Wildcard Letsencrypt Certificates, Google Domains, DNSSEC, and BIND

Bill Walker bw@w5gfe.org

December 13, 2018

Contents

1 Introduction

I run a personal server in my home. That machine offers services to the external world through email and web, and also provides services to my own internal network, which includes security cameras, several terminals, an internal wireless arrangement, a couple of dedicated servers, and an internal web with pages whose purpose differs from those pages intended for external consumption.

By necessity, my installation demands "split horizon" domain name service (DNS), a need which is easily accomodated by running "BIND" on a server which faces both the external world and the internal network. This presents one "view" of my network to the external world, and another entirely different "view" (ie different IP's for the same domain) to the internal network.

In keeping with my personal desire to engage in "best practices" (whatever those are!) I wish to employ DNSSEC. I also have no intention of engaging any service for which I have to pay.

Here is a list of what I need:

- DNSSEC
- split horizon DNS
- LetsEncrypt wildcard certificates

Acquiring all of these things at the same time required a surprisingly complicated effort. This article is intended to let you know how it was managed.

2 DNSSEC

DNSSEC offers cryptographically verified responses to DNS queries. It is not terribly widely adopted, possibly because of the necessity for periodic "resigning" of data by cryptographic means. These signatures must be conveyed to all the authoritative DNS servers which might respond to a DNS inquiry. Best practices suggest that any site should have more than one DNS server situated in electronically diverse locations.

My domain was purchased through Google Domains, and as part of that purchase, I received a free DNS service associated with my account. Google offers another domain name service, which is not free, but which also has an API which could be used to advantage. The fee, while not exorbitant, is more than I am willing to pay. If you use the alternative DNS service, you can skip the rest of this article.

Google's free service does offer DNSSEC which is easily configured, and Google takes care of such things as periodic "resigning", as well as electronic diversity of location. I decided that service was just too useful to pass up, and I configured an external view of my network (ie external IP addresses) on the easy to use control panel.

There were a couple of disadvantages for the free Google Domains DNS. There is only a very minimal API, which is useless for my purposes. The other disadvantage is that it provides only an "external view" of my network, and therefore is not useful for my internal DNS requirements.

3 split horizon DNS

I really need two separate "views" of the DNS domain. One view, the "external" view, presents a set of IP addresses to a client which is external to my own

internal network (ie the Internet). The other view, the "internal" view, consists of entirely different IP addresses which are presented to a query from within my home network.

If I am using my laptop from my living room, and seek to access my local website the DNS will present a certain IP to my web browser. If I carry the same laptop to a remote location, a DNS request for the address of the same website will respond with an entirely different IP address.

A DNS server which can respond to the same enquiry with different answers depending on the origin of the query is said to have a "split horizon".

The common (and free) Berkeley Internet Name Domain (BIND) DNS server is easy to compile and configure on almost any Linux system. It offers, among many other features, "split horizon" views, as well as multiple means of querying for domain names presented by those views.

It is also possible to configure BIND "on the fly" with an API.

I installed BIND on my local machine and configured it with both an internal and an external "view", with appropriately selected IP addresses in each view.

This solved my need for internal views of my network, and at least did not conflict with the external views which were also being presented by Google's service. It solved most of my problems, but did not meet all of my needs because of the requirements discussed in the next section.

4 LetsEncrypt wildcard certificates

The use of "certificates" has been long accepted as best practice for web servers. In the past, deployment of cryptographically secure web traffic has been a fairly expensive undertaking because of the need to purchase such a certificate (which is really a set of encryption keys) from a "certificate authority".

In about January of 2016, a new certificate authority named "LetsEncrypt" came on the scene. The certificates which they issued were both "free" and "trusted" by most browsers. The LetsEncrypt effort has greatly encouraged the adoption of "secure" web protocols. At this writing (October of 2018) there are more than 70 million valid LetsEncrypt certificates.

The LetsEncrypt certificates have a rather short lifetime and must be periodically reissued or "renewed". That process is not especially cumbersome, and is automated easily.

Although the original certificates were issued only for fully qualified domain names, one could to some extent handle a few subdomains by including multiple names in a single certificate.

Recently, LetsEncrypt began issuing "wildcard" certificates, which are valid for a given domain, and any subdomains. These certificates have 90 day lifetimes, and must be renewed periodically.

