Ачилов Р.

# Построение защищенных корпоративных сетей

ДМК Пресс
издательство

**УДК 004.732:004.056**
**ББК 32.973.202**
**А97**

А97 **Ачилов Р. Н.**

Построение защищенных корпоративных сетей. – М.: ДМК Пресс, 2013. – 250 с.: ил.

**ISBN 978-5-94074-884-7**

В книге рассказано обо всем, что необходимо для построения защищенной от внешних воздействий корпоративной сети – о том, как создать собственный удостоверяющий центр для выдачи SSL-сертификатов, как выдавать, отзывать, преобразовывать и просматривать сертификаты. Как установить SSL-сертификат в ОС или браузер, как его использовать, работая с защищенным ресурсом и какие ошибки при этом возникают.

Описывается, как с помощью сертификатов защищить корпоративную электронную почту на всех этапах ее передачи – от почтовой программы пользователя до сервера получателя; как установить веб-интерфейс к хранимой на сервере почте, позволяющий просматривать ее в защищенном режиме с любой точки мира.

Также уделено внимание защите служебных коммуникаций, в частности подключения из скриптов для управления серверами. В книге приводится большое число примеров конфигурационных файлов с подробным пояснением параметров, а также скриптов на языке Bourne Shell 1.x.

Издание предназначено для системных и сетевых администраторов UNIX, администраторов средств информационной безопасности.

УДК 004.732:004.056
ББК 32.973.202



ISBN 978-5-94074-884-7

# ГЛАВА 1 Корпоративный «паспортный стол»

Большая работа всегда начинается с большого... нет, не перекура, а с большого плана. Итак:

- филиалы при подключении к VPN должны быть уверены в том, что «та» сторона – это именно тот компьютер, за который он себя выдает;
- почтовые серверы при пересылке писем между узлами VPN должны быть уверены в том, что получатель – именно тот, за которого он себя выдает;
- при подключении к веб-интерфейсу почты сервер должен быть уверен, что клиент имеет право к нему подключаться, а клиент – в том, что сервер – именно тот сервер, за которого он себя выдает.

Да и вообще при установлении защищенного соединения первоначально обе стороны должны убедиться в том, что противоположная сторона – это именно та, которая нужна нам, а не подстава.

Каким образом стороны могут убедиться в том, что с той стороны – нужный тебе абонент, а не злобный хакер? Либо стороны предварительно обмениваются какой-либо информацией по отдельному, независимому и заведомо неподконтрольному хакеру каналу (например, по сотовому созвониться), либо они доверяют некоторому третьему лицу, которое заранее выдало нечто, что бы подтверждало то, что ты – это ты, причем выдало обеим сторонам. В России такой третьей стороной является УФМС, иначе говоря – паспортная служба. Предполагается, что доверие сторон к третьему лицу абсолютно – если, допустим, на паспорте стоит печать УФМС и фото,

значит, он подлинный. Вот поначалу мы и займемся тем, что создадим собственный «паспортный стол», а потом объявим, что в масштабах компании наличие сертификата, подписанного сертификатом нашего УЦ, означает его подлинность.

УЦ будет размещаться по адресу 212.20.5.1, на компьютере www.deltahw.ru (это чтобы не путаться при создании сертификатов).

И точно так же, как наличие паспорта у человека означает предоставление ему некоторых привилегий, так и наличие сертификата будет означать предоставление привилегий с точки зрения программы. Правда, привилегия будет всего одна – предоставить доступ к данным.

# 1.1. Создание удостоверяющего центра

Прежде чем выдавать паспорта, надо построить паспортный стол. Прежде чем выдавать сертификаты, необходимо будет создать удостоверяющий центр (УЦ). Правда, мы будем пользоваться его англоязычным наименованием – CA, Center of Authority – так будет понятнее при чтении документации и конфигурационных файлов. В UNIX для создания не нужно ничего запускать: собственно весь CA – это несколько файлов и каталогов, ключ самого CA, сертификат CA и список отозванных сертификатов. Ну и конечно же конфигурационный файл openssl.cnf, настройки которого будут играть чрезвычайно важную роль. Собственно говоря, openssl.cnf и ключ корневого сертификата CA плюс его индексный файл и есть CA, потому что все остальное – просто обвязка.

## 1.1.1. Файловая структура CA

Сначала определяемся с путями к файлам. Нам понадобится отдельный каталог, доступный только пользователю root, в котором будут находиться:

- принятые нами запросы на сертификацию (CSR);
- подписанные нами сертификаты (CRT);
- список отозванных сертификатов (CRL);
- закрытый ключ нашего собственного сертификата.

