How to generate a self-signed SSL certificate using OpenSSL?

Asked 9 years, 10 months ago Active 20 days ago Viewed 2.3m times

1678

993

I'm adding HTTPS support to an embedded Linux device. I have tried to generate a self-signed certificate with these steps:

```
openssl req -new > cert.csr
openssl rsa -in privkey.pem -out key.pem
openssl x509 -in cert.csr -out cert.pem -req -signkey key.pem -days 1001
cat key.pem>>cert.pem
```

This works, but I get some errors with, for example, Google Chrome:

> This is probably not the site you are looking for!
> The site's security certificate is not trusted!

Am I missing something? Is this the correct way to build a self-signed certificate?

ssl openssl certificate ssl-certificate x509certificate

Share Follow

edited May 25 2021 at 6:57
Jesse Nickles

asked Apr 16 2012 at 14:14
michelemarcon
20.9k 16 51 66

55 Self-signed certificates are considered insecure for the Internet. Firefox will treat the site as having an invalid certificate, while Chrome will act as if the connection was plain HTTP. More details: gerv.net/security/self-signed-certs – user1202136 Apr 16 2012 at 14:17

51 You need to import your CA certificate into your browsers and tell the browsers you trust the certificate -or- get it signed by one of the big money-for-nothing organizations that are already trusted by the browsers -or- ignore the warning and click past it. I like the last option myself. – trojanfoe Apr 16 2012 at 14:20

19 You should not use the "stock" OpenSSL settings like that. That's because you cannot place DNS names in the Subject Alternate Name (SAN). You need to provide a configuration file with an `alternate_names` section and pass it with the `-config` option. Also, placing a DNS name in the Common Name (CN) is deprecated (but not prohibited) by both the IETF and the CA/Browser Forums. Any DNS name in the CN must also be present in the SAN. There's no way to avoid using the SAN. See answer below. – jww Jan 13 2015 at 22:01

6 In addition to @jww 's comment. Per may 2017 Chrome doesn't accept certs w/o (emtpy) SAN's anymore: "The certificate for this site does not contain a Subject Alternative Name extension containing a domain name or IP address." – GerardJP May 25 2017 at 7:35

9 These days, as long as your webserver is accessible by its FQDN on port 80 over the internet, you can use LetsEncrypt and get free full CA certs (valid for 90 days, renewal can be automated) that won't give any browser warnings/messages. www.letsencrypt.com – DisappointedByUnaccountableMod Mar 27 2018 at 12:13

Active Oldest Score

2710

You can do that in one command:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -days 365
```

You can also add `-nodes` (short for `no DES`) if you don't want to protect your private key with a passphrase. Otherwise it will prompt you for "at least a 4 character" password.

The `days` parameter (365) you can replace with any number to affect the expiration date. It will then prompt you for things like "Country Name", but you can just hit Enter and accept the defaults.

Add `-subj '/CN=localhost'` to suppress questions about the contents of the certificate (replace `localhost` with your desired domain).

Self-signed certificates are not validated with any third party unless you import them to the browsers previously. If you need more security, you should use a certificate signed by a certificate authority (CA).

Share Follow

edited Oct 27 2021 at 12:35

Hongli
18.2k 15 76 104

answered Apr 16 2012 at 15:04

Diego Woitasen
29.5k 1 15 20

20 For anyone who's interested, here is the documentation, if you want to verify anything yourself. – user456814 Apr 15 2014 at 17:34

24 How does signing with a 3rd-party provide more security? – James Mills Jul 11 2014 at 3:14

284 For anyone else using this in automation, here's all of the common parameters for the subject:
lame/OU=Org/CN=www.example.com"

guy with "free candy" written on the side
g to think twice and be on guard about it --
naw man, he's legit" you're going to be all
3:39

ed certificate. – Gea-Suan Lin Jan 25

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

Am I missing something? Is this the correct way to build a self-signed certificate?

625

It's easy to create a self-signed certificate. You just use the `openssl req` command. It can be tricky to create one that can be consumed by the largest selection of clients, like browsers and command line tools.

It's difficult because the browsers have their own set of requirements, and they are more restrictive than the [IETF](). The requirements used by browsers are documented at the [CA/Browser Forums]() (see references below). The restrictions arise in two key areas: (1) trust anchors, and (2) DNS names.

Modern browsers (like the warez we're using in 2014/2015) want a certificate that chains back to a trust anchor, and they want DNS names to be presented in particular ways in the certificate. And browsers are actively moving against self-signed server certificates.

Some browsers don't exactly make it easy to import a self-signed server certificate. In fact, you can't with some browsers, like Android's browser. So the complete solution is to become your own authority.

In the absence of becoming your own authority, you have to get the DNS names right to give the certificate the greatest chance of success. But I would encourage you to become your own authority. It's easy to become your own authority, and it will sidestep all the trust issues (who better to trust than yourself?).

This is probably not the site you are looking for!

anchors to validate server certificates.

d anchor.

Your privacy

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

erver

a self-signed certificate with `CA:`

`true` and proper key usage. That means the *Subject* and *Issuer* are the same entity, CA is

Store used by the browser.

Steps 2 - 4 are roughly what you do now for a public facing server when you enlist the services of a CA like [Startcom]() or [CAcert](). Steps 1 and 5 allows you to avoid the third-party authority, and act as your own authority (who better to trust than yourself?).

The next best way to avoid the browser warning is to trust the server's certificate. But some browsers, like Android's default browser, do not let you do it. So it will never work on the platform.

The issue of browsers (and other similar user agents) *not* trusting self-signed certificates is going to be a big problem in the Internet of Things (IoT). For example, what is going to happen when you connect to your thermostat or refrigerator to program it? The answer is, nothing good as far as the user experience is concerned.

The W3C's WebAppSec Working Group is starting to look at the issue. See, for example, [Proposal: Marking HTTP As Non-Secure]().

How to create a self-signed certificate with OpenSSL

The commands below and the configuration file create a self-signed certificate (it also shows you how to create a signing request). They differ from other answers in one respect: the DNS names used for the self signed certificate are in the *Subject Alternate Name (SAN)*, and not the *Common Name (CN)*.

The DNS names are placed in the SAN through the configuration file with the line

do it through the command line).

ration file (you should tune this to

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

