYZBLD) ACC(READ)

   This gives userid XYZBLD READ access to profile XYZBLD.VALIDATED.BOOT.SIGNING.KEYRING.LST in class RDATALIB.

5. Tell RACF to use this keyring and certificate for validated boot signing purposes for userid XYZBLD.

   RDEFINE FACILITY IRR.PROGRAM.V2.SIGNING.XYZBLD APPLDATA('SHA512 XYZBLD/ VALIDATED.BOOT.SIGNING.KEYRING')

   This defines FACILITY class profile IRR.PROGRAM.V2.SIGNING.XYZBLD, and puts the string 'SHA512 XYZBLD/ VALIDATED.BOOT.SIGNING.KEYRING' into the application data associated with the class. The APPLDATA string identifies to RACF the hashing algorithm, keyring owner, and keyring being used for validated boot signing.

6. Issue a RACF refresh command for the FACILITY, RDATALIB, and DIGTCERT classes.

   SETROPTS RACLIST(FACILITY RDATALIB DIGTCERT) REFRESH

7. (Optional) Generate and save somewhere safe summary information about the certificate

   RACDCERT ID(XYZBLD) LIST(LABEL('XYZ CODE SIGNING'))

   You should get output something like

   Digital certificate information for user XYZBLD:

   Label: XYZ CODE SIGNING
   Certificate ID: 2QfUw9LF1enJ5cJA4omHlYmVh0Dx
   Status: TRUST
   Start Date: 2022/10/05 00:00:00
   End Date: 2023/12/31 23:59:59
   Serial Number:
   >00<
   Issuer's Name:
   >CN= XYZ CODE SIGNING.O=XYZ Corp.C=US<
   Subject's Name:
   >CN=XYZ CODE SIGNING.O=XYZ Corp.C=US<
   Signing Algorithm: sha512ECDSA
   Key Usage: HANDSHAKE, DOCSIGN
   Key Type: NIST ECC
   Key Size: 521
   Private Key: YES
   PKDS Label: IRR.DIGTCERT.XYZBLD. DC35F89221A36A41
   Certificate Fingerprint (SHA256):
   C6:B1:A5:0B:3D:DB:21:B0:EB:48:00:F5:48:DF:86:9F:
   BE:C6:06:08:91:76:5F:6C:14:5B:A2:8A:C9:26:B2:7B
   Ring Associations:
   Ring Owner: XYZBLD
   Ring:
   > VALIDATED.BOOT.SIGNING.KEYRING<

   You will especially want to note the Certificate Fingerprint value.

Certificate Lifecycle Management

After the initial RACF setup is done, whenever a certificate expires, neither the zBootloader nor z/OS will use it for validation purposes. At this point, you have two choices:

1. Renew the certificate with the existing keys, and then export the updated certificate and replace the certificate on the HMC with the updated certificate.
2. Create a new certificate with a new corresponding private key, resign anything that had been signed with the old certificate, and distribute the new certificate to the impacted HMCs.

The HMC will give you warnings about expired certificates before expiration.

Signing

After you've done the RACF setup and created the signing infrastructure, you will then need to sign the components of z/OS: creating signed IPL Text, signing load modules that will end up in the nucleus or LPA, and, optionally, but highly recommended (or required if you use AUTOIPL), creating a signed Stand Alone Dump program. The driving system has to be running z/OS V2R5 or later with at least the PTFs identified with the SMP/E FIXCAT IBM.DrivingSystem-RequiredService installed; you can't STEPLIB to the signing libraries the way you sometimes can when building a new system. You do not need to be running on a z16 in order to sign things.

The target system needs to have all PTFs identified by the FIXCAT IBM.Function.ValidatedBoot installed. Failure to have all the PTFs installed on the target system will leave you with a system unable to come up in Validated Boot Enforce mode.

z/OSMF Portable Software Instance Signing Support

When ordering z/OS V2.5 or higher as a ServerPac through Shopz, the installable package will be a z/OSMF portable software instance. In the shipped package will be three z/OSMF Workflows. After the General Availability of z/OS Validated Boot, one of the z/OSMF Workflows, PostDeploy, will contain support for signing IPL Text, Standalone Dump Text, and appropriate in-scope load modules included in your order.

The signing of the NUCLEUS and LPA-eligible data sets will be customized for your order, and can be modified to add other data sets of your choosing that are appropriate. The JCL for the z/OSMF Workflow load module signing step can also be saved and reused for future re-signing, after PTFs have been installed if you wish.

Creating LD-IPL Text

LD-IPL Text has to be signed; you can't create LD-IPL Text without having the necessary signing infrastructure set up for the userid that will be used for the signing process. You can create LD-IPL Text by adding the LDIPL(STANDARD) option to your already existing ICKDSF IPL Text creation job. For example:

```
//INIT      EXEC  PGM=ICKDSF
//SYSPRINT DD SYSOUT=*
//IVOL     DD DISP=OLD,UNIT=3390,VOL=SER=IOS7R5
//SAMPLIB  DD DSN=SYS1.SAMPLIB(IPLRECS),DISP=SHR,UNIT=SYSALLDA,
//         VOL=SER=IOS7R5
//         DD DSN=SYS1.SAMPLIB(IEAIPL00),DISP=SHR,UNIT=SYSALLDA,
//         VOL=SER=IOS7R5
//SYSIN DD *
  REFORMAT DDNAME(IVOL) VERIFY(IOS7R5) IPLDD(SAMPLIB,OBJ) IPLEXIST -
  LDIPL(STANDARD)
```

When you create signed LD-IPL Text, the IPL text can also be used to IPL in CCW mode.

Creating Signed Stand Alone Dump

In order to sign the Stand Alone Dump program when you are creating it, you only need to add LDIPL=YES to your GENPARMS DD statement. This will write out a Stand Alone Dump program that will be able to be the target of a list directed IPL in Enforce mode.