Также здесь будут находиться одна из копий сертификата нашего CA и рабочие файлы CA. Имена файлов можно использовать произвольные, какого-либо стандарта на это нет. Я использую структуру каталогов, которой когда-то придерживался модуль mod_ssl веб-сервера Apache 1.x, хотя конечно же можно использовать любые другие.

Структура каталогов следующая:

- /usr/local/share/ca-dhw. Как видно из листинга ниже – корневой каталог CA;
- подкаталоги:
  - ssl.crl – для хранения CRL;
  - ssl.crt – для хранения подписанных сертификатов;
  - ssl.csr – для хранения запросов на сертификацию;
  - ssl.key – для хранения ключей. Как правило, в этом каталоге лежит самый главный ключ CA.

Создаем каталоги:

```
# mkdir /usr/local/share/ca-dhw
# cd /usr/local/share/ca-dhw
# mkdir ssl.crl
# mkdir ssl.crt
# mkdir ssl.csr
# mkdir ssl.key
# cd ..
# chmod -R 0700 ca-dhw
```

Создаем служебные файлы:

```
# cd ca-dhw
# touch index.txt
# echo '01' > serial
```

Файл index.txt – это файл списка подписанных нами сертификатов от самого начала. Он будет расти в размере с каждым выданным сертификатом. Несмотря на то, что это просто текст, лучше его текстовыми редакторами не править – ну разве что удалить строчку о заведомо неверном сертификате, разумеется, после удаления самого сертификата.

Формат этого файла очень простой. На каждый сертификат отводится по одной строке. В первом столбце указывается «V» для действующего сертификата (видимо, от *vita* – жизнь (лат.)) или «R» – для

отозванного (а это, видимо, от слова *Revoked* – отозванный). Во втором столбце указывается дата окончания срока действия сертификата. Формат даты текстовый, в виде строки ГГММДДЧЧММССZ (последний символ – не обозначение, это именно буква Z – это строка типа `ASN.1 UTCTime`). В третьем столбце, если он есть, указываются дата отзыва сертификата и причина отзыва через запятую. Причины отзыва мы рассмотрим, когда дойдем до рассмотрения CRL. В четвертом столбце указывается серийный номер сертификата в шестнадцатеричной форме без префикса 0x. Пятый столбец почему-то всегда содержит слово «unknown». Ну и в шестом столбце приводится полное значение поля `Subject` сертификата. Только этот файл да еще файл serial – вот и все, что нужно поправить в том случае, если вдруг выдали ошибочный сертификат.

Файл serial содержит серийный номер следующего сертификата, который будет выдан. В этом же каталоге обычно образуется файл index.txt.attr, но этот файл служебный, его содержимое неинтересно.

### 1.1.2. Конфигурационный файл openssl.cnf

Теперь займемся правкой конфигурационного файла openssl.cnf. Это очень важный этап, поэтому файл будет разбираться по секциям. Не все секции для нас важны, но тем не менее из песни слов не выкинешь. О тех секциях, где ничего нет необходимости менять, мы просто упомянем вскользь. Чтобы получить из этого готовый конфигурационный файл, нужно просто объединить все секции в один файл. Впрочем, особой необходимости это делать нет – OpenSSL поставляется всегда с готовым файлом openssl.cnf, который уже можно править.

Начало, то есть задание `HOME`, `oid_section` и секция `[new_oids]`. В секции `[new_oids]` должны размещаться новые OID'ы, но у нас их нет, поэтому секция пустая.

```
HOME                    = .
RANDFILE                = $ENV::HOME/.rnd
oid_section             = new_oids
[new_oids]
```

Секция с параметрами CA. Ее мы рассмотрим весьма подробно – в ней множество важных вещей. В секции `[ca]` указывается единственный параметр – имя секции с настройками. Зачем так сделано, непонятно.

```
[ ca ]
default_ca              = CA_default

[ CA_default ]
Dir                     = /usr/local/share/ca-dhw
Certs                   = $dir/ssl.crt
crl_dir                 = $dir/ssl.crl
database                = $dir/index.txt
new_certs_dir           = $dir/ssl.crt
certificate             = $dir/caserv.crt
serial                  = $dir/serial
crl                     = $dir/cacrl.crl
private_key             = $dir/ssl.key/caserv.key
RANDFILE                = $dir/ssl.key/.rand
x509_extensions         = usr_cert
name_op                 = ca_default
cert_opt                = ca_default
copy_extensions         = copy
crl_extensions          = crl_ext
default_days            = 365
default_crl_days        = 30
default_md              = md5
preserve                = no
policy                  = policy_match
```

Пожалуй, самая важная секция файла. Впрочем, в этом файле все секции важные.

*dir* – указывает корень всего CA.

*certs* – каталог для хранения выданных сертификатов.

*crl_dir* – каталог для хранения созданных CRL.

*database* – индексный файл для хранения списка сертификатов. Должен существовать к началу процесса выдачи сертификатов.