```
t want them or
 development.
```

Related: browsers follow the CA/Browser Forum policies; and not the IETF policies. That's one of the reasons a certificate created with OpenSSL (which generally follows the IETF) sometimes does not validate under a browser (browsers follow the CA/B). They are different standards, they have different issuing policies and different validation requirements.

Create a self signed certificate (notice the addition of `-x509` option):

```
openssl req -config example-com.conf -new -x509 -sha256 -newkey rsa:2048 -nodes \
    -keyout example-com.key.pem -days 365 -out example-com.cert.pem
```

Create a signing request (notice the lack of `-x509` option):

```
openssl req -config example-com.conf -new -sha256 -newkey rsa:2048 -nodes \
    -keyout example-com.key.pem -days 365 -out example-com.req.pem
```

Print a self-signed certificate:

```
openssl x509 -in example-com.cert.pem -text -noout
```

Print a signing request:

```
openssl req -in example-com.req.pem -text -noout
```

Configuration file (passed via `-config` option)

```
commonName_default        = Example Company

emailAddress            = Email Address
emailAddress_default        = test@example.com

# Section x509_ext is used when generating a self-signed certificate. I.e., openssl
req -x509 ...
[ x509_ext ]
```

You may need to do the following for Chrome. Otherwise [Chrome may complain a *Common Name* is invalid (`ERR_CERT_COMMON_NAME_INVALID`)]. I'm not sure what the relationship is between an IP address in the SAN and a CN in this instance.

```
# IPv4 localhost
# IP.1       = 127.0.0.1

# IPv6 localhost
# IP.2     = ::1
```

There are other rules concerning the handling of DNS names in X.509/PKIX certificates. Refer to these documents for the rules:

- RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC 6125, Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)
- RFC 6797, Appendix A, HTTP Strict Transport Security (HSTS)
- RFC 7469, Public Key Pinning Extension for HTTP

5 Is it possible to use wildcards in the `alternate_names` section? Particularly sub-sub domains. I have a question referencing this answer here: serverfault.com/questions/711596/... – LeonardChallis Aug 12 2015 at 10:02

3 I've just replied to his specific question. I think doesn't make sense to add this long security description when the answer was so simple – Diego Woitasen Feb 21 2016 at 1:44

19 @diegows - your answer is not complete or correct. The reason it is not correct is discussed in the long post you don't want to read :) – jww Feb 21 2016 at 4:42

1 Thanks! I found your post very helpful. FYI I was recently playing with vault and found it insisted on IP.x 127.0.0.1 rather than DNS.x 127... I didn't check if this is in the standard or not. – Chomeh Aug 22 2016 at 0:32

4 Thank you @jww. You said, *"1. Create your own authority (i.e, become a CA)"*, then said, *"5. Install the CA certificate on the client"*. If the root key became compromised, a malicious person could sign a cert for any domain with that key, and if they trick you into going to their website, they can now do a man-in-the-middle attack. Is there a way to create the root CA such that it can only sign intermediary CAs and not certificates? Then you can protect your intermediary CA with a name constraint. – Robin Zimmermann Mar 30 2017 at 18:41

532

# Update 2022

As of 2022 with OpenSSL ≥ 1.1.1, the following command serves all your needs, including Subject Alternate Name (SAN):