Signing Load Modules

In order to come up in Validated Boot Enforce mode, the nucleus, along with any load module that is going be loaded into LPA needs to be signed. Some load modules are always loaded by the system; others are specified in parmlib. When signing load modules, you can either sign load modules individually, or sign every load module in a dataset. Either option is supported; however, since there's no harm in signing a load module that isn't loaded into LPA, we'd suggest signing entire datasets, as it will lead to simpler JCL.

When signing, you need to specify a source data set, a target data set, and, optionally, a list of include and exclude masks to determine which load modules to sign. By default, all load modules in a data set will be signed. Only signed load modules will be placed into the target data set. If you have anything in your load library that isn't a load module, or you only want to sign a subset of the load modules, you then need to copy anything that was not signed into the target data set separately. If you specify the same source and target data set, the load modules will be signed in place.

There are space considerations that need to be taken when signing a load module. A signed load module will take up a little bit more space than an unsigned load module, so in the case of a separate source and target data set, you'll need to make sure that the target data set has enough excess space for the signatures when allocating it. However, since some of the datasets that need to be signed will end up in linklist as well as LPA, make sure that the dataset doesn't go into extents. (Best practice is to allocate data sets that will go into linklist with 0 secondary

space.) If you're signing in place, due to the way PDS space allocation works, you may need a great deal of extra space allocated, and after the signing is finished, you'll probably want to compress the data set.

Note that the signing utility is stricter in what it will accept as input than z/OS is; z/OS will tolerate things in load libraries that the signing utility will reject as invalid, such as certain types of alias errors, or members that are not actual load modules. Usually, the signing utility will simply ignore such issues with a warning message.

When signing the nucleus, it is possible that you will have SYSCATxx members. These are text files that were used to designate which system catalog to use, and can sometimes still be used in failure scenarios. If you have these, and still need them, you'll need to copy them over separately.

Instead of treating signing as a one-off process, we suggest integrating it into your build jobs, and just re-sign everything whenever you make a change, or at least before you distribute the driver to other systems. Re-signing an already signed load module won't hurt anything, and making it part of your default build process means there's no risk of forgetting to run the signing process manually.

The signing utility can be used to sign load modules, remove signatures from load modules, and report on the signing status of load modules. The ACTION= parameter controls what action the signing utility will take.

A sample proc for creating a new data set with signed members is as follows:

```
//SIGN      PROC VSERIN=VOLIN,
//           VSEROUT=VOLOUT,
//           DSNAME=MY.DSNAME
//SIGNS EXEC PGM=IEWSIGN,PARM='ACTION=SIGN'
//SYSPRINT DD SYSOUT=*
//INFILE   DD DSN=&DSNAME,DISP=SHR,UNIT=3390,
//         VOL=SER=&VSERIN
//OUTFILE  DD DSN=&DSNAME..SIGNED,UNIT=3390,
//         DISP=(NEW,KEEP),LIKE=&DSNAME,
//         VOL=SER=&VSEROUT,
//         DCB=(RECFM=U,BLKSIZE=32760)
// PEND
```

I'm using a PROC here because you'll need to sign a number of different datasets, and using a PROC reduces the JCL complexity. You pass three symbolics, &VSERIN, &VSEROUT, and &DSNAME when you call the procedure.

In this PROC, a data set called &DSNAME on volume &VSERIN will be taken as input. A new dataset, &DSNAME.SIGNED is created on volume &VSEROUT. It will have the same characteristics as &DSNAME.

A sample proc for signing a data set in place is:

```
//SIGNINP   PROC VOLSER=VOLIN,
//          DSNAME=MY.DSNAME
//SIGNS EXEC PGM=IEWSIGN,PARM='ACTION=SIGN'
//SYSPRINT DD SYSOUT=*
//INFILE   DD DSN=&DSNAME.,DISP=SHR,UNIT=3390,
//         VOL=SER=&VOLSER.
//OUTFILE  DD DSN=&DSNAME.,UNIT=3390,
//         DISP=(OLD,KEEP),VOL=SER=&VOLSER.
// PEND
```

Unsigning Load Modules

If you want to, you can use the signing utility to unsign load modules, that is, to remove the signature data from the dataset containing the member. You do this by specifying parameter "ACTION=UNSIGN," such as in the following:

```
//UNSIGN EXEC PGM=IEWSIGN,PARM='ACTION=UNSIGN'
```

The same space considerations that apply to signing apply to unsigning.

Load Module Reporting

If you want to get information about whether or not a module is signed, you can use the IEWSIGN "ACTION=REPORT." Depending on the "ReportLevel" option you specify, you will get different levels of information about the load module(s) you have specified. For example:

```
//UNSIGN EXEC PGM=IEWSIGN,PARM='ACTION=REPORT'

//UNSIGN EXEC PGM=IEWSIGN,PARM='ACTION=REPORT,REPORTLEVEL=2'

//UNSIGN EXEC PGM=IEWSIGN,PARM='ACTION=REPORT,REPORTLEVEL=3'
```

Note that this only reports whether or not the module is signed; it doesn't give you any information about whether you would be able to successfully IPL in Enforce mode.

Debugging Signing Problems

If there is an error during the signing process, you will get an error message from the IEWSIGN program. You can use this information to determine what the cause of the error is. For example:

```
IEW6016S RACF has not been configured correctly to sign a load module.
         RACF API R_PgmSignVer failed. Function Code=SIGINIT , SAF
return code=8 RACF return code=8, RACF reason code=112.
```

You can then look up this message in the appropriate book. For example, in this case, the z/OS Security Server RACF Callable Services book, in the R_PgmSignVer Return and reason codes section, which tells us "Key ring does not exist or does not contain a default certificate." In this case, I didn't specify the DEFAULT option when connecting the certificate to the key ring. Making the certificate the default option resolved the problem and allowed me to continue signing.

