Using a Serials file with Validation Authority:

Each certificate is required to have a serial number. This serial is assigned by the CA at the time of signing. Because the data type is specified as a non-negative integer of up to 20 octets length (160 bit), a CA can create an astronomical high number of certs. The serial number is not a normal integer datatype, OpenSSL uses the special ASN1_INTEGER data type which is not really a 'number' but rather an array of bytes. See RFC3280 for more information on serial numbers in X.509 certificates.

Per standard, the serial number should be unique per CA, however it is up to the CA code to enforce this. For example, with OpenSSL makes it possible to manually set the serial during signing, using the `-set_serial` option. Although not officially standardized, a CA should give out serials at random on one hand (to prevent predictability), and tracking them to be unique on the other hand.

The admin UI only allows you to generate a presigned OCSP response database based on a serial number range. However, some CAs generate serial numbers randomly and some CAs use nonsequential serial numbers. Because it is difficult for users to input a list of nonsequential serial number for precomputing OCSP responses using the admin UI, VA Server includes the `vatool` commandline tool.

A serials file is nothing but a text file containing the serial numbers in ASN1_INTEGER form of your certificates. They can be in decimal or hexadecimal form but if they are hexadecimal they must be preceded with “0x”.

There are numerous ways to determine the serial numbers used in CA certificates. We list three such methods.

1. This is probably the easiest and most straight forward way to determine the serials numbers used. It consists of asking your Certificate Authority what serial numbers they assign and then you compile them in your “serials” file. Note it is not unusual for a CA to be asked what serial numbers they assign to their certificates.

2. The second method involves using the `vatool` command the is provided with Validation Authority installations. It is located in the “tools” directory. The following example shows the format of the `vatool` command to publish a list of serial numbers to the VA for precomputing OCSP responses.

```
vatool -cacert <cacertfile> -crlsdir <crlsdir> -serialsfile <serialsfile>
       -inform [ HEX | DEC ]
```

The following table describes the command arguments.

Argument	Description
cacert <cacertfile>	Specifies the directory for the CA that issued the serial numbers to be precomputed.
crlsdir <crlsdir>	Specifies the directory containing the CRLs.
serialsfile <serialsfile>	Specifies the name of the ASCII file that contains the list of serial numbers for which OCSP responses should be precomputed.
inform [HEX | DEC]	Specifies the numerical format of the output, where HEX means hexadecimal and DEC means decimal values.

`vatool` performs the following functionality:

- Uses the CA certificate and server base directory to figure out the CA directory in the server installation directory.
- Validates the input ACII Serial numbers file. The format of this file must be as follows:
  o Serial file will contain one or more lines of absolute serial numbers or serial ranges.
  o Absolute serial numbers are specified in either hexadecimal or decimal number format in a string.
  o Serial range is specified as two serial numbers separated by a '-' (hyphen) character.
  o Following grammar describes the serial file format
    - SerialInfoList:= (SerialInfo)*
    - SerialInfo:= Serial | SerialRange
    - SerialRange:= Serial-Serial
- Converts the ASCII serial number information to ASN.1 serial number information.
- Saves the ASN.1 serial numbers information to file serials.bin in the CA directory
- Returns 0 for success, error code for failure.

If the CA has been configured to precompute OCSP responses for Certificates Listed in Serials File in the admin UI, then a scheduled task in the server runs periodically to check for updates to the ASN.1 file generated by vatool. If the Server detects a change in the file, it triggers the OCSP precomputation for those CA serial numbers. The frequency with which the Serials file is monitored can be also be configured through the admin UI.

If you select Certificates Listed in Serials File, you can also select Pre-compute Incremental DBs to enable VA Server to precompute the recently added certificates in the OCSP database incrementally. A new incremental database is generated every time VA Server detects a change to the serials.txt file. Serials File Monitor Frequency represents the frequency used to look for changes to CA issued serial numbers for precomputing OCSP responses. This is only relevant in cases where the CA is publishing a list of serial numbers (in conjunction with the CRL) to the VA Responder. This ensures that OCSP responses are only generated for issued certificates, as opposed to trying to ascertain the number of issued certificates based on the CRL alone.