*new_certs_dir* – куда будут помещаться новые сертификаты по умолчанию.

*certificate* – имя файла сертификата CA.

*serial* – имя служебного файла, содержащего следующий номер сертификата. Должен существовать к началу процесса выдачи сертификатов и содержать правильный номер.

*crl* – имя файла CRL.

*private_key* – имя файла ключа сертификата CA.

*RANDFILE* – файл рэндома для работы генераторов случайных чисел.

*x509_extensions* – имя секции с описанием расширений сертификата.

*name_opt* – имя секции с параметрами отображения имен.

*cert_opt* – имя секции с параметрами полей сертификатов.

*copy_extensions* – если задано `copy`, то в сертификат копируются все отсутствующие там расширения из CSR. Если задано `copyall`, то в случае присутствия расширения в сертификате оно сначала удаляется, а потом копируется из CSR.

*crl_extensions* – имя секции с описанием расширений CRL.

*default_days* – срок действия сертификата по умолчанию.

*default_crl_days* – срок действия CRL по умолчанию. Обратите внимание, он достаточно короткий!

*default_md* – message digest для сертификата по умолчанию. Другие варианты: `sha1`, `mdc2`.

*preserve* – не сохранять порядок полей, заданный в поле `Subject` при создании сертификата.

*policy* – имя секции, описывающей элементы DN. Если в секции элемент DN не описан, он просто молча удаляется.

Секции политик для создания сертификатов CA называются `[policy_match]`, а для создания обычных сертификатов – `[policy_anything]`. В них ничего не меняется. Различные варианты параметра – `match`, `supplied`, `optional` – указывают на то, является данный атрибут обязательным или нет. Не думаю, что вам когда-либо придется править эти секции, разве только возникнет необходимость добавить какой-либо атрибут к запросу атрибутов при создании CSR.

```
[ policy_match ]
countryName                     = match
stateOrProvinceName             = match
organizationName                = match
organizationalUnitName          = optional
commonName                      = supplied
emailAddress                    = optional
[ policy_anything ]
countryName                     = optional
stateOrProvinceName             = optional
localityName                    = optional
organizationName                = optional
organizationalUnitName          = optional
commonName                      = supplied
emailAddress                    = optional
```

Дальше идет секция `[req]`, которая задействуется как раз тогда, когда создается CSR. Секция небольшая, но важная:

```
[ req ]
default_bits                    = 2048
default_keyfile                 = privkey.pem
distinguished_name              = req_distinguished_name
attributes                      = req_attributes
x509_extensions                 = v3_ca
string_mask                     = nombstr
req_extensions                  = v3_req
```

*default_bits* – размер ключа по умолчанию в битах.

*default_keyfile* – имя файла ключа по умолчанию.

*distinguished_name* – имя секции, которая описывает правила запроса DN у пользователя при формировании CSR.

*attributes* – имя секции, которая описывает дополнительные атрибуты CSR.

*x509_extensions* – имя секции, которая описывает расширения, добавляемые в сертификат CA или самоподписанный сертификат.

*input_password, output_password* – здесь может быть задан пароль для открытия ключа или для формирования ключа. Если он не задан здесь и не отключен другим ключом, то запрашивается у пользователя.

*string_mask* – маска, накладываемая на строковые значения. Строки данных в сертификатах могут быть разного типа и содержать различные наборы символов. Всего типов строк четыре: `PrintableString`, `T16String (TelelexString)`, `BMPString` и `UTF8String`. Не все браузеры поддерживают работу с `BMPString` и `UTF8String`, которые позволяют закодировать буквы национальных алфавитов. Значением параметра может быть одна из перечисленных ниже строк:

```
default: PrintableString, T61String, BMPString
pkix: PrintableString, BMPString
utf8only: только UTF8Strings
nombstr : PrintableString, T61String (но не BMPStrings или UTF8Strings).
```

Обычно используется *nombstr* для гарантии совместимости, потому что некоторые программы могут аварийно завершаться, обнаружив строку типа `BMPString` или `UTF8String`. И, хотя эти типы позволяют использовать в сертификатах символы национальных алфавитов, я бы не рекомендовал ими пользоваться – сертификат выдается надолго, и пользоваться им будут в разных местах разными программами.

*req_extensions* – имя секции, в которой описываются включаемые в CSR расширения.

