Automated certificate management (method 1)

Background

At Sinodun we use dehydrated https://github.com/lukas2511/dehydrated to manage our certificates. Also we use the dns-01 challenge to renew them.

Since we run multiple DNS-over-TLS servers, the method used here employs a single 'certificate management' server to renew the certificates, update the zone with the dns-01 challenge and make the renewed certificates available via ftp. A script is then run on each DNS server to download any new certificates using ftp to each server and restart the DNS service. In this example we use TLS proxies in front of BIND, or Knot resolver.

DNS setup

You could put the dns-01 challenge responses in your organisations main zone. However it is more flexible to dedicate a separate zone for that, especially if you are responsible for running the servers but someone else or some other organisation is responsible for the DNS.

For authoritative DNS we use knot https://www.knot-dns.cz/ because it has a nice interface for managing zones. The same effect could be obtained using dynamic updates.

It is a good idea to sign your zone if you are going to be putting challenge response records in it. However, the nameserver must be able to sign updates as they are applied.

CNAMES

Lets assume that we are managing certificates for two servers dnsovertls1.<YOURDOMAIN>. and dnsovertls.<YOURDOMAIN>.

Add CNAME RRs to the <YOURDOMAIN> zone like these. These will redirect queries for the dns-01 challenge to a dedicated zone which can exist on and be served by a dedicated certificate management server

Add CNAMES

```
_acme-challenge.dnsovertls1.<YOURDOMAIN>. 300 CNAME dnsovertls1.<YOURDOMAIN>.acme.<YOURDOMAIN>.
_acme-challenge.dnsovertls.<YOURDOMAIN>. 300 CNAME dnsovertls.<YOURDOMAIN>.acme.<YOURDOMAIN>.
```

Delegate acme.<YOURDOMAIN> to the certificate management server by adding NS and DS RRs to the <YOURDOMAIN> zone. Lets assume the server is called ns1.acme.<YOURDOMAIN>

Create an empty zone for acme.<YOURDOMAIN> (SOA and NS and A/AAAA RRs). Configure knot to sign the domain

```
server:
  listen: <YOURIP4>@53
  listen: <YOURIP6>@53
  user: knot:knot

log:
  - target: syslog
    any: info

policy:
  - id: dnssec
    algorithm: ECDSAP256SHA256

zone:
  - domain: acme.<YOURDOMAIN>
    file: "acme.<YOURDOMAIN>.zone"
    dnssec-signing: on
    dnssec-policy: dnssec
```

```
acme.<YOURDOMAIN>.       3600    SOA     ns1.acme.<YOURDOMAIN>. user.<YOURDOMAIN>. 2016120927 30000 300 604800
300
acme.<YOURDOMAIN>.       3600    NS      ns1.acme.<YOURDOMAIN>.
ns1.acme.<YOURDOMAIN>.   3600    A       <YOURIP4>
ns1.acme.<YOURDOMAIN>.   3600    AAAA    <YOURIP6>
```

Configure dehydrated

Get dehydrated https://github.com/lukas2511/dehydrated and create some config.

Uncomment the first two lines until you are sure everything is working. This will allow you to run the scripts for debugging without exceeding the production Let's Encrypt limits

dehydrated config

```
# CA="https://acme-staging.api.letsencrypt.org/directory"
# CA_TERMS="https://acme-staging.api.letsencrypt.org/terms"
LICENSE="https://letsencrypt.org/documents/LE-SA-v1.1.1-August-1-2016.pdf"
CERTDIR=/<PATH>/dehydrated/certs
DOMAINS_TXT=/<PATH>/dehydrated/dehydrated.domains
CHALLENGETYPE="dns-01"
HOOK=/<PATH>/dehydrated/hook.sh
PRIVATE_KEY_RENEW="no"
PRIVATE_KEY_ROLLOVER="no"
CONTACT_EMAIL=you AT email
```

The two PRIVATE_KEY lines ensure that the key is not replaced and so SPKI pinning will work.

Create a domains file

Create a dehydrated.domains file (*/<PATH>/dehydrated/dehydrated.domains*)

dehydrated domains

```
dnsovertls.<YOURDOMAIN>
dnsovertls1.<YOURDOMAIN>
```

Write a hook script

Create a hook.sh script (*/<PATH>/dehydrated/hook.sh*).

