1.6 “Настройка безопасного удаленного доступа к серверу Ч.2”

Инфраструктура открытого ключа

Приведенный выше пример использования асимметричного шифрования для защиты соединения подходит для ограниченного доступа к серверу, когды мы сами генерируем пару ключей и уверены в их идентичности. Однако такой принцип совершенно не подходит для защиты соединения к публичным сервисам, например, для защиты HTTP (использование аналогичной схемы для HTTPS приведет к необходимости заранее сохранять ключ к КАЖДОМУ сайту, который пользователь планирует посетить). Кроме того необходимо при подключении однозначно удостовериться, что удаленная сторона является той, кем представляется (то есть дополнительно к ключу необходимо получать достоверную информацию о его владельце), что ключ не был отозван вследствие компрометации приватной части или других причин. Для решения данной проблемы используется инфраструктура открытого ключа (PKI), использование которой подразумевает создание специальных электронных документов - сертификатов, выдаваемых доверенными удостоверяющими центрами субъектам, подлинность которых необходимо удостоверить (обычно используется доменное имя, но можно использовать IP адрес или даже собственное имя для выпуска личного сертификата). Удостоверяющие центры, с помощью которых можно как выпустить сертификат, так и при необходимости проверить подлинность выпущенного ранее, являются доверенной третьей стороной.

Сертификат - это подписанный электронной подписью документ, содержащий публичный ключ, сведения о том, кому, когда и кем выдан данный сертификат, некоторую дополнительную информацию о владельце, а также срок действия сертификата, цели его использования, алгоритм подписи и тд. Поскольку от сертификата в первую очередь требуется обеспечение однозначной идентификации удаленной стороны при соединении/передаче данных, т.е. определении, что удаленная сторона действительно является тем, за кого себя выдает, необходимо, чтобы на клиентской стороне был или “правильный”

сертификат, с которым будет сравниваться предложенный сервером сертификат при установлении соединения, что опять же ведет к необходимости хранить всевозможные сертификаты в локальном хранилище, либо должен использоваться способ проверить подлинность сертификата, не имея его "эталона". Такой способ заключается в использовании локального хранилища сертификатов в ОС для хранения сертификатов доверенных удостоверяющих центров сертификации. Получив при соединении от удаленной стороны сертификат и определив на основе содержащейся в нем информации удостоверяющий центр (чаще всего корневые центры сертификации не выдают сертификаты, а удостоверяют другие центры, образуя цепочку от корневого центра до непосредственно центра, выпустившего данный сертификат), можно проверить по цепочке CA сертификат и соответственно доверять ему или нет, при условии того, что Вы доверяете или не доверяете CA в цепочке.

Назначение центра сертификации (CA) - выпуск различных сертификатов, подтверждающих принадлежность открытого ключа владельцу, указанному в сертификате, а также публикация списка отозванных сертификатов. Существуют корневые и промежуточные удостоверяющие центры сертификации. Корневые центры, как правило, не подписывают сертификаты клиентов (кроме, может быть, автономного CA для небольшой организации), они удостоверяют подлинность промежуточных CA, которые уже в свою очередь могут выпускать сертификаты или удостоверять другие CA от имени корневого, в результате чего в сертификатах содержится вся цепочка до корневого CA.

Минимальная рекомендуемая конфигурация для использования в реальной производственной среде - корневой CA и промежуточный CA, выдающий сертификаты по требованиям. Если ключ CA будет скомпрометирован, потребуется обновить ключ и, соответственно, корневой сертификат тоже. Это приведёт к тому, что все сертификаты, подписанные этим CA станут недействительны, для избежания подобной ситуации и используется многоуровневая структура, чтобы снизить риск компрометации приватных ключей корневых CA, которые строго рекомендуется развертывать на изолированных физических серверах с

отсутствием (постоянного) подключения к сети (вплоть до того, что требуется вывести из строя все сетевые адаптеры).