```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
  -keyout example.key -out example.crt -subj "/CN=example.com" \
  -addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
```

On old systems with OpenSSL ≤ 1.1.0, such as Debian ≤ 9 or CentOS ≤ 7, a longer version of this command needs to be used:

```
openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
  -keyout example.key -out example.crt -extensions san -config \
  <(echo "[req]";
    echo distinguished_name=req;
    echo "[san]";
    echo subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1
    ) \
  -subj "/CN=example.com"
```

Either command creates a certificate that is

- valid for the (sub)domains `example.com` and `www.example.net` (SAN),
- also valid for the IP address `10.0.0.1` (SAN),
- relatively strong (as of 2022) and

s **no interactive input** that annoys
d with. All necessary steps are
te key generation up to the self-

epted by users manually, it doesn't
aphy.

Theoretically you could leave out the `-nodes` parameter (which means "no DES encryption"), in which case `example.key` would be encrypted with a password. However, this is almost never useful for a server installation, because you would either have to store the password on the server as well, or you'd have to enter it manually on each reboot.

Remark #3: See also

- Provide subjectAltName to openssl directly on command line
- How to add multiple email addresses to an SSL certificate via the command line?
- More information about MSYS_NO_PATHCONV

Share Follow edited Jan 19 at 13:11 answered Dec 28 2016 at 17:30

vog
20k 9 55 73

2 I tried to use the oneliner #2 (modern) on windows in mingw64, and I faced a bug with -subj parameter. ` $ openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes -keyout localhost.key -out localhost.crt -subj '/CN=localhost' -addext subjectAltName=DNS:localhost,IP:127.0.0.1 Generating a RSA private key [...] writing new private key to 'localhost.key' ----- name is expected to be in the format /type0=value0/type1=value1/type2=...
that format: 'C:/Program Files/Git
ri Pozniak Dec 23 2018 at 14:12

N=localhost expanding to C:/Program
plain cmd.exe and it worked just fine.
zniak Dec 23 2018 at 14:15

10 years from now? Because that's the
short expiration or weak crypto. Most
ost. Regarding OpenSSL 1.1.1, I'm still
change if you want a stronger hash.

ed SAN without having to create a long-
ua Pinter Sep 23 2019 at 3:35

really would like to see a reference that
Part of me wonders if it's just because the
e big tech cos. What is going to be
ament of that the amount of activity this
020 at 2:51

441

Here are the options described in @diegows's answer, described in more detail, from the documentation:

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days XXX
```

> `req`
>
> PKCS#10 certificate request and certificate generating utility.
>
> `-x509`
>
> this option outputs a self signed certificate instead of a certificate request. This is typically used to generate a test certificate or a self signed root CA.
>
> `-newkey arg`
>
> this option creates a new certificate request and a new private key. The argument takes one of several forms. **rsa:nbits**, where **nbits** is the number of bits, generates an RSA key **nbits** in size.
>
> `-keyout filename`
>
> this gives the filename to write the newly created private key to.
>
> ndard output by default.
>
> the number of days to certify the
>
> ated it will not be encrypted.

bove; I just summarized it here.

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

3 The `XXX` in the original command should be replaced with the 'number of days to certify the certificate for'. The default is 30 days. For example, `-days XXX` becomes `-days 365` if you want your cert to be valid for 365 days. See the docs for more. – Nathan Jones Oct 19 2016 at 21:11

Thanks for adding the documentation. This IBM link on creating a self-signed certificate using command which seems identical to this answer – The Red Pea Jun 1 2017 at 15:30

169

I can't comment, so I will put this as a separate answer. I found a few issues with the accepted one-liner answer:

- The one-liner includes a passphrase in the key.
- The one-liner uses SHA-1 which in many browsers throws warnings in console.