dehydrated hook

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

set -e
set -u
set -o pipefail

check_knotc_exit_code() {
  if [[ ! $1 -eq 0 ]] ; then
    /usr/sbin/knotc zone-abort acme.<YOURDOMAIN>.
    exit 1
  fi
}

deploy_challenge() {
  local DOMAIN="${1}.acme.<YOURDOMAIN>." RDATA="${3}"
```

```
  if [[ "${1}" == "ns1.acme.<YOURDOMAIN>" ]] ; then
    DOMAIN="_acme-challenge.${1}."
  fi
  echo "Adding $DOMAIN 10 TXT \"$RDATA\""
  echo "zone-begin acme.<YOURDOMAIN>."
  /usr/sbin/knotc zone-begin acme.<YOURDOMAIN>.
  check_knotc_exit_code $?
  echo "zone-set acme.<YOURDOMAIN>. $DOMAIN 10 TXT \"$RDATA\""
  /usr/sbin/knotc zone-set acme.<YOURDOMAIN>. $DOMAIN 10 TXT \"$RDATA\"
  check_knotc_exit_code $?
  echo "zone-commit acme.<YOURDOMAIN>."
  /usr/sbin/knotc zone-commit acme.<YOURDOMAIN>.
  check_knotc_exit_code $?
}

clean_challenge() {
   local DOMAIN="${1}.acme.<YOURDOMAIN>." RDATA="${3}"
   if [[ "${1}" == "ns1.acme.<YOURDOMAIN>" ]] ; then
     DOMAIN="_acme-challenge.${1}."
   fi
   echo "Removing $DOMAIN 10 TXT \"$RDATA\""
   echo "zone-begin acme.<YOURDOMAIN>."
   /usr/sbin/knotc zone-begin acme.<YOURDOMAIN>.
   check_knotc_exit_code $?
   echo "zone-unset acme.<YOURDOMAIN>. $DOMAIN 10 TXT \"$RDATA\""
   /usr/sbin/knotc zone-unset acme.<YOURDOMAIN>. $DOMAIN TXT
   check_knotc_exit_code $?
   echo "zone-commit acme.<YOURDOMAIN>."
   /usr/sbin/knotc zone-commit acme.<YOURDOMAIN>.
   check_knotc_exit_code $?
}

deploy_cert() {
   local DOMAIN="${1}" KEYFILE="${2}" CERTFILE="${3}" FULLCHAINFILE="${4}" CHAINFILE="${5}" TIMESTAMP="${6}"
   # Copy cert files (not the keys) to your download site of choice - web, ftp etc...
    echo "Deploying certs to ftp server"
    echo "Deploy for domain ${DOMAIN}: ${CERTFILE} ${FULLCHAINFILE} ${CHAINFILE}"
    cp ${CERTFILE} /srv/ftp/certs/${DOMAIN}/
    cp ${FULLCHAINFILE} /srv/ftp/certs/${DOMAIN}/
    cp ${CHAINFILE} /srv/ftp/certs/${DOMAIN}/
    chmod 644 /srv/ftp/certs/${DOMAIN}/*
}

unchanged_cert() {
   local DOMAIN="${1}" KEYFILE="${2}" CERTFILE="${3}" FULLCHAINFILE="${4}" CHAINFILE="${5}"
   # nothing yet..
}

startup_hook() {
  # This hook is called before the cron command to do some initial tasks
  # (e.g. starting a webserver).
   for i in $(awk ' { print $1 } ' /<PATH>/dehydrated/dehydrated.domains ) ; do
      CERTPATH=/srv/ftp/certs/$i
      if [[ ! -d ${CERTPATH} ]] || \
         [[ $(stat -c %a ${CERTPATH}) -ne 770 ]] || \
         [[ $(stat -c %U ${CERTPATH}) != "<USER THE CERTS ARE CREATED BY>" ]] || \
         [[ $(stat -c %G ${CERTPATH}) != "ftp" ]] ; then
           echo "Creating, chmod, chown FTP directory for $i"
           mkdir -p ${CERTPATH}
           chown <USER THE CERTS ARE CREATED BY>:ftp ${CERTPATH}
           chmod 770 ${CERTPATH}
      fi
   done
   for i in $(cat /<PATH>/dehydrated/dehydrated.domains) ; do
      # Now check that the zone owner has a CNAME pointing to us for this domain
      CNAME=$(dig _acme-challenge.$i CNAME +short)
      if [[ -z ${CNAME} ]] ; then
        echo "There is no CNAME pointing here for $i in the DNS"
        exit 1
      fi
  done
```

```
}

exit_hook() {
  # This hook is called at the end of the cron command and can be used to
  # do some final (cleanup or other) tasks.