Создание локального CA с помощью OpenSSL

Для генерации ключей RSA, создания и подписания сертификатов может использоваться OpenSSL — пакет с криптографической библиотекой со свободным и открытым исходным кодом, которая предоставляет инструменты для работы с цифровыми сертификатами. В данной работе для упрощения мы будем выдавать сертификаты конечным потребителям от имени корневого CA и не будем касаться процедуры отзыва сертификатов.

Создайте каталог, в котором будете хранить сертификаты и ключи

```
mkdir ~/ca && cd ~/ca
```

**В рамках данной работы мы храним сертификаты и ключи в домашней директории пользователя, но стоит отметить, что для реального CA правильнее будет хранить сертификаты и ключи в специально существующих для этого директориях* `/etc/pki/tls/certs /etc/pki/tls/private`.

```
umask 077
```

Сгенерируйте защищенный приватный ключ будущего CA длиной 4096 бит с помощью следующей команды:

```
openssl genrsa -aes256 -out rootCA.key 4096
```

Затем выпустите самоподписанный корневой сертификат с указанными после ключа `subj` параметрами (в CN вместо *`voip.local`* можете подставить `hostname` CentOS) с помощью следующей команды:

```
openssl req -new -x509 -key rootCA.key -sha256 -days 10000 -out rootCA.crt -subj '/C=RU/ST=Moscow/L=Zelenograd/O=MIET/OU=TCS/CN=voip.local CA'
```

Добавьте созданный корневой сертификат в список доверенных центров сертификации, для этого скопируйте его в директорию для доверенных сертификатов CA и обновите список доверенных CA

```
sudo cp ca/rootCA.crt /etc/pki/ca-trust/source/anchors/rootCA.crt
```

```
sudo update-ca-trust
```

Сгенерируйте ключ для клиентского сертификата

```
openssl genrsa -out pma.key 2048
```

Сгенерируйте запрос на подпись сертификата с указанием параметров (в CN подставьте IP адрес CentOS)

```
openssl req -new -key pma.key -out pma.csr -subj '/C=RU/ST=Moscow/L=Zelenograd/O=MIET/OU=TCS/CN=IP_address'
```

Выпустите сертификат согласно запросу, подписав его ранее выпущенным сертификатом CA

```
openssl x509 -req -in pma.csr -days 365 -CA rootCA.crt -CAkey rootCA.key -CAcreateserial -out pma.crt
```

Посмотреть данные сертификата

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

Проверка, что сертификат действительно выпущен указанным CA

```
openssl verify -verbose -CAfile rootCA.crt pma.crt
```

**если бы сертификат был выдан промежуточным центром, то пришлось бы добавить данные о сертификатах всех промежуточных центров, чтобы сервер отсылал клиенту корректные данные о цепочке сертификатов*

Создайте каталог для сертификатов SSL, используемых NGINX:

```
sudo mkdir /etc/nginx/ssl/
```

Скопируйте сертификаты, предназначенные для phpmyadmin в созданную директорию

```
sudo cp ~/ca/pma.key /etc/nginx/ssl/ && sudo cp ~/ca/pma.crt /etc/nginx/ssl/
```

Сгенерируйте файл параметров алгоритма обмена ключами Диффи-Хеллмана для усиления стойкости используемых шифров и использования forward secrecy. Эта технология позволяет уменьшить возможный ущерб при компрометации ключа путем генерации отдельного ключа для каждой сессии (а не использовать один общий ключ).

```
openssl dhparam -out /etc/nginx/ssl/dhparam.pem 2048
```

В файле конфигурации для сайта phpMyAdmin секцию `server` приведите к следующему виду:

```
server {
    listen       443 ssl http2;
    server_name  _;
    ssl_protocols         TLSv1.2 TLSv1.3;
    ssl_certificate     /etc/nginx/ssl/pma.crt;
    ssl_certificate_key /etc/nginx/ssl/pma.key;
    ssl_session_timeout               10m;
    ssl_session_cache      shared:TLS:10m;
    ssl_session_tickets               off;
    ssl_ciphers          HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers          on;
    ssl_dhparam /etc/nginx/ssl/dhparam.pem;
    root /var/www/phpmyadmin;
    charset utf-8;
    index  index.php index.html index.htm;

    location / {
        try_files $uri /index.php?$args;

    }

    location ~ \.php$ {
        try_files  $uri =404;
        fastcgi_pass unix:/run/php-fpm/www.sock;
        fastcgi_index index.php;
        fastcgi_param HTTPS on;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param QUERY_STRING    $query_string;
        include fastcgi_params;
    }
```