Signing after installing service

During the installation of PTFs, any load modules that are updated will not be re-signed, and will become unsigned. SMP/E will not attempt to preserve the signature, which would be invalid after a rebind anyway, nor will it attempt to sign the updated module itself. The result is that any time service is applied, you should plan on resigning load modules. One way to do this would be to sign in place using the 'ACTION=SIGN,STATE=UNSIGNED' option; this way you don't have to re-sign already signed modules.

Copying the Driver

As mentioned above, once the driver is built, you will need to do a full volume pack to pack copy in order to preserve the signed IPL text and all signed load modules.

```
//COPYVOL  JOB 'D1008P,D10,B7062D36',SYSTEM.TEST,
//          CLASS=A,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//OUTDD OUTPUT DEFAULT=YES,JESDS=ALL,OUTDISP=(HOLD,HOLD)
//*
//      EXEC PGM=ADRDSSU,REGION=4M
//SYSPRINT  DD SYSOUT=*
//IN1      DD UNIT=3390,VOL=SER=C9D11B,DISP=OLD
//OUT1     DD UNIT=3390,VOL=SER=D83VB5,DISP=OLD
 COPY FULL INDD(IN1) OUTDD(OUT1) ALLDATA(*) ALLEXCP
```

Examining the IPL Text

If you want to make sure that you created the LD-IPL text on your driver, you can do so using ADRDSSU's PRINT command:

```
//DSSPRT   JOB ,
//          MSGCLASS=A,CLASS=A,NOTIFY=&SYSUID,
//          USER=&SYSUID,RD=NC,
//          TIME=(0,30),MSGLEVEL=(1,1)
//STEP1 EXEC  PGM=ADRDSSU,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSABEND DD SYSOUT=A
//DASD     DD UNIT=3390,VOL=SER=D83TL5,DISP=OLD
//SYSIN    DD *
  PRINT  TRACKS(0,0,0,0)  INDDNAME(DASD)
/*
```

This will print the raw contents of track 0 on the specified DASD. If there is LD-IPL text on the DASD, at the bottom of the output, you will see sections with eyecatchers LDI1, LDI2, and so on.

```
0000  D3C4C9F1 7A49504C 00000001 00000000  00000000 00000000 00060000 00000000  *LDI1:.&<.....................*
0020  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *.............................*
PAGE 0009     5695-DF175  DFSMSDSS V2R05.0 DATA SET SERVICES     2023.123 14:47
0040  TO  01FF   SAME AS ABOVE
0200  00003031                                                                  *....                          *
   COUNT  0000000006040200
0000  D3C4C9F2 7A49504C 00000001 00000000  00000000 00000000 00070000 00000000  *LDI2:.&<.....................*
0020  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *.............................*
0040  TO  01FF   SAME AS ABOVE
0200  00000000                                                                  *....                          *
   COUNT  0000000007040200
0000  D3C4C9F3 7A49504C 00000000 00000000  00000000 00000000 00000000 00000000  *LDI3:.&<.....................*
0020  00000000 00000000 00080000 00000000  00000000 00000000 00000003 01000000  *.............................*
0040  000000FB 00000000 00090000 00000000  00000000 00000000 00000002 00000000  *.............................*
0060  00000000 00000000 00000000 00000000  00000000 00000000 00000001 00000000  *.............................*
0080  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *.............................*
00A0  TO  01FF   SAME AS ABOVE
```

Target z/OS System Setup

Switching to SCM PLPA data sets

Pageable modules that are in LPA can be paged to PLPA data sets. However, since the load module signature can not be checked when the module is being paged back in, to ensure that these load modules aren't tampered with while paged out, Validated Boot requires the use of Storage Class Memory (SCM) for your page data sets. If you don't have any SCM on your CPC, you will need to talk to your IBM Representative to get some. Until you have SCM, you can always IPL in Audit mode to check the rest of your setup.

In order to use SCM for PLPA, you need to first, make sure that the physical LPAR has space for SCM available to it, and second, tell z/OS to use SCM for PLPA. For more information about SCM, see the z/OS MVS Initialization and Tuning Guide.

You add or remove SCM from the image's Activation Profile, on the Storage tab. On the Activation Profile, it is referred to as Virtual Flash Memory.