Here is a simplified version that removes the passphrase, ups the security to suppress warnings and includes a suggestion in comments to pass in -subj to remove the full question list:

```
openssl genrsa -out server.key 2048
openssl rsa -in server.key -out server.key
openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'
openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out
server.crt
```

ou will need to run the first two
assphrase.

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

46

124

answered Aug 13 2015 at 9:44

Mike N
5,467 3 22 19

7 I needed a dev certificate for github.com/molnarg/node-http2 and this answer is just the best. – Capaj Nov 8 2015 at 11:09

1 To combine the certificate and the key in a single file: `cat server.crt server.key >foo-cert.pem` . Works with the example in `openssl-1.0.2d/demos/ssl/` – 18446744073709551615 Nov 9 2015 at 12:33

The cert I generated this way is still using SHA1. – user169771 Jan 7 2016 at 14:58

1 Tks, works great to create a self signed certificate on `FreeBSD 10` `OpenLDAP 2.4` `with` `TLS` – Thiago Pereira Oct 3 2016 at 20:34

3 What about the key.pem file? – quikchange Nov 17 2016 at 10:24

80

Modern browsers now throw a security error for otherwise well-formed self-signed certificates if they are missing a SAN (Subject Alternate Name). ***OpenSSL does not provide a command-line way to specify this***, so many developers' tutorials and bookmarks are suddenly outdated.

The quickest way to get running again is a short, stand-alone conf file:

1. Create an OpenSSL config file (example: `req.cnf`)

```
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_req
prompt = no
[req_distinguished_name]
C = US
ST = VA
L = SomeCity
O = MyCompany
OU = MyDivision
CN = www.company.com
[v3_req]
keyUsage = critical, digitalSignature, keyAgreement
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1 = www.company.com
DNS.2 = company.com
DNS.3 = company.net
```

2. Create the certificate referencing this config file

```
openssl req -x509 -nodes -days 730 -newkey rsa:2048 \
                                           f -sha256
```

[TX135602](#)

11 answered May 9 2017 at 2:37

278 rymo **3,099** 2 32 39

1 It worked for me after removing the last parameter -extensions 'v3_req' which was causing an error. Using OpenSSL for windows. Finally, I manage to fix this issue! Thanks. – CGodo May 10 2017 at 18:25

1 @Kyopaxa you're right - that parameter is redundant with line 3 of the cnf file; updated. – rymo May 10 2017 at 21:23

2 Solid way. Thanks. I'd suggest adding `-sha256` . – cherouvim Jul 27 2017 at 10:13

5 You can now specify the SAN on the command line with `-extension 'subjectAltName = DNS:dom.ain, DNS:oth.er'` see github.com/openssl/openssl/pull/4986 – Alexandre DuBreuil Jan 23 2018 at 11:02

3 Looks like this option is called `-addext` now. – Alexandr Zarubkin Nov 20 2018 at 14:10

73

I would recommend to add the **-sha256** parameter, to use the SHA-2 hash algorithm, because major browsers are considering to show "SHA-1 certificates" as not secure.

The same command line from the accepted answer - @diegows with added -sha256

> openssl req -x509 **-sha256** -newkey rsa:2048 -keyout key.pem -out cert.pem -days XXX

More information in Google Security blog.

Update May 2018. As many noted in the comments that using SHA-2 does not add any security to a self-signed certificate. But I still recommend using it as a good habit of not using outdated / insecure cryptographic hash functions. Full explanation is available in *Why*

o be SHA-1 based?.

16 answered Oct 20 2014 at 9:52

124 Maris B.
2,172 3 20 34

Your privacy

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

1 If it's a self signed key, it's going to generate browser errors anyway, so this doesn't really matter – Mark Dec 16 2014 at 13:43

32 @Mark, it matters, because SHA-2 is more secure – Maris B. Dec 17 2014 at 15:38

1 Opening the certificate in windows after renaming the cert.pem to cert.cer says the fingerprint algorithm still is Sha1, but the signature hash algorithm is sha256. – sinned Dec 19 2014 at 8:33

2 "World-class encryption * zero authentication = zero security" gerv.net/security/self-signed-certs – x-yuri Mar 29 2018 at 9:03

4 Note that the signature algorithm used on a self-signed certificate is irrelevant in deciding whether it's trustworthy or not. Root CA certs are self-signed. and as of May 2018, there are still many active root CA certificates that are SHA-1 signed. Because it doesn't matter if a certificate trusts itself, nor how that certificate verifies that trust. You either trust the root/self-signed cert for *who* it says it is, or you don't. See security.stackexchange.com/questions/91913/... – Andrew Henle May 8 2018 at 18:55

23

This is the script I use on local boxes to set the SAN (subjectAltName) in self-signed certificates.

