Build Dynamic DNS Server

Normal DNS services require human intervention to change an IP address, and this is usually pretty secure. However, there are situations when you need the DNS to change, most notably when a public DHCP address changes on normal, residential internet connections.

There are many commercial services that will do this for you, but they generally are limited by the number of domain names you can use. As an alternative, if you run your own DNS server, you can it up to provide your own Dynamic DNS service.

There are tons of ways to do this, but basically, the remote client must contact the DNS server, then the DNS server changes the IP address based on some message sent. This can be very insecure, as a system which does no authentication could have their Dynamic DNS addresses changed/added with no limits. It can be done with any protocol. https is very common, but this example will use ssh with public key authentication for extra security.

The system described here is *fairly secure*, meaning there are ways to get around it, but they are difficult. There are a few tweaks that can be done to make it more secure, which I'll describe later on.

Our basic system will be:

1. Set up Bind9 on a server
    1. Designate one domain to hold all of your dynamic addresses.
        1. It is **not** a good idea to have a domain that runs both dynamic and static
        2. If you must use a single domain, consider setting up a subdomain to hold the dynamic IP's
    2. Each client will have a unique FQDN in the domain in question
    3. Set Bind to allow dynamic updates using a shared key
2. Clients will indicate an IP address change by making an ssh connection to the DNS server
    1. DNS server will allow communication from clients who use public key authentication
    2. DNS server will accept one parameter on the SSH connection, the FQDN to be modified
    3. The DNS server will pass valid ssh connections to a script which will authenticate them (again), then send an nsupdate command to the Bind service to update the IP address

The main purpose for my learning this was the need to be able to access various clients sites who had dhcp addresses. Even though they are (for the most part) public IP's, they changed all the time. I set up a cron job on each client machine to update their IP address every hour, which was more than sufficient for our purposes.

Note: The command *named-checkconf* is your friend. Use it often. This will allow you to validate configuration changes you make to the named configuration files before you restart (and accidentally kill) your name service.

```
named-checkconf /etc/bind/named.conf
```

. No message means it passes the checks.

Set up Bind9 for DynDNS

Bind9 allows external programs (and CLI commands) to modify a running system dynamically via the *nsupdate* utility.

If you are going to be updating from the server BIND is running on, the implementation is simple, and only slightly more complex if you want to control your BIND server from a different machine.

Control from same server

BIND now has the ability to dynamically modify a zone on the from the same machine with one simple added statment in the zone definition: *update-policy local*. Add this to your zone definition in named.conf and named will create a keyfile each time it starts. The location is implementation specific, but the default is */var/run/named/session.key*

```
zone "example.com" {
   type master;
   file "/var/cache/example.com.dns";
   update-policy local;
};
```

A user on the same machine with appropriate permissions may now do dynamic updates with

```
nsupdate -l
```

Control from second server

This will generate instructions to STDOUT.

```
ddns-confgen -z example.com
```

Following is an example of the output.

```
# To activate this key, place the following in named.conf, and
# in a separate keyfile on the system or systems from which nsupdate
# will be run:
key "ddns-key.example.com" {
    algorithm hmac-sha256;
    secret "HpLNKmpu3ioAXdIe4Tu0ol4y4lSllbiS+gDIjQ4KvW8=";
};

# Then, in the "zone" definition statement for "example.com",
# place an "update-policy" statement like this one, adjusted as
# needed for your preferred permissions:
update-policy {
      grant ddns-key.example.com zonesub ANY;
};
```

```
# After the keyfile has been placed, the following command will
# execute nsupdate using this key:
nsupdate -k <keyfile>
```

Take the top part (the key definition) and put it in two locations; in named.conf and as a separate file on the machine which will be issuing the commands. I try to keep the one in named.conf physically close (but before) the example.com zone definition.

The second part goes inside the example.com zone definition in named.conf. My final code may look something like this (combing part #1 and part #2 for named.conf)

```
key "ddns-key.example.com" {
    algorithm hmac-sha256;
    secret "HpLNKmpu3ioAXdIe4Tu0ol4y4lSllbiS+gDIjQ4KvW8=";
};

zone "example.com" {
   type master;
   file "/var/cache/example.com.dns";
   update-policy {
      grant ddns-key.example.com zonesub ANY;
   };
};
```

Testing

As some user who has permissions to make the modifications, execute the nsupdate command. How you call nsupdate depends on whether you are in local only mode, or whether you are on a different server than BIND is running on.