The current renewal protocol involves issuing a "challenge" request to domain name service, asking the authoritative DNS server to present a specific TXT record as part of a "handshake" to authenticate the exchange and allow

the certificate authority to establish the identity of the server making the request for renewal.

There is a shell script, termed "acme.sh", which can interact with the LetsEncrypt issuing authority, accept the challenge, and issue, via an API, orders to the DNS server to present the specific TXT record to meet the request, thus "validating" the request and allowing the renewal to proceed without human intervention (ie via cron, or other scheduled means). In other words, it can automate the renewal process.

5 My Problem

My problem is that I am using two separate DNS mechanisms, one of which I can control programatically (my own internal DNS server using BIND) and the other which I cannot control programatically (the Google DNS servers).

6 The Solution in a Nutshell

- First we need a valid wildcard certificate from LetsEncrypt. The shell script "acme.sh" can help with this task. The certificate will be good for 90 days.
- When the time comes to renew the certificate we can use the shell script "acme.sh" to issue the renewal request.

  LetsEncrypt will issue a response to the renewal request, but that response will be directed at the Google Domain DNS server, which cannot handle the challenge.
- Therefore I must arrange that the "challenge" which LetsEncrypt sends to the Google servers be then referred to my own BIND server's external view for resolution.
- The challenge will then be answered by the "acme.sh" script (on my machine) which is able to programatically adjust my local BIND server to answer the challenge correctly (by adjusting only the external view) and thus allow the renewal to proceed.

7 How it ties together

The order of things is important.

7.1 Configuring Google Domains

Suppose your domain is "example.com". In the Google Domains DNS panel select the Google Domain Name Servers, and enter any records that you need to,

including an A record for your own server, which we will assume is "ns.example.com". You will need two "extra" records as well. The listing in Listing 1 may help.

Listing 1: Minimal Google Domain DNS Entries

```
ns.example.com    A      1hr XXX.XXX.XXX.XXX
_acme-challenge   CNAME  1hr _acme-challenge.acmesh.example.com
acmesh            NS     1hr ns.example.com
```

Go ahead and enable DNSSEC. Now you are done with Google Domains, (but don't lose the access credentials)!

7.2 Configuring the BIND server

The local BIND server on your own machine needs several things. First run the program tsig-keygen, and capture its output. The "keyname" can be of your own chosing, but even the word "keyname" is OK. It will have to be used consistently below.

```
tsig-keygen keyname > keyfile.key
```

We are going to put the contents of the keyfile.key into the bind configuration shortly, and also use it for another purpose, so don't discard it.

The next step is going to take a while unless you are familiar with BIND. You must configure BIND with two separate "views".

First, near the top of the "named.conf" file, right after the "options" section, place the contents of the "keyfile.key" you generated above. See Listing 2.

Listing 2: Partial Global BIND Configuration

```
options {
  ...
} ;

key "keyname" {
  algorithm hmac-sha256;
  secret "pV+PIedIOh9fdF+ybHIhB6tKvvv+HKz5wrxxzk6kPc=";
} ;
```

It's a little tricky — the semicolons are all needed !

Add the internal view first, then configure the external view. You will need both of the "match-clients" sections, and you will need the acmesh.example.com zone in the external view. The "allow-update" clause is what enables us to programatically update the parameters of that zone (which is the one which receives the challenge). See Listing 3.

Listing 3: Internal and External view BIND Configuration

```
view "internal" {
        match-clients {
           !key "keyname";
           localnets;
           ::1/128;
         };

        recursion yes;

        zone "example.com" {
                type hint;
                file "internal.db";
        };

 };



 view "external" {
        match-clients {
           key "keyname";
           !localnets;
           any;
        };
        recursion no;


       zone "example.com" {
                type master;
                notify no;
                file "external.db";
              };

       zone "acmesh.example.com" {
                type master;
                file "acmesh.example.com.db";
                allow-update { key "keyname"; };
              };
};
```

The file “acmesh.example.com.db” contains (Listing 4.):

Listing 4: local zone file acmesh.example.com.db

```
$ORIGIN .
$TTL 86400      ; 1 day
acmesh.example.com        IN SOA  ns.example.com. adminid.example.com.
    (
                                2018101408 ; serial  (do it however you
                                     want to)
                                14400      ; refresh (4 hours)
                                7200       ; retry (2 hours)
                                1209600    ; expire (2 weeks)
                                43200      ; minimum (12 hours)
                                )
                          NS    ns.example.com.
```

Now we should start BIND and test things:
Use the command in Listing 5.