Далее идет секция `[req_distinguished_name]`, которую обычно правят на предмет умолчаний в значениях, запрашиваемых при формировании DN. В ней описываются запрашиваемые поля, их значения по умолчанию, минимальные и максимальные длины.

```
[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = RU
countryName_min                 = 2
countryName_max                 = 2
stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Novosibirsk region
localityName                    = Locality Name (eg, city)
localityName_default            = Novosibirsk
0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = DeltaHardware Ltd
organizationalUnitName          = Organizational Unit Name (eg, section)
commonName                      = Common Name (eg, YOUR name)
commonName_max                  = 64
emailAddress                    = Email Address
emailAddress_max                = 64
```

Конечно же, вы уже обратили внимание на то, что запрос `organizationName` записан в каком-то странном виде с нулем впереди. Это означает, что теоретически можно было бы запросить еще одно `organizationName` и для его запроса использовать форму **1.organizationName**. Данной возможностью – иметь в сертификате несколько атрибутов одного типа, перечисляемых через `0.<имя_атрибута>`, `1.<имя_атрибута>`, – редко кто пользуется, но тем не менее она есть.

При создании запроса на сертификат пользователю на консоль будет выдано сообщение **Country Name (2 letter code) [RU]:**, и в ответ будет ожидаться ввод двух символов. Исправите параметр `countryName` – и будет выведен другой текст, исправите `countryName_default` – изменится значение по умолчанию в квадратных скобках, исправите `countryName_min` или `countryName_max` – изменятся минимальная и максимальная длины ввода. Разумеется, не обязательно описывать все параметры: если что-то не описано, то принимается значение по умолчанию. Исправляют обычно значения по умолчанию и иногда длины, хотя, с моей точки зрения, длины достаточные. Максимальные длины полей для данных, вводимых при создании запроса на сертификат, приведены в табл. 1.1

**Таблица 1.1.** Максимальные длины полей элементов атрибута Subject

| Имя поля CSR | Максимальная длина поля в байтах |
|---|---|
| Country | 2 |
| CommonName | 64 |
| Locality | 128 |
| StateOrProvince | 128 |
| OrganizationalName | 64 |
| OrganizationalNameUnit | 64 |
| Email | 128 |

В конфигурационном файле приведены описания не всех возможных полей DN, описание всех возможных полей заняло бы слишком много места, приведены только наиболее нужные. Если вам для каких-либо целей нужно расширить количество запрашиваемых атрибутов, это можно сделать, добавив нужные поля к секции. Какие есть еще атрибуты, можно посмотреть в [1]. Например, если добавить в секцию приведенные ниже строки, то при генерации CSR будет запрашиваться **Given Name**.

```
givenName               = Given Name
givenName_max           = 128
```

За ней идет секция `[req_attributes]`, которая описывает необязательные атрибуты CSR. Обычно ее никто не правит.

```
[ req_attributes ]
challengePassword       = A challenge password
challengePassword_min   = 4
challengePassword_max   = 20
unstructuredName        = An optional company name
```

За `[req_attributes]` идет секция `[usr_cert]` с расширениями, добавляемыми в сертификат по умолчанию. Как правило, она не используется, потому что требуется обычно подписывать сертификаты SSL-серверов и сертификаты SSL-клиентов. Поэтому секция чаще всего полупустая и ненастроенная. Хотя конечно же никто не мешает вам настроить ее и пользоваться.

```
[ usr_cert ]
basicConstraints        = CA:FALSE
keyUsage                = digitalSignature, keyEncipherment
nsComment               = "OpenSSL Default Generated Certificate"
```

```
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer:always
```

За `[usr_cert]` идут собственно секции, описывающие атрибуты для создаваемого сертификата сервера, – `[ssl_server]` и клиента – `[ssl_client]`. Ниже разбирается секция сервера, секция клиента точно такая же, только значение атрибута `nsCertType` будет другим.

```
[ ssl_server ]
basicConstraints        = CA:FALSE
nsCertType              = server, email
keyUsage                = digitalSignature, keyEncipherment
extendedKeyUsage        = serverAuth, emailProtection
nsComment               = "OpenSSL Certificate for SSL Web Server"
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid,issuer:always
authorityInfoAccess     = caIssuers;URI:http://certs.deltahw.ru
issuerAltName           = issuer:copy, URI:http://certs.deltahw.ru/certs/caserv.crt
crlDistributionPoints   = URI:http://certs.deltahw.ru/certs/cacrl.crl
nsCaRevocationUrl       = http://certs.deltahw.ru/certs/cacrl.crl
```

*basicConstraints* указывает, что это не сертификат CA. Больше он ни для чего не нужен – он только указывает, это сертификат CA или нет.

*nsCertType* – устаревший атрибут, указывавший тип создаваемого сертификата. Этот атрибут указывать было не обязательно. Возможные значения этого атрибута: client, server, email, objsign, reserved, sslCA, emailCA, objCA. Все эти значения задают различные типы сертификата, наиболее употребимые – это client, server и email, которые задают соответственно сертификат SSL-клиента, SSL-сервера и сертификат для защиты электронной почты. Фактически этот атрибут дублирует атрибут `extendedkeyUsage`.