В начало того же файла с конфигурацией для сайта phpMyAdmin добавьте еще одну секцию server для перенаправления всех запросов от клиентов с порта 80 (HTTP) на 443 порт (HTTPS).

```
server {
        listen 80;
        server_name _;

        location / {
            return 301 https://$host$request_uri;
        }
```

```
}
```

Проверьте корректность файла конфигурации `sudo nginx -t`

Перезапустите NGINX `sudo systemctl restart nginx`

Теперь покажем, как можно проверить работоспособность сервера, использующего SSL/TLS с помощью OpenSSL клиента, это также полезно использовать при устранении ошибок/неисправностей.

`openssl s_client -host localhost -port 443`

При работоспособном HTTPS сервере верхняя часть вывода должна выглядеть подобно приведенному ниже, указано состояние (CONNECTED), отображена цепочка сертификации и результаты проверки валидности сертификатов (возврат 1 означает успешное прохождение):

```
CONNECTED(00000003)
Can't use SSL_get_servername
depth=1 C = RU, ST = Moscow, L = Zelenograd, O = MIET, OU = TCS, CN = voip.local CA
verify return:1
depth=0 C = RU, ST = Moscow, L = Zelenograd, O = MIET, OU = TCS, CN = 192.168.127.137
verify return:1
---
Certificate chain
 0 s:C = RU, ST = Moscow, L = Zelenograd, O = MIET, OU = TCS, CN = 192.168.127.137
   i:C = RU, ST = Moscow, L = Zelenograd, O = MIET, OU = TCS, CN = voip.local CA
```

Для доступа к порту 443 потребуется добавить его в список разрешенных

`sudo systemctl firewall-cmd --add-service=https --permanent`

Зайдите на веб-интерфейс phpMyAdmin. Браузер должен предупредить об угрозе безопасности вследствие недоверенного издателя сертификата. На данном этапе игнорируйте это предупреждение.

**если веб-интерфейс недоступен, проверьте, что прослушивается 443 порт (присутствует в выводе команды `ss -tlpn`), проверьте права доступа на сертификаты (владелец - nginx*

Защита сайта

Для дополнительной защиты доступа к некоторым сайтам/разделам можно использовать несколько различных механизмов аутентификации и авторизации, рассмотрим один из простейших, базовую аутентификацию HTTP. При ее использовании сервер при обращении к защищенному ресурсу будет посылать

ответ `401 Unauthorized` и добавлять дополнительный заголовок HTTP `WWW-Authenticate` с указанием схемы (Basic в данном случае) и сопутствующих параметров аутентификации. Браузер, получив такой ответ, отображает окно ввода учетных данных и потом отсылает их в незашифрованном виде, вставляя заголовок `Authorization`. При использовании HTTPS такая схема достаточно безопасно, поскольку заголовок шифруется вместе с данными.

Для использования базовой аутентификации с NGINX, необходимо установить пакет htpasswd для создания хэшированных скрытых одноименных файлов, содержащих имена пользователей и пароли. Пакет входит в состав утилит для веб-сервера Apache и устанавливается следующей командой:

```
sudo yum install httpd-tools
```

Создайте пользователя developer (после ввода данной команды система запросит ввод пароля для этого нового пользователя, его задайте самостоятельно):

```
sudo htpasswd -c /etc/nginx/conf.d/.htpasswd developer
```