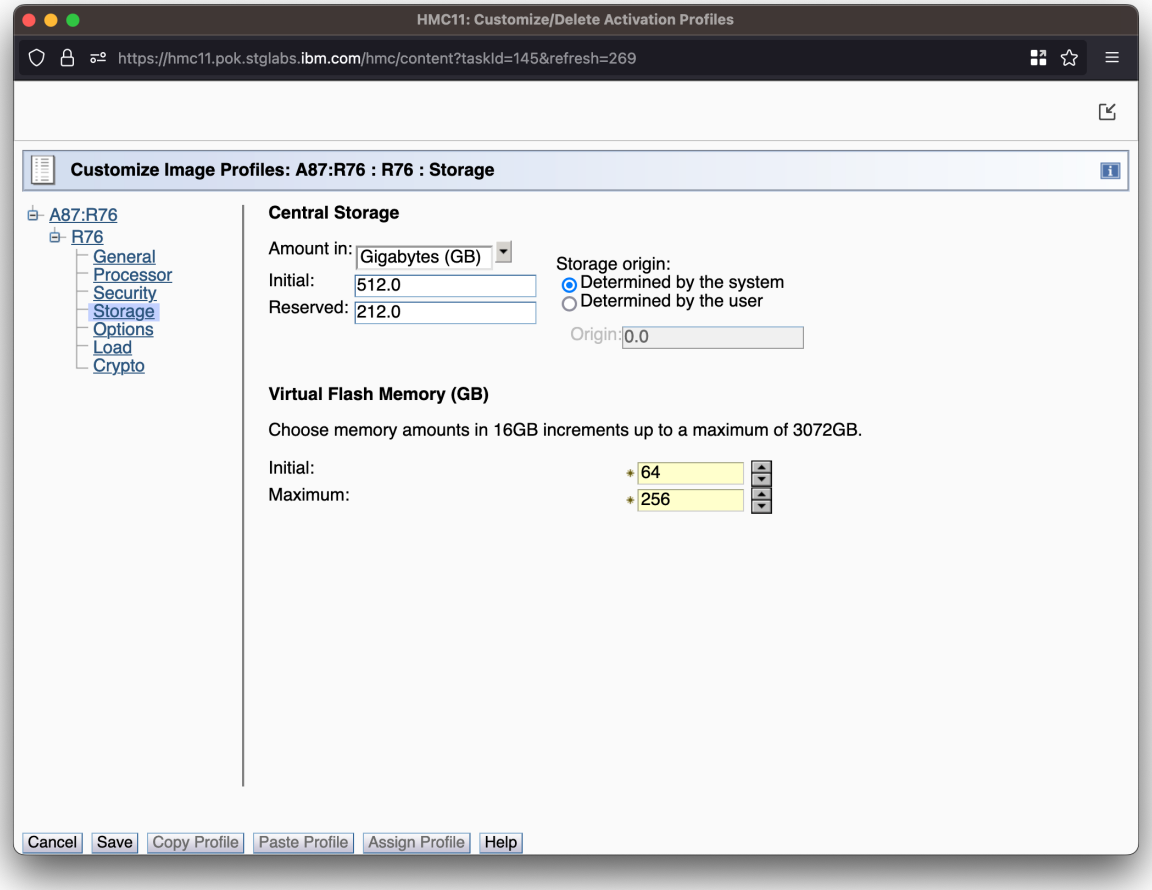


Figure 1: LPAR with Virtual Flash Memory defined

In z/OS itself, you specify what to use for PLPA in the parmlib member IEASYSxx, with the PAGE and PAGESCM parameters. To explicitly disable PLPA paging to DASD, you need to specify *NONE* as the first PAGE parameter.

In Validated Boot mode, if SCM is available, z/OS will automatically use it for PLPA paging, no matter what is specified in parmlib. In Audit mode, a message will be issued during IPL, and you will get an informational warning message when running the reporting tool after you IPL. If no SCM is available, the system will IPL, and you will get a warning message when running the reporting tool.

In enforce mode, If no SCM is available, then z/OS will wait state during IPL with wait state code A2D. If SCM is available, but the PAGE= statement directs z/OS to use a PLPA page data set, warning message ILR041I will be issued during IPL to notify you that SCM storage is being used.

Identifying Load Modules that need to be signed

In addition to SYS1.NUCLEUS, any modules that are loaded into LPA need to be signed. There are three types of LPA: pagable LPA (PLPA), fixed LPA (FLPA), and modified LPA (MLPA). You can identify these modules by either IPLing a system in audit mode, and running an audit report, or by looking at your system configuration and determining the list of modules that will be placed in each area.

Modules are loaded into PLPA at the dataset level, and are typically specified in LPALSTxx members, which are pointed to from the IEASYSxx parmlib member with LPA= statements.
Modules are loaded into FLPA on a module by module basis, and are specified in IEAFIXxx parmlib members, which are pointed to by FIX= statements in IEASYSxx.
Modules are loaded into MLPA on a module by module basis, and are specified in IEALPAxx parmlib members, and pointed to by an MLPA= statement in IEASYSxx.

Target Hardware Setup

The hardware support for Validated Boot will need to be installed on any CPC that will be IPLing in Validated Boot mode. Validated Boot is only supported on z16 CPCs.

Certificate Management

z/OS

Using the example provided above, the private signing key will be stored in ICSF's PKDS, and under ICSF's control. Other mechanisms may also be used to store the private signing key. A certificate containing the public key, however, needs to be exported so that it can be associated with every LPAR that will be IPLing a driver signed with the corresponding private key. In order to do that, you need to write the certificate to a dataset using RACF commands. The following command will write the certificate from above to data set XYZBLD.CERT1.FWCSCA.DER :

```
RACDCERT EXPORT(LABEL('XYZ Code Signing'))  DSN(cert1.fwcsca.der)
FORMAT(CERTDER) ID(XYZBLD)
```

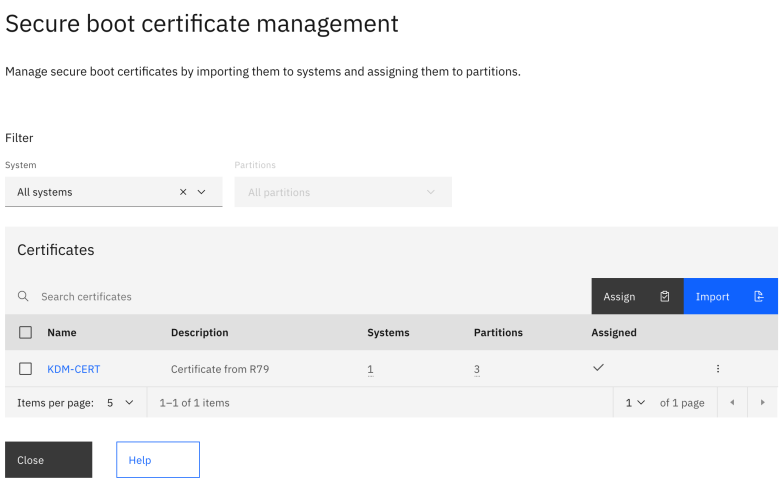
Once exported, the certificate will be in a binary format, so care must be taken to always transfer it in binary format.