3. The third and final method for determining the serial numbers of certificates is to compile and run the included program GetSerial.c listed below. Note that this program requires the certificate be in the PEM format. Techniques for converting other formats to PEM are provided below the source listing for GetSerial.c.

```
/* ---------------------------------------------------------- *
 * file:        GetSerial.c                                   *
 * purpose:     Program for extracting OpenSSL certificate    *
 *              serial numbers.                               *
 * To compile run:                                            *
 *              gcc -o GetSerial GetSerial.c -lssl -lcrypto   *
 * ---------------------------------------------------------- */

#include <openssl/bio.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/x509.h>

int main(int argc, char *argv[]) {

  /* const char cert_filestr[]; */
  ASN1_INTEGER *asn1_serial = NULL;
  BIO             *certbio = NULL;
  BIO              *outbio = NULL;
  X509               *cert = NULL;
  const char *neg;
```

```
  int ret, i, iter;
  long l;

for (iter=1; iter < argc; iter++) {

   printf("[%3d of %3d] File: %s ", iter, argc-1, argv[iter]);

   /* These function calls initialize openssl for correct work.  */
   OpenSSL_add_all_algorithms();
   ERR_load_BIO_strings();
   ERR_load_crypto_strings();

   /* Create the Input/Output BIO's.                              */
   certbio = BIO_new(BIO_s_file());
   outbio  = BIO_new_fp(stdout, BIO_NOCLOSE);

   /* Load the certificate from file (PEM).                       */
   ret = BIO_read_filename(certbio, argv[iter]);
   if (! (cert = PEM_read_bio_X509(certbio, NULL, 0, NULL)))
      BIO_printf(outbio, "Error loading cert into memory\n");

   /* Extract the certificate's serial number.                    */
   asn1_serial = X509_get_serialNumber(cert);
   if (asn1_serial == NULL)
      BIO_printf(outbio,
         "Error getting serial number from certificate");

   /* Print the serial number value, openssl x509 -serial style  */
   BIO_puts(outbio,"serial: 0x");
   i2a_ASN1_INTEGER(outbio, asn1_serial);
   BIO_puts(outbio, "\n");

   /* Print the serial number value, openssl x509 -text style    */
   if (asn1_serial->length <= (int)sizeof(long)) {
      l=ASN1_INTEGER_get(asn1_serial);
      if (asn1_serial->type == V_ASN1_NEG_INTEGER) {
         l= -l;
         neg="-";
      }
      else
         neg="";

      if (BIO_printf(outbio,
         " (%s%lu) %s0x%lx\n", neg, l, neg, l) <= 0)
         BIO_printf(outbio, "Error during printing the serial.\n");
   }
   else
   {
      neg=(asn1_serial->type == V_ASN1_NEG_INTEGER)?" (Negative)":"";
      if (BIO_printf(outbio, "serial: %s", "", neg) <= 0)
         BIO_printf(outbio, "Error during printing the serial.\n");

      for (i=0; i<asn1_serial->length; i++) {
         if (BIO_printf(outbio, "%02x%c", asn1_serial->data[i],
              ((i+1 == asn1_serial->length)?'\n':':')) <= 0)
         BIO_printf(outbio, "Error during printing the serial.\n");
      }
```

```
    }
}

X509_free(cert);
BIO_free_all(certbio);
BIO_free_all(outbio);
exit(0);
}
```

The following show how to compile and link `GetSerial.c` to produce the executable “GetSerial”.

```
[root@RHEL6-8 ~]# gcc -o GetSerial GetSerial.c -lssl -lcrypto
```

Next we run the program on each of the PEM files located in the `/opt` directory. Notice that five PEM files are found but the program ends on the third because it is not a proper certificate.

```
[root@RHEL6-8 ~]# ./GetSerial `find /opt -name "*.pem"`
[  1 of   5] File: /opt/va/apache/cgi-bin/cert_import.pem serial: 0x05
 (5) 0x5
[  2 of   5] File: /opt/va/tools/demoCA/CAcert.pem serial: 0xD1CE3BDE4DA5D1D0
 (15118086825100825040) 0xd1ce3bde4da5d1d0
[  3 of   5] File: /opt/va/tools/demoCA/private/CAkey.pem Error loading cert into
memory
Segmentation fault (core dumped)
```

Using the openssl command to convert to PEM format files

P7B to PEM	`openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer`
DER to PEM	`openssl x509 -inform der -in certificate.cer -out certificate.pem`