*keyUsage* – указывает, для чего будет использоваться сертификат, сгенерированный по данному запросу. Возможные значения атрибута: digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly and decipherOnly. Практически все значения имеют достаточно понятные названия – шифрование ключей и данных, подпись сертификатов и CRL. Атрибут nonRepudiation устанавливается, когда ключ используется для проверки сигнатур, используемых для предоставления «безотказного» сервиса, который не позволит второй стороне отказаться от факта участия в обмене данными. Такое вот юридически-полицей-

ское толкование. Ну а KeyAgreement используется для согласования сертификатов.

*extendedKeyUsage* – фактически настоящее применение сертификата описывается здесь. Возможные значения атрибута:

- serverAuth – аутентификация сервера;
- clientAuth – аутентифкация клиента;
- codeSigning – подпись программного кода;
- emailProtection – защита электронной почты;
- ipsecEndSystem – сервер IPSec;
- ipsecTunnel – туннель IPSec;
- ipsecUser – оконечный пользователь IPSec;
- timeStamping – простановка меток времени.

Видимое расхождение с атрибутом `KeyUsage` объясняется просто – этот атрибут относится к расширенным, а атрибут `KeyUsage` – к основным.

*nsComment* – просто комментарий.

*subjectKeyIdentifier* – идентификатор ключа субъекта. Обязательный атрибут. Может быть задан либо словом `hash` – в таком случае в качестве идентификатора будет взят 160-битный SHA-1 хэш от общего ключа субъекта или же задан вручную. Задавать вручную категорически не рекомендуется.

*authorityKeyIdentifier* – идентификатор ключа CA. Обязательный атрибут. Может быть задан двумя параметрами – `keyid` и `issuer`, каждый может сопровождаться опцией `always`. При указании `keyid` в сертификат копируется идентификатор ключа субъекта родительского сертификата. При указании `issuer` в сертификат копируются атрибут `Subject` и серийный номер ключа CA. Если указана опция `always`, то при невозможности скопировать атрибут процесс создания сертификата завершается аварийно.

*authorityInfoAccess* – дополнительная информация о CA, выдавшем сертификат. Указывается в формате `OID;<имя или ссылка>`, где OID должен иметь значение `caIssuers` или `OSCP`, а формат поля `<имя или ссылка>`, который на самом деле то же самое, что SAN, будет рассмотрен после полного примера конфигурационного файла openssl.cnf.

*subjectAlternativeName* (SAN) – дополнительное имя субъекта. Несмотря на то что в примере этой строки нет, мы рассмотрим ее здесь. Дело в том, что изначально в конфигурационном файле она была, но впоследствии была удалена, потому что, оказывается, нельзя объеди-

нять атрибуты, заданные в конфигурационном файле, и атрибуты, задаваемые в CSR. Можно либо ничего не делать с атрибутами из CSR (и так и делается по умолчанию), либо скопировать то, что отсутствует в сертификате (copy), либо скопировать все, удалив предварительно дублирующиеся расширения (copyall). Поэтому атрибут был удален, и задаваться он будет в CSR. Задаваться SAN может большим количеством разнообразных способов, чему будет посвящен отдельный раздел.

*issuerAlternativeName* – дополнительное имя CA. Может задаваться параметром `issuer` с опцией `copy`, что будет означать скопировать в сертификат SAN из сертификата CA, а также всеми возможными способами задания SAN. В примере задана ссылка на сертификат CA через опцию HTTP.

*crlDistributionPoints*, а также *nsCARevocationUrl* – задают точку распространения CRL. Причем первый атрибут стандартный, задает ее в виде URI, а второй, устаревший, но еще повально много где используемый, задает ее просто в виде ссылки. Оба атрибута указывают непосредственно на файл CRL.

Секция [ssl_client], на основании которой генерируется сертификат клиента, ничем практически не отличается от секции сервера, отличаются только ключевые параметры `nsCertType`, `extendedKeyUsage`:

```
[ ssl_client ]
basicConstraints            = CA:FALSE
nsCertType                  = client, email
keyUsage                    = digitalSignature, keyEncipherment
extendedKeyUsage            = clientAuth, emailProtection
nsComment                   = "OpenSSL Certificate for SSL Web Server"
subjectKeyIdentifier        = hash
authorityKeyIdentifier      = keyid,issuer:always
authorityInfoAccess         = caIssuers;URI:http://certs.deltahw.ru
issuerAltName               = issuer:copy, URI:http://certs.deltahw.ru/certs/caserv.crt
crlDistributionPoints       = URI:http://certs.deltahw.ru/certs/cacrl.crl
nsCaRevocationUrl           = http://certs.deltahw.ru/certs/cacrl.crl
```

Далее следует небольшая секция `[v3_req]`, которая как раз и описывает атрибуты CSR:

```
[ v3_req ]
basicConstraints            = CA:FALSE
keyUsage                    = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName              = ${ENV::SAN}
```

Вот именно здесь появляется атрибут *subjectAlternativeName*, про формат которого мы поговорим, когда закончим анализировать конфигурационный файл openssl. Остальные параметры нам уже знакомы. Для задания собственно SAN нужно задавать его в переменной окружения и запускать программу генерации сертификатов.