This script takes the domain name (example.com) and generates the SAN for *.example.com and example.com in the same certificate. The sections below are commented. Name the script (e.g. `generate-ssl.sh`) and give it executable permissions. The files will be written to the same directory as the script.

Chrome 58 an onward requires SAN to be set in self-signed certificates.

```
#!/usr/bin/env bash

# Set the TLD domain we want to use
BASE_DOMAIN="example.com"

# Days for the cert to live
DAYS=1095

# A blank passphrase
PASSPHRASE=""

# Generated configuration file
CONFIG_FILE="config.txt"

cat > $CONFIG_FILE <<-EOF
[req]
default_bits = 2048
prompt = no
default_md = sha256
x509_extensions = v3_req
distinguished_name = dn
```

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

spect the new certificate and verify

```
3:3c:5a:c4:
da:3d
```

If you are using Apache, then you can reference the above certificate in your configuration file like so:

```
<VirtualHost _default_:443>
    ServerName example.com
    ServerAlias www.example.com
    DocumentRoot /var/www/htdocs

    SSLEngine on
    SSLCertificateFile path/to/your/example.com.crt
    SSLCertificateKeyFile path/to/your/example.com.key
</VirtualHost>
```

Remember to restart your Apache (or Nginx, or IIS) server for the new certificate to take effect.

Share Follow

edited Dec 28 2018 at 18:59
Peter Mortensen
29.6k 21 98 124

answered May 13 2017 at 20:21
Drakes
21.8k 3 46 89

Works on macOS High Siera and Chrome 58 – Saqib Omer Dec 23 2017 at 11:58

I'm still not sure how the CN affects the overall setup? I'm attempting to run this as `localhost` or
for something like this. – DJ2 Apr 16 2018

ır 16 2018 at 23:11

23

I can`t comment so I add a separate answer. I tried to create a self-signed certificate for NGINX and it was easy, but when I wanted to add it to Chrome white list I had a problem. And my solution was to create a Root certificate and signed a child certificate by it.

So step by step. Create file **config_ssl_ca.cnf** Notice, config file has an option **basicConstraints=CA:true** which means that this certificate is supposed to be root.

> This is a good practice, because you create it once and can reuse.

```
[ req ]
default_bits = 2048

prompt = no
distinguished_name=req_distinguished_name
req_extensions = v3_req

[ req_distinguished_name ]
countryName=UA
stateOrProvinceName=root region
localityName=root city
organizationName=Market(localhost)
organizationalUnitName=roote department
commonName=market.localhost
emailAddress=root_email@root.localhost

[ alternate_names ]
DNS.1        = market.localhost
DNS.2        = www.market.localhost
DNS.3        = mail.market.localhost
DNS.4        = ftp.market.localhost
DNS.5        = *.market.localhost
```

ig_ssl.cnf.

```
organizationalUnitName=market place department
```

```
DNS.5         = *.market.localhost

[ v3_req ]
keyUsage=digitalSignature
basicConstraints=CA:false
subjectAltName = @alternate_names
subjectKeyIdentifier = hash
```

The first step - create Root key and certificate

```
openssl genrsa -out ca.key 2048
openssl req -new -x509 -key ca.key -out ca.crt -days 365 -config config_ssl_ca.cnf
```

The second step creates child key and file CSR - Certificate Signing Request. Because the idea is to sign the child certificate by root and get a correct certificate

```
openssl genrsa -out market.key 2048
openssl req -new -sha256 -key market.key -config config_ssl.cnf -out market.csr
```

Open Linux terminal and do this command

```
echo 00 > ca.srl
touch index.txt
```

The **ca.srl** text file containing the next serial number to use in hex. Mandatory. This file must be present and contain a valid serial number.