HMC

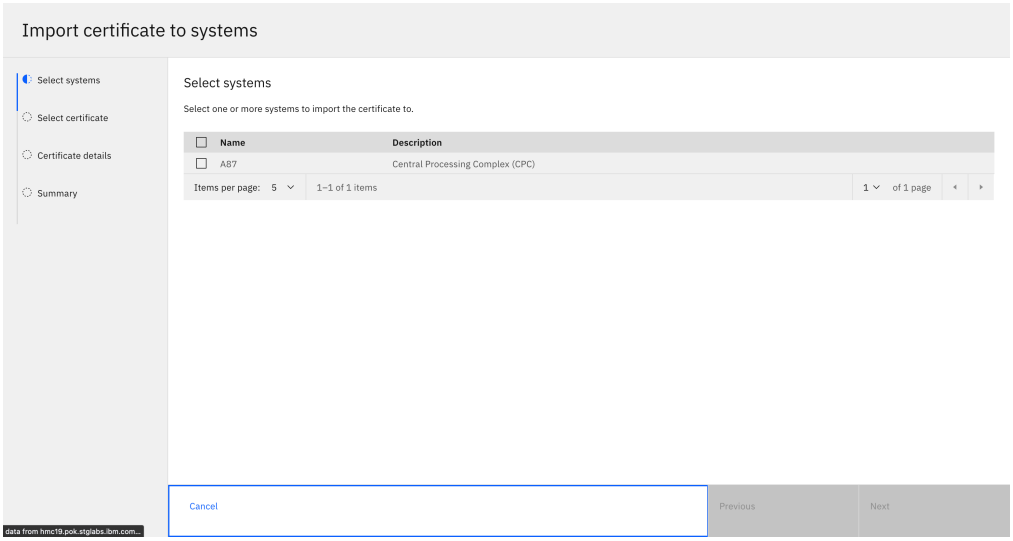
Once you've written the certificate out to a dataset, you need to import it to the CPC's SE via a connected HMC. You can do this from your browser, over the network, or, if you have physical access to an HMC, from a USB stick.

Over the network, you can import it via using FTP, FTPS, or SFTP. This could potentially be directly from z/OS, if the HMC can connect to it via one of the above mechanisms, or some other server. On the HMC, you use the "Secure boot certificate management" task to import certificates.

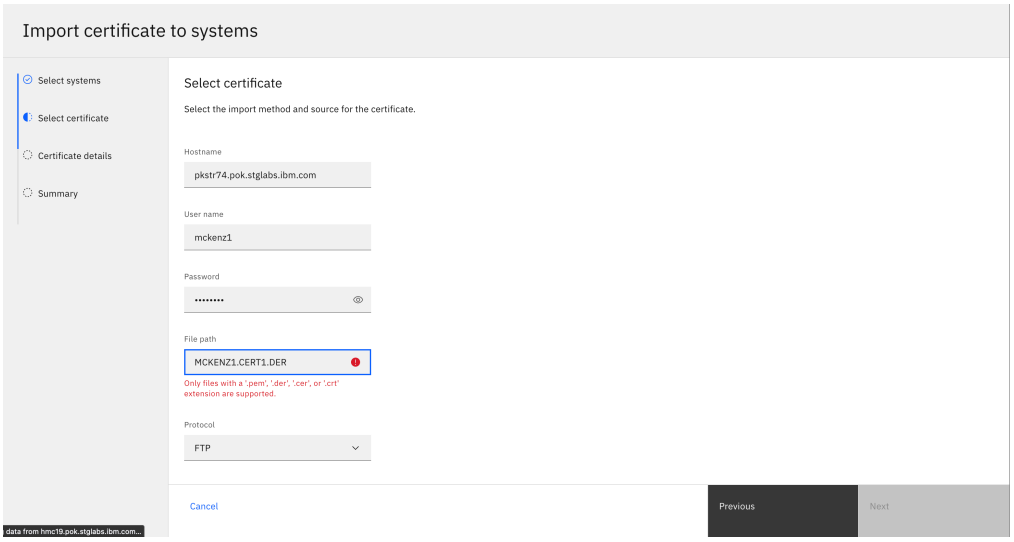
In this example, I'll be importing a certificate from userid MCKENZ1 on a z/OS image.



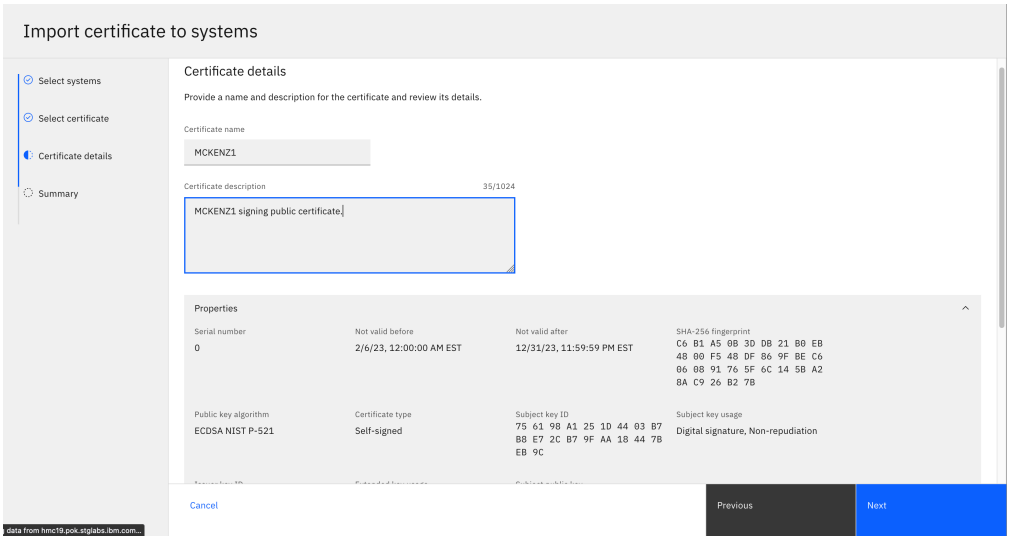
After choosing "Import", I'll be presented with a list of CPCs that the certificate can be imported to.



Next, you're presented with a screen prompting you for connection information. Note that, at the moment, the suffix of the file needs to be in lower case. Depending on where you're importing from, you may need to rename the file on the source system:



After connecting, you'll be prompted to give information about the certificate for identification purposes, and to review the summary information about the certificate:



The SHA-256 fingerprint should match the fingerprint generated in Step 8 of the RACF setup process.