Ну и последними идут секции с параметрами создаваемых сертификатов CA и CRL – `[v3_ca]` и `[crl_ext]`. В них нет никаких особенных параметров – все то же самое, что и в других секциях. Ну, разве что значения параметров `keyUsage` и `nsCertType` соответствуют тому, что создается сертификат для CA, а не простой сертификат, да `BasicConstraints` установлен нужным образом.

```
[ v3_ca ]
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer:always
basicConstraints        = CA:true
keyUsage                = cRLSign, keyCertSign
nsCertType              = sslCA, emailCA
nsComment               = "OpenSSL certificate for Certificate Authority"
subjectAltName          = email:copy

[ crl_ext ]
issuerAltName           = issuer:copy
authorityKeyIdentifier  = keyid:always,issuer:always
```

Здесь, правда, нас поджидает очень интересная и вовсе уж неожиданная засада. В атрибуте `subjectAltName` мы задали использование переменной окружения, которая может быть определена либо не определена. Засада же состоит в том, что openssl *каждый раз* при вызове читает *весь* конфигурационный файл и подставляет значения всех переменных, даже если они находятся в заведомо неиспользуемых секциях. Не спрашивайте меня, почему так. Ну а поскольку переменная не определена, то вы можете неожиданно для себя увидеть вот такое:

**>nslookup**

```
Auto configuration failed
676335936:error:0E065068:configuration file routines:STR_COPY:variable
has no value:/usr/src/secure/lib/libcrypto/../../../crypto/openssl/
crypto/conf/conf_def.c:626:line 224
```

Выходов из этой ситуации два – либо определить переменную SAN в стартовом скрипте своего шелла, либо создать два конфигурационных файла. Переменная SAN должна быть определена не просто

каким-либо текстом, а именно по правилам задания SAN, описанным в следующем разделе, поэтому мне не подошел этот путь, не люблю я задания фиктивных переменных – потом очень быстро забывается, что это за переменная и для чего она тут была введена. В разделе 1.1.5 приведены скрипты по управлению CA, в которых при запросе CSR на openssl.cnf накладывается небольшой патч, а после генерации CSR-файл возвращается в исходное состояние, поскольку патч нужен только для генерации CSR.

### 1.1.3. Задание subjectAltName

Атрибут `subjectAltName` (SAN, альтернативное имя субъекта) может быть задан столь большим количеством вариантов со столь разным представлением данных, что мы будем рассматривать его отдельно ото всего. Нам пригодится это описание и при задании многих других атрибутов, которые задаются в той же форме, что и SAN.

Общий формат задания атрибута = «<ключ>:<значение>,<ключ>:<значение>...». При этом в самом значении могут присутствовать двоеточия, а в ключах – еще и опции ключей.

В качестве ключа могут использоваться следующие слова:

*email*. Значением является допустимый адрес электронной почты. Допустимым значением является также слово `copy`, которое означает копирование в атрибут адреса электронной почты из поля `Subject`. Использование интернет-формата представления адреса (root.deltahw.ru вместо root@deltahw.ru) недопустимо. Например: `email:root@deltahw.ru`.

*DNS*. Значением является DNS-имя узла. Пустая строка недопустима. Например: `DNS:www.deltahardware.ru`.

*URI*. Значением является собственно URI, то есть Uniform Resource Locator – ссылка на ресурс. Например: `URI: http://www.deltahw.ru/certs/index.html`. Ссылка, включаемая в URI, должна включать в себя протокол (http, ftp и т. д.) и не может быть относительной ссылкой.

*IP*. Значением является IP-адрес узла. Например: `IP:212.20.5.1`.

*dirName*. Значением является имя секции, в которой на каждой строке задается по одному параметру атрибута `Subject`. Например:

```
subjectAltName=dirName:dir_sect

[dir_sect]
```

```
C=RU
O=DeltaHardware Ltd.
OU=Web Server
CN=www.deltahw.ru
```

*RID*. Значением является идентификатор зарегистрированного объекта (Registered Identifier). Например: `RID:1.3.6.5.5.7.8`.