Last Step, create one more config file and call it **config_ca.cnf**

```
usign the ca command

o use in hex. Mandatory.
erial number.

ile must be present though

ill be placed. Mandatory.
```

```
policy = my_policy

# MOST IMPORTANT PART OF THIS CONFIG
copy_extensions = copy

[ my_policy ]
# if the value is "match" then the field value must match the same field in the
# CA certificate. If the value is "supplied" then it must be present.
# Optional means it may be present. Any fields not mentioned are silently
```

You may ask, why so difficult, why we must create one more config to sign child certificate by root. The answer is simple because child certificate must have a SAN block - Subject Alternative Names. If we sign the child certificate by "openssl x509" utils, the Root certificate will delete the SAN field in child certificate. So we use "openssl ca" instead of "openssl x509" to avoid the deleting of the SAN field. We create a new config file and tell it to copy all extended fields **copy_extensions = copy**.

```
openssl ca -config config_ca.cnf -out market.crt -in market.csr
```

The program asks you 2 questions:

1. Sign the certificate? Say "Y"
2. 1 out of 1 certificate requests certified, commit? Say "Y"

In terminal you can see a sentence with the word "Database", it means file index.txt which you create by the command "touch". It will contain all information by all certificates you create by "openssl ca" util. To check the certificate valid use:

```
openssl rsa -in market.key -check
```

3 Although, this process looks complicated, this is exactly what we need for .dev domain, as this domain does not support self-signed certificates and Chrome and Firefox are forcing HSTS. What I did is followed this steps, which is creating CA, creating a certificate and signing it with my CA and at the end trusting my CA in the browser. Thanks. – bajicdusko Apr 17 2020 at 23:08

1 Your common name is wrong. Not firstname/lastname. its your domain cn i.e. www.yoursite.com . see ssl.com/faqs/common-name – Jimmy MG Lim Aug 16 2020 at 8:54

1 no problem. there are some documents which also say name (yourname) which is a bit misleading. but common name should be the actual domain. in this sense it would be (your"domain"name) they are trying to say. when running thru with interactive method of creating the certs, it does say cn=domain example. so commonname should be domain – Jimmy MG Lim Aug 19 2020 at 9:20

UPD answer to resolve stackoverflow.com/questions/39270992/... – mrkiril Oct 26 2020 at 14:37

9

2017 one-liner:

```
openssl req \
-newkey rsa:2048 \
-x509 \
-nodes \
-keyout server.pem \
```

```
\
)) \
```

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

without having another configuration

ivate key and cert. You can move

02 answered Sep 20 2017 at 16:27

124 joemillervi
838 1 7 18

2 For Linux users you'll need to change that path for the config. e.g. on current Ubuntu `/etc/ssl/openssl.conf` works – declension Sep 27 2018 at 10:53

For a one-liner that doesn't require you to specify the openssl.cnf location, see: stackoverflow.com/a/41366949/19163 – vog Nov 26 2018 at 9:56

8

One-liner version 2017:

CentOS:

```
openssl req -x509 -nodes -sha256 -newkey rsa:2048 \
-keyout localhost.key -out localhost.crt \
-days 3650 \
-subj "CN=localhost" \
-reqexts SAN -extensions SAN \
-config <(cat /etc/pki/tls/openssl.cnf <(printf "\n[SAN]
\nsubjectAltName=IP:127.0.0.1,DNS:localhost"))
```

Ubuntu:

```
openssl req -x509 -nodes -sha256 -newkey rsa:2048 \
-keyout localhost.key -out localhost.crt \
-days 3650 \
-subj "/CN=localhost" \
-reqexts SAN -extensions SAN \
-config <(cat /etc/ssl/openssl.cnf <(printf "\n[SAN]
\nsubjectAltName=IP:127.0.0.1,DNS:localhost"))
```

Edit: added prepending Slash to 'subj' option for Ubuntu.

answered Nov 28 2017 at 10:11
user327843
462 6 17

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

7

One liner FTW. I like to keep it simple. Why not use one command that contains ALL the arguments needed? This is how I like it - this creates an x509 certificate and its PEM key:

```
openssl req -x509 \
 -nodes -days 365 -newkey rsa:4096 \
 -keyout self.key.pem \
 -out self-x509.crt \
 -subj "/C=US/ST=WA/L=Seattle/CN=example.com/emailAddress=someEmail@gmail.com"
```

That single command contains all the answers you would normally provide for the certificate details. This way you can set the parameters and run the command, get your output - then go for coffee.

>> More here <<

Share Follow

edited Dec 28 2018 at 19:01
Peter Mortensen
29.6k 21 98 124

answered Sep 11 2017 at 15:14
OkezieE
4,735 3 23 25

1 All the arguments except for SANs... @vog's answer covers that as well (and predate this) (This has a more complete "Subject" field filled in though...) (Not a big fan of the one year expiry either) – Gert van den Berg Dec 18 2018 at 10:38

vog's answer. Linked, because usernames are neither unique nor immutable. "vog" could change to "scoogie" at any point in time. – jpaugh Jan 17 at 3:45

6

You have the general procedure correct. The syntax for the command is below.

```
openssl req -new -key {private key file} -out {output file}
```

However, the warnings are displayed, because the browser was not able to verify the identify by validating the certificate with a known Certificate Authority (CA).