After the import is finished, you will be taken back to the Certificate Management panel and choose which LPAR(s) on each CPC you want to be able to use the certificate with. You do this by clicking "Assign":

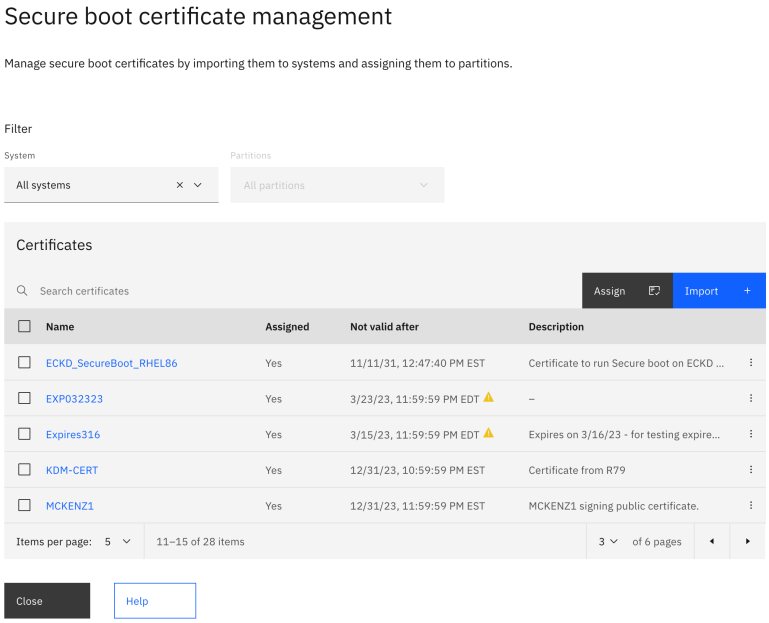
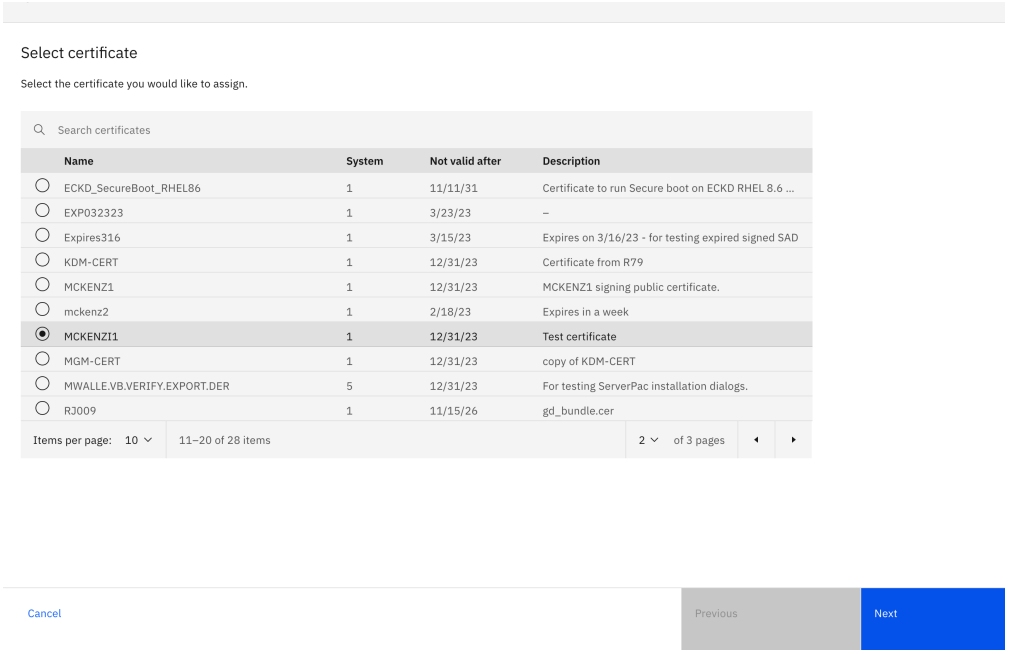
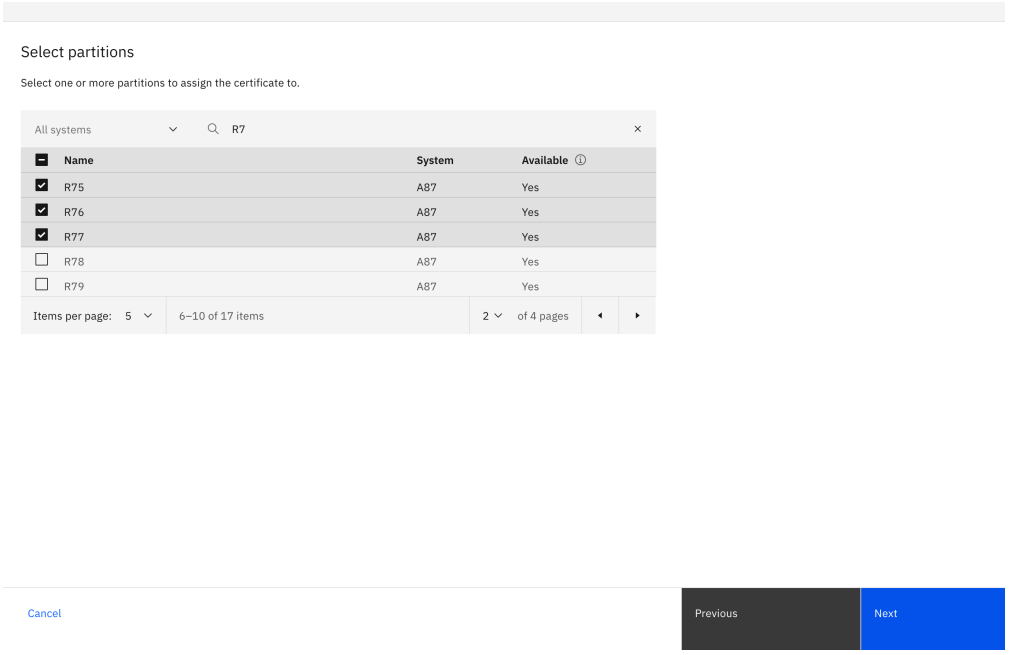


