HTTPD TCP Bridge

Design Document

Contact: Greg Badros <badros@gmail.com>

Last Updated: 2017-08-27

Summary

The HTTPD TCP Bridge enables interaction with TCP-based protocols especially for embedded and home-control devices and controllers that lack HTTP-based APIs or have better-designed and/or simpler TCP-based interactions. The Bridge also serves as a webhook target for arbitrary interactions and can have an ngrok-like endpoint configured at a public IP address for proxying into the home via an outbound connection. It can also dequeue requests off of an Amazon SQS channel. Oauth2 for the public access point is also supported to provide access control. Parameters modifying the default behaviour of each bridged TCP command can be specified on a per-request basis, but defaults per host:port can be pre-configured in a simple line-based configuration file.

Problem Statement

TCP Bridge Design

An HTTP (with or without SSL) request to the Bridge has a path component of the form:

/*host*:*port*/*data*[?*params*]

Where **host** is the hostname or IP address of the destination and **port** is the destination port at that host to receive the TCP command. After the first slash is the **data** of the TCP command -- this is the literal characters to send to that TCP socket. They are, of course, URI-encoded once in the URL enabling any octet to be part of the command. The extra URL parameters, **params**, include:

html ::= return TCP response line as an HTML response type embedded in an `<html><body>`***response***`</body></html>` envelope. The response is, of course, HTML-escaped inside that envelope. This option makes no changes to the TCP interaction and only affects the HTTP response sent by the bridge server.

uridecode ::= do a single URI unescaping of the TCP response before returning the response. The response may still be subject to HTML-escaping if the **html** parameter is specified.

singlesend ::= send the entire **data** field as a single TCP write to the socket. Contrast this with the default which is to split on `\n` and do a separate send for each line of **data**.

addlf ::= Add a trailing linefeed (i.e., "`\n`") to each data chunk sent to the TCP socket. This can be especially useful when **singlesend** is not specified so the `%0a`'s appearing in the data field of the URL are used as separators while the actual TCP command language that the target server uses requires linefeeds to terminate the lines.

Configuration File

There is a single per-user configuration file, `~/.httpd-tcp-bridge`, that controls the behaviour of the server. The format of that file is three **`^L`**-separated sections: 1) a list of configuration rules, one per line; 2) a list of host/port aliases, one per line; and 3) a list of data aliases, one per line.

Configuration Rules (section 1 of config)

The parameters that can be specified on each HTTP request can also be set via the first section of the configuration file called `~/.httpd-tcp-bridge`. one configuration rule per line, and each configuration rule contains TAB-separated columns. The first column is a regular expression that must match the *host*:*port* of the TCP command, and the following columns are any of the supported URI parameters (e.g., **html, uridecode, addlf, singlesend**) possibly prefixed by "!" to mean "turn that option off". All matching rules in the config file are applied, in order from top to bottom of the file (but see also the "stop" pseudo-parameter).

There are also special parameters that can only occur in the columns of the configuration file: **forcedone**. This work as follows:

forcedone ::= ends processing of the configuration parameters immediately at this line so that rules specified in later lines cannot add nor change parameters for host:dest that match the current rule.

Note that configuration rules override any parameters specified on the command-line, though you have to specify all of the options in the configuration file to override all parameters.

Host and Data Aliases (sections 2 and 3 of config)

The second section of the configuration file is two TAB-separated columns. Column 1 is an identifier and column 2 is a *hostname:port* that that identifier expands into. The host alias is

expanded if and only if the URL path begins with a **/d** or **/D** pathname component. For example, if section 2 of the config file contains the line:

```
GCIR_GreatRoom   192.168.0.248:4998
```

Then fetching **http://localhost:7316/D/GCIR_GreatRoom/getdevices%0d** will be identical to the request
http://localhost:7316/192.168.0.248:4998/getdevices%0d

The third section of the config file is data aliases in the same format, except that column 1's identifier is searched for in the data (rather than the host/port) portion of the URL and expanded there in place. The expansion of the data aliases happen if and only if the URL path begins with a **/d** pathname component (/D only does the host alias expansion). For example, given the prior hostname alias in section 2 and this data alias in section 3:

```
Onkyo_VolDown    sendir,1:3,1,38000,…%0d
```

Then fetching **http://localhost:7316/d/GCIR_GreatRoom/Onkyo_VolDown** will be rewritten to and identical to fetching
http://localhost:7316/192.168.0.248:4998/sendir,1:3,1,38000,…%0d

The alias names are restricted to the regex **\w+.** All other parameters/options are interpreted after the textual expansion occurs (e.g., adding the trailing

Global Options to the Bridge

There are several global options to the server, specified as command line options. These are:

`-H` Default to text/html responses
`-U` Default to url-unescaping the TCP response
`-s` Default to doing a single socket send
`-p` *PORT* Use PORT for the HTTP service (defaults to 7316)
`-P` Promiscuous mode lets any destination be sent to even if it's not in the config file
`-R REMOTE_SERVER:PORT`
Initiate TCP forwarding from REMOTE_SERVER:PORT to this service

Tested Destination Services

The Bridge is intended to be general purpose for any TCP service, but was developed and tested against two line-based TCP protocols specifically: 1) the Logitech Media Center's command line interface (formerly squeezebox server, former formerly slimserver) operating on

port 9090 by default; 2) The Vantage InFusion lighting controller operating on port 3001 by default; and 3) GlobalCache IR/Contact-Closure devices. In particular, by exposing an HTTP interface to those services, it enables HTTP-interactions from a Samsung SmartThings Hub.

Implementations

The first implementation was in Perl 5 (~/bin/share/httpd-tcp-bridge) and that remains a reasonably complete implementation, but I've stopped work on it.

The second and current implementation is written in Groovy using the JVM runtime. It is currently (12/27/16) complete and will be extended over time.

Future Work and Ideas

- Integrate the vantage_http_fetcher.pl TCP->HTTP functionality into this service by having it also listen on a TCP port for TCP commands to be turned into HTTP requests (or maybe that should remain a separate service)
- Listen to globalcache's port 4998 for TCP commands to impersonate a globalcache device to be able to learn from the commands that other tools send to the IR device (e.g., SimpleControl's device commands could get learned by this service pretty readily)
- Make the aliases (both host:port and data) able to be set via a REST API and store them in a simple database rather than only in a config file.
- Introspect on the config file (and/or the real-time settings from the DB)