As this is a self-signed certificate there is no CA and you can safely ignore the warning and proceed. Should you want to get a real certificate that will be recognizable by anyone on the public Internet then the procedure is below.

1. Generate a private key
2. Use that private key to create a CSR file
3. Submit CSR to CA (Verisign or others, etc.)
4. Install received cert from CA on web server
5. Add other certs to authentication chain depending on the type cert

I have more details about this in a post at *Securing the Connection: Creating a Security Certificate with OpenSSL*

Share Follow

edited Dec 28 2018 at 18:45
Peter Mortensen
29.6k 21 98 124

answered Apr 2 2015 at 6:15
nneko
688 7 10

6

Generate keys

I am using `/etc/mysql` for cert storage because `/etc/apparmor.d/usr.sbin.mysqld` contains `/etc/mysql/*.pem r`.

```
sudo su -
cd /etc/mysql
openssl genrsa -out ca-key.pem 2048;
openssl req -new -x509 -nodes -days 1000 -key ca-key.pem -out ca-cert.pem;
openssl req -newkey rsa:2048 -days 1000 -nodes -keyout server-key.pem -out
server-req.pem;
openssl x509 -req -in server-req.pem -days 1000 -CA ca-cert.pem -CAkey ca-
key.pem -set_serial 01 -out server-cert.pem;
openssl req -newkey rsa:2048 -days 1000 -nodes -keyout client-key.pem -out
client-req.pem;
openssl x509 -req -in client-req.pem -days 1000 -CA ca-cert.pem -CAkey ca-
key.pem -set_serial 01 -out client-cert.pem;
```

Add configuration

`/etc/mysql/my.cnf`

```
[client]
ssl-ca=/etc/mysql/ca-cert.pem
ssl-cert=/etc/mysql/client-cert.pem
ssl-key=/etc/mysql/client-key.pem

[mysqld]
ssl-ca=/etc/mysql/ca-cert.pem
```

/error.log

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

cate file if it is not in apparmors

, save all our certificates as `.pem`

d by default by apparmor (or modify

er you stored them.)

efault `rsa:2048` format

```
mysql -u root -p
mysql> show variables like "%ssl%";
+---------------+----------------------------+
| Variable_name | Value                      |
+---------------+----------------------------+
| have_openssl  | YES                        |
| have_ssl      | YES                        |
| ssl_ca        | /etc/mysql/ca-cert.pem     |
| ssl_capath    |                            |
| ssl_cert      | /etc/mysql/server-cert.pem |
| ssl_cipher    |                            |
| ssl_key       | /etc/mysql/server-key.pem  |
+---------------+----------------------------+
```

- Verifying a connection to the database is SSL encrypted:

Verifying connection

When logged in to the MySQL instance, you can issue the query:

```
show status like 'Ssl_cipher';
```

If your connection is not encrypted, the result will be blank:

```
mysql> show status like 'Ssl_cipher';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Ssl_cipher    |       |
+---------------+-------+
1 row in set (0.00 sec)
```

ing for the cypher in use:

ssl'):

Alternate link: Lengthy tutorial in *Secure PHP Connections to MySQL with SSL*.

Share Follow

edited Dec 28 2018 at 18:51

community wiki
3 revs, 3 users 80%
ThorSummoner

-1; this is largely tangential to the question asked, and also does a bad job of making clear where its quotes are from. – Mark Amery Sep 29 2019 at 17:12

This shows provisioning CA, Server/Client certs signed by the CA, configure them for reading by mysqld on a host with apparmor. It exemplifies a rather useless case of hosting the ca, server, and client on the same machine, and dangerously exposing that ca's authority to the mysqld process. This setup doesn't really make sense other than to test ssl configuration in a test environment. For operating an internal CA, I would recommend the gnuttls toolchain over openssl help.ubuntu.com/community/GnuTLS and having a good understanding of tls before working around the mysqld+apparmor case – ThorSummoner Sep 29 2019 at 20:57