The nsupdate command (both versions) will drop you to an nsupdate command line interpreter. See *man nsupdate* for a list of commands. After that, everything you enter will be interpreted by nsupdate until you type the *quit* command.

```
# If you are working on the same server, just one command is needed
nsupdate -l
```

```
# for a remote server, use the following instead
/usr/bin/nsupdate -k /full/path/to/keyfile
# then, connect to the server
server server.domain.tld
```

Now that you are in nsupdate, you can issue some commands. Following is an example of changing an A record for a server. *help* gives you some very, very basic help

```
zone example.com
update delete server1.example.com A
update add server1.example.com 3600 A 192.168.145.1
show
```

```
send
answer
quit
```

- *zone example.com* sets the zone for the following commands
- *update delete server1.example.com A* deletes all A records for server1.example.com
- *update add server1.example.com 3600 A 192.168.145.1* adds an A record for server2.example.com with a TTL of 3600
- *show* displays which updates are in the queue
- *send* updates the zone
- *answer* displays the result of the last *send*
- *quit* exits the shell

Note that your zone file may not be updated immediately. Instead, the information is placed in the jnl file (journal) and updated at a later date. You can verify the changes with dig or nslookup, ie

```
dig @localhost server2.example.com
```

Automation

There are many ways of doing this. The client machine simply makes a connection to the server with enough information to allow the server to then update the DNS key.

The following example uses SSH, available on all Unix operating systems (including Mac OS X), and with a third party add on for Microsoft Windows. A client makes a connection to the server, passing one or two parameters, then the server takes those parameters and decides what to do.

To start this, the client creates a public key pair and gives the public part of the pair to the dyndns systems administrator. If the client wants to automate this, they set the password to nothing (press the enter key when asked). To create a public key pair on a unix machine, issue the following command as the user who will be making the update calls. It can be any user. Just press the Enter key at all the prompts (take the defaults).

```
ssh-keygen -t rsa -b 4096
```

Once complete, look for the file ~/.ssh/id_dsa.pub (or id_rsa.pub if you chose rsa) and send that to the administrator of the dyndns server.

The administrator adds a line similar to the following in their .ssh/authorized_keys

```
command="/opt/updatedns/updatedns", CONTENTS OF id_rsa.pub
```

The 'CONTENTS OF id_rsa.pub' in the above is the exact copy of id_rsa.pub from the client. **Note:** there must be a space between the comma of 'command=“/opt/bin/updatedns”,' and the contents of id_rsa.pub.

The 'command=“/opt/updatedns/updatedns”, ' forces any connection made using that public key to be passed to updatedns (assuming it is in /opt/bin) for processing.

The following is the sample script. While this is acceptable, there are some security enhancements

which could be added. An updated version of this script is available at

```
svn export http://svn.dailydata.net/svn/sysadmin_scripts/trunk/updatedns
```

. The script is released under the 3 Section BSD license, so feel free to use/abuse it any way you like.

NOTE: for an added security enhancement, you can require uses to “register”, and you manually put the entry in the status file. Then, modify the final if..else (about line 176)