  :
}

invalid_challenge() {
    local DOMAIN="${1}" RESPONSE="${2}"

    # This hook is called if the challenge response has failed, so domain
    # owners can be aware and act accordingly.
    #
    # Parameters:
    # - DOMAIN
    #   The primary domain name, i.e. the certificate common
    #   name (CN).
    # - RESPONSE
    #   The response that the verification server returned
}

request_failure() {
    local STATUSCODE="${1}" REASON="${2}" REQTYPE="${3}"

    # This hook is called when an HTTP request fails (e.g., when the ACME
    # server is busy, returns an error, etc). It will be called upon any
    # response code that does not start with '2'. Useful to alert admins
    # about problems with requests.
    #
    # Parameters:
    # - STATUSCODE
    #   The HTML status code that originated the error.
    # - REASON
    #   The specified reason for the error.
    # - REQTYPE
    #   The kind of request that was made (GET, POST...)
}

HANDLER="$1"; shift
if [[ "${HANDLER}" =~ ^
(deploy_challenge|clean_challenge|deploy_cert|unchanged_cert|invalid_challenge|request_failure|startup_hook|exit
_hook)$ ]]; then
  "$HANDLER" "$@"
fi
```

Write a script to download the certificates to your DNS servers

Do something like the following on each servers to download the certificates from your main certificate server.

Download certificates

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

set -e
set -u
set -o pipefail

if [[ ${LOGNAME} != "root" ]] ; then
  echo "Must be root. Exiting..."
  exit 1
fi
```

```
# depending on what you are using to terminate the TLS connections
# you might need to ensure the correct user can access the certificate files
SERVICEAPP=$1
if [[ ${SERVICEAPP} == "haproxy" ]] || \
   [[ ${SERVICEAPP} == "knot-resolver" ]] ; then
  GROUPUSER=${SERVICEAPP}
elif [[ ${SERVICEAPP} == "apache2" ]] || \
     [[ ${SERVICEAPP} == "apache" ]] || \
     [[ ${SERVICEAPP} == "nginx" ]] ; then
  GROUPUSER="www-data"
else
  echo "Unsupported service application. Exiting..."
  exit 1
fi

CERTPATH="/etc/certs"

if [[ ! -d ${CERTPATH} ]] || \
   [[ $(stat -c %a ${CERTPATH}) -ne 750 ]] || \
   [[ $(stat -c %U ${CERTPATH}) != "root" ]] || \
   [[ $(stat -c %G ${CERTPATH}) != ${GROUPUSER} ]] ; then
  echo "No ${CERTPATH} directory or permissions not good."
  echo "Do something like:"
  echo "  sudo mkdir -p ${CERTPATH}"
  echo "  sudo chmod 750 ${CERTPATH}"
  echo "  sudo chown root:${GROUPUSER} ${CERTPATH}"
  echo "Exiting..."
  exit 1
fi

cd ${CERTPATH}

if [[ ! -f privkey.pem ]] || \
   [[ $(stat -c %a privkey.pem) -ne 640 ]] || \
   [[ $(stat -c %U privkey.pem) != "root" ]] || \
   [[ $(stat -c %G ${CERTPATH}/privkey.pem) != ${GROUPUSER} ]] || \
   [[ ! -s privkey.pem ]] ; then
  echo "Private key not found or permissions not good. Exiting..."
  exit 1
fi

# Use the hostname to figure out the cert CN
CN=$(hostname -f)
if [[ "${CN}" == "test1.dnsovertls.nl" ]] ; then
  CN="dnsovertls.<YOURDOMAIN>"
elif [[ "${CN}" == "test2.dnsovertls.nl" ]] ; then
  CN="dnsovertls1.<YOURDOMAIN>"
fi

tar -cf backup-$(date +%s).tar *.pem


curl -s --ssl-reqd --tlsv1 -O ftp://ns1.acme.<YOURDOMAIN>:21/certs/${CN}/cert.pem
curl -s --ssl-reqd --tlsv1 -O ftp://ns1.acme.<YOURDOMAIN>:21/certs/${CN}/chain.pem
curl -s --ssl-reqd --tlsv1 -O ftp://ns1.acme.<YOURDOMAIN>:21/certs/${CN}/fullchain.pem
if [[ ! -s cert.pem ]] || \
   [[ ! -s chain.pem ]] || \
   [[ ! -s fullchain.pem ]] ; then
  echo "At least one of the certificate files is empty or missing. Exiting..."
  exit 1
fi

cat fullchain.pem > keycert.pem
cat privkey.pem >> keycert.pem
chmod 640 cert.pem chain.pem fullchain.pem keycert.pem
chown root:${GROUPUSER} cert.pem chain.pem fullchain.pem keycert.pem

if [[ ${SERVICEAPP} == "haproxy" ]] || \
   [[ ${SERVICEAPP} == "apache2" ]] || \
   [[ ${SERVICEAPP} == "nginx" ]] ; then
  INITNAME=${SERVICEAPP}
```

```
elif [[ ${SERVICEAPP} == "apache" ]] ; then
  INITNAME="apache2"
else
  echo "Unsupported service application. Exiting..."
  exit 1
fi
systemctl reload-or-restart ${INITNAME}
```