5

`openssl` allows to generate self-signed certificate by a single command (`-newkey` instructs to generate a private key and `-x509` instructs to issue a self-signed certificate instead of a signing request)::

```
openssl req -x509 -newkey rsa:4096 \
-keyout my.key -passout pass:123456 -out my.crt \
-days 365 \
-subj /CN=localhost/O=home/C=US/emailAddress=me@mail.internal \
-addext "subjectAltName = DNS:localhost,DNS:web.internal,email:me@mail.internal" \
-addext keyUsage=digitalSignature -addext extendedKeyUsage=serverAuth
```

You can generate a private key and construct a self-signing certificate in separate steps::

```
openssl genrsa -out my.key -passout pass:123456 2048

openssl req -x509 \
-key my.key -passin pass:123456 -out my.csr \
-days 3650 \
-subj /CN=localhost/O=home/C=US/emailAddress=me@mail.internal \
-addext "subjectAltName = DNS:localhost,DNS:web.internal,email:me@mail.internal" \
-addext keyUsage=digitalSignature -addext extendedKeyUsage=serverAuth
```

Review the resulting certificate::

```
openssl x509 -text -noout -in my.crt
```

Java `keytool` creates PKCS#12 store::

```
 \
ity 3650 \

mail.internal'

\
```

Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies

Customize settings

attributes from CLI. I don't like to mess with config files ((

4

As has been discussed in detail, self-signed certificates are not trusted for the Internet. You can add your self-signed certificate to many but not all browsers. Alternatively you can become your own certificate authority.

The primary reason one does not want to get a signed certificate from a certificate authority is cost -- Symantec charges between $995 - $1,999 per year for certificates -- just for a certificate intended for internal network, Symantec charges $399 per year. That cost is easy to justify if you are processing credit card payments or work for the profit center of a highly profitable company. It is more than many can afford for a personal project one is creating on the internet, or for a non-profit running on a minimal budget, or if one works in a cost center of an organization -- cost centers always try to do more with less.

An alternative is to use certbot (see about certbot). Certbot is an easy-to-use automatic client that fetches and deploys SSL/TLS certificates for your web server.

If you setup certbot, you can enable it to create and maintain a certificate for you issued by the Let's Encrypt certificate authority.

I did this over the weekend for my organization. I installed the required packages for certbot on my server (Ubuntu 16.04) and then ran the command necessary to setup and enable certbot. One likely needs a DNS plugin for certbot - we are presently using DigitalOcean though may be migrating to another service soon.

Note that some of the instructions were not quite right and took a little poking and time with Google to figure out. This took a fair amount of my time the first time but now I think I could do it in minutes.

prompted to input the path to your
ring to is the Applications & API
have or generate a personal access
65 character hexadecimal string.
rver from which you are running
omments start with #). The seccond

```
ccddddeeeeffff
```

DigitalOcean's API, it was pretty
that one does not have to setup a
ain and sub-domain that one wants
that required the credentials INI file
cean.

date and to trigger renewal when it is 30 days or less until it expires. I will then add this script to cron and run it once per day.

Here is the command to read your certificate's expiration date:

```
root@prod-host:~# /usr/bin/openssl x509 -enddate -noout -in path-to-certificate-
pem-file
notAfter=May 25 19:24:12 2019 GMT
```

Share Follow

answered Feb 25 2019 at 21:52

Peter Jirak Eldritch
513 1 5 13

3

this worked for me

```
openssl req -x509 -nodes -subj '/CN=localhost'  -newkey rsa:4096 -keyout ./sslcert
/key.pem -out ./sslcert/cert.pem -days 365
```

server.js

```
var fs = require('fs');
var path = require('path');
var http = require('http');
var https = require('https');
var compression = require('compression');
var express = require('express');
var app = express();

app.use(compression());
app.use(express.static(__dirname + '/www'));

app.get('/*', function(req,res) {
  res.sendFile(path.join(__dirname+'/www/index.html'));
});

// your express configuration here

var httpServer = http.createServer(app);
var credentials = {
    key: fs.readFileSync('./sslcert/key.pem', 'utf8'),
    cert: fs.readFileSync('./sslcert/cert.pem', 'utf8')
};
var httpsServer = https.createServer(credentials, app);

httpServer.listen(8080);
httpsServer.listen(8443);
```

answered Dec 5 2020 at 6:16
Leonardo Pineda
844 8 9

0

Generate a key without password and certificate for 10 years, the short way:

```
openssl req  -x509 -nodes -new  -keyout server.key -out server.crt -days 3650 -subj
"/C=/ST=/L=/O=/OU=web/CN=www.server.com"
```

Share Follow

answered Dec 27 2021 at 13:35

Maoz Zadok
3,529 2 23 34

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.