Figure 2: Selecting assign

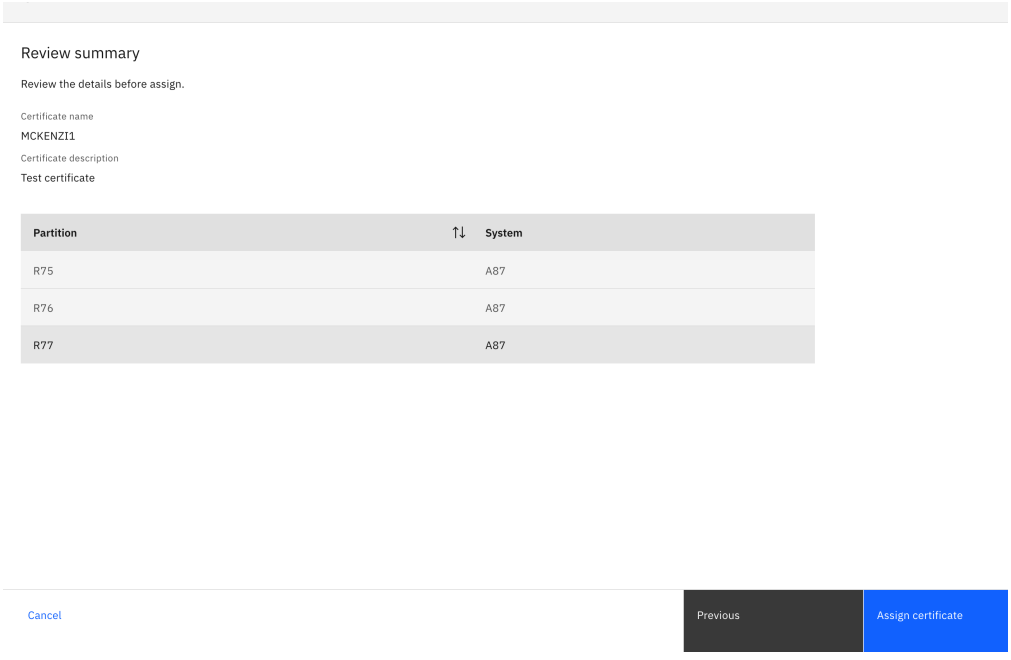
On the next panel, you select which certificate you want to assign and click the “Next” button.



Then you select which partition(s) you want to sign the certificate to, and click next:



After reviewing the Summary panel, click “Assign certificate” to assign the certificate to the selected images.



Using Validated Boot

IPLing a validated boot driver

Once the appropriate firmware is installed, you can attempt to IPL in validated boot Audit or Enforce mode. A new type of IPL, List Directed IPL, is required to IPL in Validated Boot mode, so the HMC panels look somewhat different. The IPL process itself will also look slightly different, as the zBootLoader will start prior to z/OS IPLing.

The panels are significantly different, partly due to the new Validated Boot support, but also due to the addition of the Load type.

In order to IPL z/OS, you need to select Device type ECKD. For IPL Type, you have two choices: Channel Command Word (CCW), and List-directed. CCW IPL is classic, non-Validated Boot IPL. If you select List-directed IPL, you'll come up in Validated Boot mode. If the "Enable Secure Boot" box is checked, you will come up in Validated Boot Enforce Mode. If the box is not checked, you will come up in Validated Boot Audit Mode.

The Load type choice is added with the Validated Boot firmware, but is in general unrelated. It was added to respond to a number of client requests, where clients had meant to capture diagnostic data, but accidentally re-IPLed z/OS instead. Now, you will need to explicitly say if you are IPLing an OS or a dump program.