Listing 5: A Command for Testing

```
dig @localhost -k keyfile.key _acme-challenge.acmesh.example.com. any
```

to test the external view of your server. The key in the keyfile.key is what lets dig “see” the external view, rather than the internal view.

7.3 Issuing the first certificate

When you are satisfied that your BIND server, and the Google Domain servers are all working properly, you are ready to request your first certificate. You will need your external IP address XXX.XXX.XXX.XXX, and the file “keyfile.key” which you created above.

Install acme.sh by downloading it first from the website, and then issuing “acme.sh –install”

You can attempt to issue a certificate by using the command in Listing 6..

Listing 6: Issuing the First Certificate

```
NSUPDATE_SERVER=XXX.XXX.XXX.XXX  NSUPDATE_KEY=`pwd`/keyfile.key  ./acme
    .sh --log x --test \
     --issue -d example.com -d '*.example.com' --challenge-alias acmesh
         .example.com \
     --dns dns_nsupdate
```

This will create a few directories, and if all goes well place certificates in them. It will also create a log file named “x” that you can examine to see if anything went wrong.

Perhaps everything went well. If so, be aware that these certificates are not valid (they are “testing” certificates). If everything is OK, then we need to remove the “testing” status. It may be simpler just to uninstall the acme.sh

script and its associated directories and then reinstall it. Then reissue the last command without the "–test" argument. You should get valid certificates, which you can place whereever your web server requires them to be.

7.4 Requesting Renewal

Look at your crontab file! It's probably full of junk, unless you asked "acme.sh" not to create the cron entries. The last crontab entry should be the one you want. "acme.sh" installed a crontab for the account under which you were doing this work. The certificates probably did not wind up in the right place, but adding options to the crontab line will allow you to place the certificates whereever you want them, so long as you have permissions to do so. Alternatively, if you look in the directory in which acme.sh was installed, there is a directory named for your domain. Within that directory is a configuration file which can be edited to suit your needs.

The point is that after you have accomplished the tasks outlined in this essay, you probably will still have some things which you will need to clean up.

8 Tools

I found a couple of tools to be very helpful. One tool, "digger", is used to see what answers your local BIND DNS server provides to external queries. It's a one-liner, (Listing 7) but the tool can save you some typing. Of course you could just create an alias instead.

Listing 7: The "digger" tool

```
#!/bin/bash

dig @localhost -k keyfile.key $@
```

Use it almost as you would "dig". For instance, the command in Listing 8 will provide an (almost complete) external view of your DNS zone.

Listing 8: An example of using digger

```
digger example.com any
```

A second tool is useful for examining the certificates which are issued is shown in Listing 9.

Listing 9: A script to examine a certificate

```
#!/bin/bash
# usage: certlook nameofcert
openssl x509 -text -noout -certopt no_subject,no_header,no_version,
    no_serial,no_signame,no_subject,no_issuer,no_pubkey,no_sigdump,
    no_aux -in $1
```

9 Where to find things

9.1 Various URLS

- I suggest that you visit the certbot support forums at LetsEncrypt

  `https://community.letsencrypt.org`

  There is a wealth of experience and knowledge available, and an active community of support.

- The "acme.sh" script, central to this article, is available from Github

  `https://acme.sh/`

- You may find this site useful to be sure your DNSSEC is working:

  `http://dnsviz.net/`

- To verify the operation of your SSL installation, use:

  `https://www.ssllabs.com/ssltest/`
  and
  `https://www.htbridge.com/ssl/`

9.2 strugglers.net article

A really helpful article is at

```
https://strugglers.net/~andy/blog/2018/03/19/\
lets-encrypt-wildcard-certificates-acme-sh-and-automated-dns-verification/
```

Ironically, the article can be accessed through either http or https. This is the article which let me to the split-horizon solution presented in this paper, and I gratefully acknowledge the information in this paper as being mostly plundered from Andy's article.

When I started this project I began with a web search and landed on the strugglers.net article. I read it with much interest, and then set about getting things working.

Later, when writing this article I looked back at the strugglers.net site again, and was much chagrined to discover that this article is an unintentional paraphrase of Andy's web article. I almost trashed this article altogether, then decided to keep it because it chronicles my project, and its solution in my environment. In that spirit, perhaps someone else will find elucidation in a turn of phrase.

Therefore, thanks Andy, for your useful, complete, and inspiring article. My current effort is merely detail.