Теперь осталось только указать NGINX, для каких сайтов/локаций требуется использовать созданный файл. Вставьте следующие строки в локацию, соответствующую phpMyAdmin внутри секции server для 443 порта:

```
auth_basic              "Restricted Access!";
auth_basic_user_file    /etc/nginx/conf.d/.htpasswd;
```

Защита соединений SIP RTP для Asterisk

Протоколы SIP и RTP передают все данные в открытом виде, как Вы могли сами убедиться, анализируя их сообщения в Wireshark, для защиты передаваемых данных были созданы протокол Secure SIP (SIPS) для защиты сигнального трафика и протоколы Secure RTP (SRTP) и Z RTP (ZRTP) для защиты медиа трафика. Аналогично протоколу HTTPS, протоколам SIPS SRTP для безопасной передачи с шифрованием данных сигнального и медиа трафика также требуется сертификат.

Сгенерируйте ключ и сертификат для сервера Asterisk, в CN вместо *IP_addr* подставьте IP адрес CentOS

```
cd ~/ca
```

```
openssl genrsa -out asterisk.key 2048
openssl req -new -key asterisk.key -out asterisk.csr -subj
'/C=RU/ST=Moscow/L=Zelenograd/O=MIET/OU=TCS/CN=IP_addr'
openssl x509 -req -in asterisk.csr -days 365 -CA rootCA.crt -CAkey
rootCA.key -CAcreateserial -out asterisk.crt
```

Сгенерируйте ключ и сертификат для SIP-клиента с номером XXXX (подставьте существующий в Вашей конфигурации номер), он будет подписан не корневым сертификатом, а сертификатом, выданным для Asterisk.

```
openssl genrsa -out clientXXXX.key 2048
openssl req -new -key clientXXXX.key -out clientXXXX.csr -subj
'/C=RU/ST=Moscow/L=Zelenograd/O=MIET/OU=TCS/CN=XXXX'
openssl x509 -req -in clientXXXX.csr -days 365 -CA asterisk.crt -
CAkey asterisk.key -CAcreateserial -out clientXXXX.crt
```

Задание Сгенерируйте таким образом сертификат для каждого клиента

Создайте директорию для хранения ключей и сертификатов Asterisk

```
sudo mkdir /etc/asterisk/keys && umask 277 /etc/asterisk/keys
```

Скопируйте в эту директорию ключ и сертификат для Asterisk и клиента

Установите владельца директории и файлов на asterisk

```
sudo chown -R asterisk.asterisk /etc/asterisk/keys
```

Для возможности клиентам использовать SIPS и SRTP, необходимо внести изменения в конфигурацию PJSIP. Для каждого клиента, описанного в таблице ps_endpoints БД asteriskdb значение параметра transport измените на `tls-transport`, добавьте параметры со следующими значениями

Параметр	Значение
dtls_cert_file	/etc/asterisk/keys/clientXXXX.crt
dtls_private_key	etc/asterisk/keys/clientXXXX.key
media encryption	sdes

добавьте новую секцию `tls-transport` в начало файла `/etc/asterisk.pjsip.conf`

```
[tls-transport]
type=transport
protocol=tls
bind=0.0.0.0:5061
direct_media=no;
cert_file=/etc/asterisk/keys/asterisk.crt
priv_key_file=/etc/asterisk/keys/asterisk.key
cipher=ADH-AES256-SHA,ADH-AES128-SHA
method=tlsv1
allow_reload=true
```