updatedns

```
#! /usr/bin/env perl

use warnings;
use strict;

use Sys::Syslog;
use version
our $VERSION = version->declare( '2.0.0' );

#  Copyright (c) 2021, R. W. Rodolico
#
#  Redistribution and use in source and binary forms, with or without
#  modification, are permitted provided that the following conditions
#  are met:
#
#      Redistributions of source code must retain the above copyright
#      notice, this list of conditions and the following disclaimer.
#
#      Redistributions in binary form must reproduce the above
#      copyright notice, this list of conditions and the following
#      disclaimer in the documentation and/or other materials provided
#      with the distribution.
#
#  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
#  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
#  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
#  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
#  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
#  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
#  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
#  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
#  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
#  POSSIBILITY OF SUCH DAMAGE.[9]

# called as an ssh command
# ssh my_server hostname [ip]
# where hostname is the host name to be processed
# if IP is not given, will take the source IP from the ssh connection
```

```
# Can also be called from command line as
# ./updatedns hostname ip

# version 2.0.0
# 20210417 RWR
# major modification to use one script for local and remote
# also, added ability to set up a configuration file


# we'll store the configuration here, so we can load it easily from
# a config file, if it exists
my %configuration = (
   'local'  => 1,
   'domain' => '',
   'server' => '',
   'keyfile' => '',
   'ttl' => 3600
   );


############################################################
# you should not have to change anything below here. #
############################################################

# get script path so we can find the auxiliary files
use Cwd qw(abs_path);
use File::Basename;
my $appDir = dirname(abs_path(__FILE__)) . '/';

my $statusFile = $appDir . 'updatedns.status'; # stores current status
of known entries
my $template = $appDir . 'updatedns.template'; # a template used for
commands to nsupdate
# If local set, use -l parameter, otherwise, use the key file
my $nsupdate = `which nsupdate`; chomp $nsupdate;
my $configFile = $appDir . 'updatedns.conf';

my $hostname;
my $realIP;
my %entries;

# this reads an INI type file in the form
# key:value
# where : is any delimiter
# it then merges it into %$hash, overwriting anything with the same
# key with a new value
sub readFile {
   my %temp;
   my ($filename,$hash,$delimiter ) = @_;
   $delimiter = ':' unless $delimiter;
   return unless -f $filename;
```

```
   open CONF,"<$filename" or die "Could not read existing configuration
file $filename: $!\n";
   # turn the delimited file into a hash
   chomp( %temp = map { split $delimiter } grep{ !/^#/ } my (@a) =
<CONF> );
   close CONF;
   # merge into existing hash, leaving items not common alone
   @$hash{keys %temp} = values %temp;
}

# merge the template file with our values and save it to a file
# in /tmp/hostname.nsupdate (overwriting if necessary)
# Call nsupdate with the filename as a parameter.
# leaves the file in /tmp for debugging
sub doUpdateNS {
   my ($hostname, $ip) = @_;
   my $nsupdateFileName = "/tmp/$hostname.nsupdate";
   # grab the template
   open TEMPLATE, "<$template" or die "could not open template
$template for read: $!\n";
   my $template = join( '', <TEMPLATE> );
   close TEMPLATE;
   # now, replace our keys with our current values
   $template =~ s/\{hostname\}/$hostname/gi;
   $template =~ s/\{server\}/$configuration{'server'}/gi;
   $template =~ s/\{zone\}/$configuration{'domain'}/gi;
   $template =~ s/\{ttl\}/$configuration{'ttl'}/gi;
   $template =~ s/\{ip\}/$ip/gi;
   # save the file
   open OUTPUT, ">$nsupdateFileName" or die "Could not create
$nsupdateFileName: $!\n";
   print OUTPUT $template;
   close OUTPUT;
   # execute nsupdate, and return the results to the caller
   return `$nsupdate $nsupdateFileName`;
}

&readFile( $configFile, \%configuration, '=' );

# prepend 'server ' to the server command if it exists and we are not
local
$configuration{'server'} = "server $configuration{server}" if
$configuration{'server'} and not $configuration{'local'};
# and set nsupdate for local or remote
$nsupdate .= $configuration{'local'} ? ' -l ' : " -k
$configuration{keyfile} ";


#foreach my $key ( sort keys %configuration ) {
#   print "$key\t$configuration{$key}\n";
#}
```

```
# die;

# user should send hostname as a parameter on the command
# user may also send the IP

if ( defined $ENV{'SSH_ORIGINAL_COMMAND'} ) {
   ($hostname,$realIP) = split( ' ', $ENV{'SSH_ORIGINAL_COMMAND'});
   $realIP = $ENV{'SSH_CLIENT'} unless $realIP;
} else {
   $hostname = shift;
   $realIP = shift;
}

die "Invalid call\n" unless $hostname and $realIP;
# validate IP is valid IPv4
# NOT a very good regex for that, but ...
$realIP =~ m/^([\d.]+)/;
$realIP = $1;

#die "$hostname\t$realIP\n";

# Start logging to syslog
openlog('updatedns', 'cons,pid', 'user');

unless ( $hostname ) {
   syslog( 'FATAL','%s',"no hostname passed in from IP $realIP" );
   die "Invalid Invocation: hostname\n" ;
}

# validate the hostname is part of the $domain
if ( $hostname !~ m/[a-z0-9-]\.$configuration{'domain'}/ ) {
   syslog( 'warning', '%s', "Attempt to set incoming server name to
invalid host [$hostname]");
   exit;
}
# slup in the status file

&readFile( $statusFile,\%entries, "\t" );

# is the entry for $hostname alread set? Do nothing except log it
if ( ( exists $entries{$hostname} ) && ( $entries{$hostname} eq $realIP
) ) {
   print "$configuration{'domain'}: Already set to the correct IP:
[$realIP]\n";
   syslog('info', '%s', "Already set to the correct IP, [$hostname] =
[$realIP]");
} else { # we have a new IP, or possibly a new hostname, so set it
   $entries{$hostname} = $realIP;
   open STATUS, ">$statusFile" or die "could not open $statusFile:
$!\n";
   foreach my $key ( keys %entries ) {
```

```
      print STATUS "$key\t$entries{$key}\n";
   } # foreach
   my $output = &doUpdateNS( $hostname, $realIP );
   print "$hostname updated to $realIP, results are\n$output\n";
   syslog('info', '%s', "$hostname updated to $realIP" );
}
closelog();

1;
```

This script reads a template file and “fills in the blanks”, then runs the file through nsupdate. A sample template file follows. Items between curly braces (**{}**) are replaced by the script.

updatedns.template

```
{server}
zone {zone}
ttl {ttl}
update delete {hostname} A
update add {hostname} {ttl} A {ip}
show
send
; answer
```

And, you can optionally include a configuration file which will override the defaults built into the code. The configuration file determines whether it is in local or remote mode, with different values needed. Use key/value pairs separated by an equals sign (ie, old INI file format).

updatedns.conf.sample.local

```
local=1
domain=example.com
```

updatedns.conf.sample.remote

```
local=0
domain=example.com
ttl=3600
keyfile=/etc/keys/Kdyndd.net.+157+35226.key
# this uses the remote, but on localhost
# bascially, the equivilent of -l if it was set
# up for bind to create the key file
server=127.0.0.1
```

Links

- https://tecadmin.net/configure-rndc-for-bind9/
- https://www.zytrax.com/books/dns/ch7/xfer.html#allow-update
- man nsupdate
- man ddns-confgen