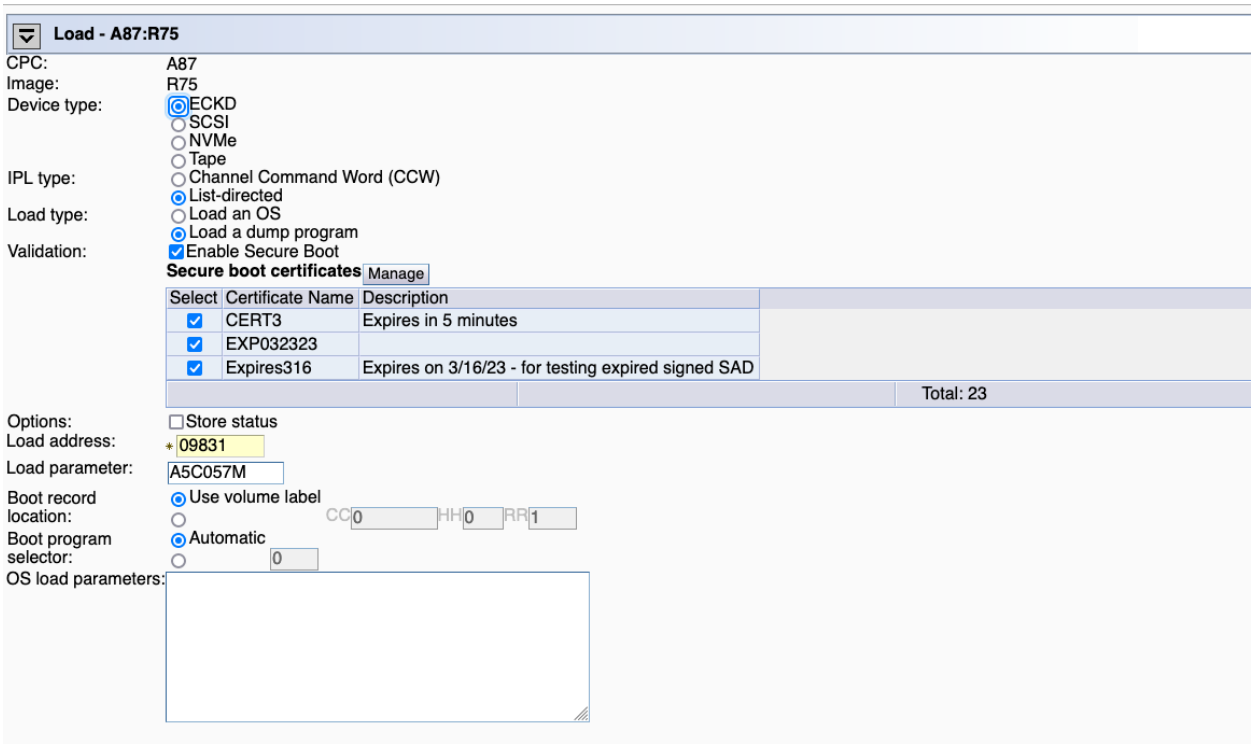


Figure 3: Validated Boot Enforce Mode

After selecting the desired options, you choose OK as normal, and confirm that you do want to IPL.

A Validated Boot IPL actually involves two steps, so it will look slightly different than a regular IPL. The first step involves the zBootLoader loading, and validating the signed IPL text of the target system. If this validation is successful, then the zBootLoader will pass control to z/OS to IPL. In practice, what this means is that you will get a few console messages from the zBootLoader prior to the z/OS IPL messages starting.

```
IPB received.
IPB sent.
System version 9.
Watchdog enabled.
Running 'ZBootLoader' version '3.1.5' level 'D51C.D51C_328.13'.
```

Displaying information about validated boot

Once the system is up, you can get information about the state of the system from the 'D IPLINFO' console command. A new status line has been added, as in the following:

```
IEE254I  14.28.16 IPLINFO DISPLAY 043
 SYSTEM IPLED AT 10.54.42 ON 04/17/2023
 RELEASE z/OS 02.05.00    LICENSE = z/OS
 USED LOAD57 IN SYS0.IPLPARM ON 0A5C0
 ARCHLVL = 2   MTLSHARE = N
 VALIDATED BOOT: AUDIT,INACTIVE
 IEASYM LIST = (X7,R5,U7,L)
 IEASYS LIST = (VB,X7) (OP)
 IODF DEVICE: ORIGINAL(0A5C0) CURRENT(0A5C0)
 IPL DEVICE: ORIGINAL(09826) CURRENT(09826) VOLUME(VLB7L5)
```

The Validated Boot line is the new one. It will either report "NO", if Validated Boot isn't enabled at all; "AUDIT" if you have IPLed in Audit mode, and "ENFORCE" if you've IPLed in Enforce Mode. The part after the comma, if present, is always going to be "INACTIVE"; it indicates if z/OS is currently doing any enforcement of Validated Boot. By the time the system is able to respond to console commands, the modules that are loaded into LPA have already been loaded, so it will always report "INACTIVE."

Debugging Validated Boot

If you IPL a system in Enforce Mode and any signing issues are found, the system will wait state, and the message issued with the wait state should indicate where the problem was. However, the system will wait state immediately upon finding an exception, so you might have to repeat

the IPL process many times to find all the exceptions. The other option is to IPL in Audit Mode and run IEAVBRPT, to help debug problems IPLing with Validated Boot. The tool will only work in Audit Mode; trying to run it in non-Validated Boot mode or in Validated Boot Enforce mode will result in an error.

Sample JCL to run the tool is as follows:

```
//VBPRT1 EXEC PGM=IEAVBPRT,TIME=1440,
//  PARM='DETAIL'
//SYSPRINT DD SYSOUT=*
```

In the job output, you'll get information about any exceptions, and then information about any load modules that failed validation, and the load library that contain those modules. At the end, you'll have a list of certificates that the system has access to, along with any certificates that the system should have access to but can't use for some reason, such as being expired.

Stand Alone Dump and AutoIPL

If you are using AutoIPL to recover in the event of a wait state, either by reIPLing z/OS immediately, or taking a Stand Alone Dump and then reIPLing z/OS, you may need to change your procedures somewhat when using Validated Boot.

First, while not specific to Validated Boot, there's a new step that needs to be taken when IPLing Stand Alone Dump with the new HMC/SE.

When taking a Stand Alone Dump of a system IPLed in Validated Boot mode manually (ie, invoked from the HMC), you do not need to use a Stand Alone Dump built with Validated Boot. A Stand Alone Dump built with Validated Boot can take a dump of a system not IPLed in Validated Boot mode, and a Stand Alone Dump not built with Validated Boot can take a dump of a system IPLed in Stand Alone Dump mode. What is not allowed is for a system image to switch between Validated Boot mode and non-Validated Boot mode without some sort of manual operator intervention. So if you use AutoIPL to automatically take a dump and then re-IPL, if the initial system is in Validated Boot mode, the Stand Alone Dump image, and the final system all need to be built with Validated Boot support as well.

This is a current partial listing of United States trademarks owned by International Business Machines Corporation ("IBM") and might also be trademarks or registered trademarks in other countries. Please note that laws concerning use and marking of trademarks or product names vary by country. Always consult a local attorney for additional guidance.

Not all common law marks used by IBM are listed on this page. Failure of a mark to appear on this page does not mean that IBM does not use the mark nor does it mean that the product is not actively marketed or is not significant within its relevant market.

Those trademarks followed by ® are registered trademarks of IBM in the United States; those followed by ™ are trademarks or common law marks of IBM in the United States.

GDPS®
IBM Z®
IBM z Systems®
IBM z15®
IBM z16®
z Systems®
z/OS®
z/VM®
z/VSE®