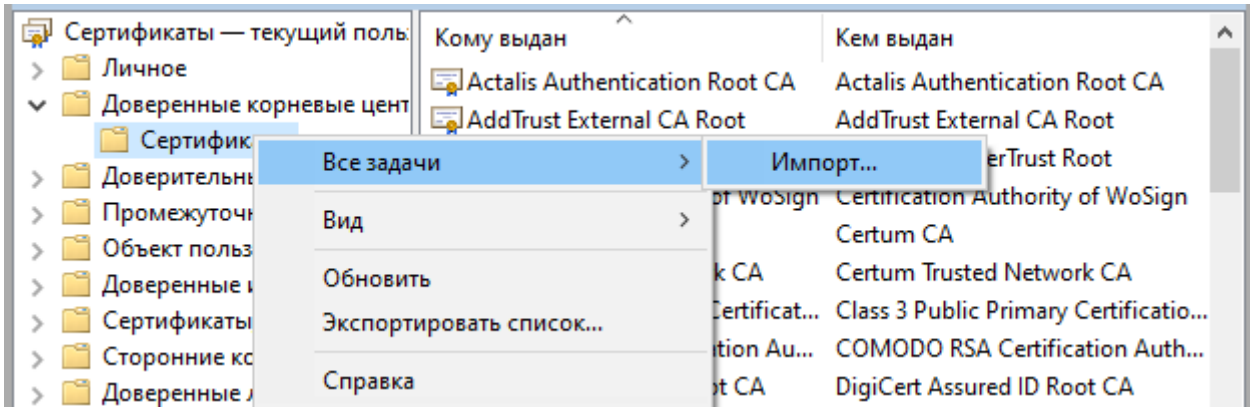
Перезапустите службу Asterisk для применения настроек нового транспорта.

Для проверки работоспособности осталось сконфигурировать клиент. Для установления защищенного соединения клиенту потребуется сертификат. С помощью WinSCP перенесите на хостовой компьютер сертификат, сгенерированный для клиента. Запустите клиент PhonerLite, во вкладке **Конфигурация**, разделе **Сеть** смените порт на указанный в конфигурации PJSIP для TLS, протокол смените на TLS, в разделе **Кодеки** отметьте пункт SRTP, в разделе **Сертификаты** в поле Сертификат клиента укажите путь до клиентского сертификата, сохраните параметры. Убедитесь, что клиент зарегистрировался через TLS-соединение (внизу окна рядом с индикатором регистрации появится символ ключа с надписью TLS), совершите вызов, убедитесь, что голосовой трафик передается.

В разделе **Сертификаты** отметьте два доступных пункта для проверки сертификата сервера и использовании локального хранилища сертификатов в Windows, попробуйте снова авторизовать клиент на Asterisk. Сертификат для Asterisk был выдан центром сертификации, с самоподписанным сертификатом, поэтому он считается недоверенным и проверка не проходит. Чтобы клиент мог доверять такому сертификату, нужно или указать путь до сертификата и ключа сервера или поместить корневой сертификат в хранилище.

Для того, чтобы добавить корневой сертификат в список доверенных центров сертификации, перенесите rootCA.crt на хостовой компьютер, откройте оснастку MMC Сертификаты (certmgr.msc). Аналогично приведенному ниже

выполните импорт сертификата, укажите расположение файла rootCA.crt, не изменяйте остальные параметры хранилища. После успешного импорта Вы сможете найти сертификат в общем списке справа.



Снова попробуйте авторизовать PhonerLite на сервере Asterisk, теперь проверка должна пройти удачно.

Настройка безопасности PHP

В файле /etc/php/php.ini найдите/добавьте следующие параметры:

```
# Отключаем опасные функции
disable_functions = phpinfo, system, mail, exec
# Максимальное время исполнения скрипта
max_execution_time = 30
# Максимальное время, которое может потратить скрипт на обработку данных запроса
max_input_time = 60
# Максимальное количество памяти, выделяемое каждому скрипту
memory_limit = 8M
# Максимальный размер данных, отсылаемых скрипту с помощью метода POST
post_max_size = 8M
# Максимальный размер загружаемых файлов
upload_max_filesize = 2M
# Не показывать ошибки PHP-скриптов пользователям
display_errors = Off
# Включаем Safe Mode
safe_mode = On
# Включаем SQL Safe Mode
sql.safe_mode = On
# Защищаемся от утечки информации о PHP
expose_php = Off
```

```
# Ведем логи
log_errors = On
# Запрещаем открытие удаленных файлов
allow_url_fopen = Off
```

Данные параметры повышают защищенность веб-сервера, ограничивая возможности по исполнению вредоносного кода PHP.