*otherName*. Значением является RID, за которым через точку с запятой следует стандартная строка, созданная в формате `ASN1_generate_nconf`. Полное описание этого формата достаточно объемное, но коротко можно сказать так: <тип>:<значение>, где <тип> – BOOL, INT, OID, UTC, GENTIME, OCT, BITSTR, IA5, UTF8, BMP, PRINTABLE, VISIBLE, T61, SET, а <значение> – его значение. Большинство типов самоочевидно. OID – идентификатор объекта в цифровой нотации (1.2.3.4), GENTIME – время, где год выражен 4 цифрами (ГГГГММДДЧЧММССZ), OCT – octet string (строка двоичных данных), IA5, PRINTABLE, VISIBLE, T61 – обычная текстовая строка, не содержащая символов с кодом выше 127, BMP, UTF8 – текстовая строка с возможностью хранить локализованные данные, SET – задается имя секции, в которой описывается содержимое. Все это, конечно, весьма непонятно и используется по этой причине относительно редко. Например:

```
subjectAltName=otherName:1.2.3.4;UTF8:some blabla text
subjectAltName=otherName:2.3.4.5;BOOL:Yes,otherName:2.2.2.3;UTC:
120325191100Z
```

## 1.1.4. Создание ключа CA

Ну вот, с настройками покончено. Теперь необходимо сгенерировать Самый Главный Ключ, который будет использоваться для того, чтобы создать сертификаты по получаемым запросам и сертификат этого ключа, который и будет самым главным сертификатом и должен быть установлен всюду, где будут использоваться наши сертификаты. Имя файла сертификата и имя файла ключа мы уже задали в настройках openssl.cnf – caserv.crt и caserv.key соответственно. Срок действия сертификата CA выбирается достаточно продолжительным – 10 и более лет, потому что при истечении срока действия сертификата CA (а также при его отзыве, компрометации и т. д.) все сертификаты, подписанные им, автоматически становятся недействительными. Данный сертификат защищается паролем, причем пароль выбирается как можно более сложный – не менее 12 символов длиной, с обяза-

тельным использованием букв в разных регистрах, цифр и спецсимволов.

Команда на создание запроса сертификата CA и на генерацию сертификата подробно разбираться здесь не будет – генерация CSR подробно разбирается в разделе 1.2, а генерация сертификата – в разделе 1.3. Здесь будет приведена только непосредственно сама команда.

**# openssl req -config /etc/ssl/openssl.cnf -new -newkey rsa:2048 -keyout caserv.key -x509 -days 7300 -out caserv.crt**

```
Generating a 2048 bit RSA private key
....+++
.....................................................................+++
writing new private key to 'caserv.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [RU]:
State or Province Name (full name) [Novosibirsk region]:
Locality Name (eg, city) [Novosibirsk]:
Organization Name (eg, company) [DeltaHardware Ltd]:
Organizational Unit Name (eg, section) []:Certificate Authority
Common Name (eg, YOUR name) []:DeltaHardware Ltd Root CA
Email Address []:certmgr@deltahw.ru
```

Созданный файл caserv.key перемещаем в каталог ssl.key – это наш самый главный ключ! За ним надо следить в оба глаза и ни в коем случае не допускать его утечки. И разумеется, пароль от него должен знать в идеале один человек, тот, кто занимается непосредственно выдачей сертификатов (ну и конечно же конверты с Самыми Важными Паролями, лежащие в сейфе). Сертификат же – caserv.crt – наоборот, раздается всем в корпоративной сети – это наш самый главный сертификат, подпись которым другого сертификата означает «штамп УФМС» в масштабах корпоративной сети.

Еще нам не хватает списка отозванных сертификатов. Ну и что, что у нас их сейчас нет. Будут. Поэтому хоть и пустой, но список надо

создать. В разделе 1.4 подробно объясняется, как управлять списками отозванных сертификатов, а сейчас мы просто выполним команду

**# openssl ca -gencrl -out cacrl.crl**

```
Using configuration from openssl.cnf
Enter pass phrase for /usr/local/ca-dhw/ssl.key/caserv.key:
```

Еще будем считать, что у нас есть небольшой специализированный сайтик http://certs.deltahw.ru, где в каталоге certs будут выложены сертификат CA и CRL, поэтому ссылки на них вписываются в выдаваемые сертификаты. Также считаем, что нам выделили почтовый адрес certmgr@deltahw.ru, по которому мы принимаем запросы на создание сертификатов и прочие вопросы, относящиеся к работе корпоративного CA. Сюда же следует направлять сообщения в том случае, если устройство, на котором у вас сертификат (ноутбук, смартфон), было у вас украдено, для того чтобы вовремя отозвать сертификат. Своего рода «жалобная книга паспортиста».

Ну вот, все. Мы готовы выступать в роли «корпоративного паспортного стола». Господа, господа! Не мешайте работе паспортиста! В очередь, пожалуйста!

### 1.1.5. Скрипты по управлению CA

Для того чтобы быстро и достаточно эффективно управлять CA – а вы не поверите, но это станет весьма трудоемким занятием, когда будет выдано достаточное количество сертификатов, когда нужно будет постоянно их перевыдавать в связи с истечением срока действия или по другим причинам, – был разработан небольшой набор скриптов на Bourne Shell.

Разумеется, их можно и не использовать. На самом деле там 80% – это проверка введенных параметров и только одна, реже две-три строки – работа с сертификатами. Но это избавляет от утомительного запоминания порядка входных параметров.

Всего скриптов шесть:

- *keyca* – создает сертификат по полученному CSR;
- *keyreq* – создает CSR;
- *keycrl* – создает CRL и выкладывает его на удаленный сервер;
- *keyrevoke* – отзывает сертификат и пересоздает CRL;
- *keygenerate* – создает CSR, создает по нему сертификат и упаковывает все это в архив PKCS#12, который шифруется паролем и готов к отправке пользователю;

- *keypkcs* – создает архив PKCS#12, который шифруется паролем и готов к отправке пользователю.

Все скрипты используют две небольшие внешние библиотечки – colorprint.sh и commonlib.sh. Первая содержит две функции разноцветного вывода не терминал – `print_color()` и `print_color_nonl()` (как нетрудно догадаться, вторая не делает перевода строки и используется тогда, когда необходимо задать вопрос). Вторая содержит различные вспомогательные функции, но нас будет интересовать только одна – `checkstop_userid()`, которая проверяет, является ли пользователь, запустивший скрипт, пользователем с id 0, и если нет, прекращает работу скрипта с выдачей соответствующего сообщения. Тексты скриптов приводятся в сокращении, в частности из каждого удалена процедура `usage()`.

```
checkstop_userid()
{
  local _myid
  myid=`id -u`
  if [ $myid -ne 0 ]; then
    print_color $red "Insufficient rights: user with id $myid cannot do this"
    exit 115
  fi
}

black="[0;30m"
red="[0;31m"
green="[0;32m"
sand="[0;33m"
blue="[0;34m"
violet="[0;35m"
magenta="[0;36m"
gray="[0;37m"
ordinary="[0;38m"

print_color()
{
  printf "\033%s%s\033%s\n" $1 "$2" $ordinary
}

print_color_nonl()
{
  printf "\033%s%s\033%s" $1 "$2" $ordinary
}
```

Все скрипты работают одинаково – проверить права, проверить параметры, если все правильно – выполнить действие, иначе сказать

что-нибудь на тему того, что сначала надо бы почитать умных книжек. Или ничего не сказать, а просто завершиться.

### *Скрипт создания CSR*

Имеет два параметра командной строки – -k – длина ключа и -f – имя файла, куда будет сохранен CSR. Обычно используется с одним параметром -f – длину ключа редко приходится менять с умолчальной, например: **./keyreq -f somefile**. При этом создается ключ somefile.key и CSR somefile.pem. Скрипт патчит openssl.cnf, если нужно создать CSR с SAN, а после окончания работы восстанавливает первоначальный вариант конфигурационного файла.

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
. commonlib.sh

checkstop_userid

filename=""
keylength=2048
_san="empty"

if [ $# -lt 1 ]; then
  exit 20
fi

args=`getopt k:f: $*`;

set -- $args

for i in $args
 do
   case "$i"
    in
        -k) keylength="$2"; shift;
            shift;;

        -f) filename="$2"; shift;
            shift;;

        --) shift; break;;
   esac
 done

print_color $blue "Generate new Certificate Request. RSA $keylength-bit
without password"
echo ""

print_color_nonl $yellow "Would you like to use SAN? [y/n]: "
```

```
read _usesan

case "$_usesan"
 in
    [Yy]) print_color_nonl $green "Enter SAN text like DNS:dom1.org,
DNS:dom2.org: "
       read _san

       cd /etc/ssl/rootca
             patch -p0 < openssl.cnf.patch
       break;;

    *) break;;
esac

env SAN=$_san openssl req -new -sha1 -newkey rsa:$keylength -nodes
-keyout  $filename.key  -out  $filename.pem  -config  /etc/ssl/openssl.
cnf

if [ ${#_san} -ne 0 ]; then
   mv -f openssl.cnf.orig openssl.cnf
fi
```

### *Скрипт создания сертификата*

Пожалуй, наиболее часто используемый из всех. Имеет два параметра командной строки: -e – имя секции расширения, которая будет использоваться при создании сертификата (это имя фактически задает, – будет создан сертификат клиента или сертификат сервера, поскольку до этого момента все CSR одинаковые) и -f – имя файла CSR.

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
. commonlib.sh

checkstop_userid

filename=""
extension="usr_cert"

if [ $# -lt 1 ]; then
  exit 20
fi

args=`getopt f:e: $*`;

set -- $args

for i in $args
```