Tcl in the Middle

Michael A. Cleverly
Intermountain Healthcare
4646 W Lakepark Blvd
Salt Lake City, Utah 84120
michael.cleverly@intermountainmail.org

September 27, 2007

Abstract

Tcl has long been recognized as an excellent language to glue existing components together to create new applications. Tcl is just as useful when interjected into the middle of functioning N-tier "enterprise" systems.

SockSpy[1] is probably the best known[2] example of a "man in the middle" Tcl application. What may not be as widely appreciated, however, is that Tcl's strong TCP sockets and event-drive I/O make construction of custom "man in the middle solutions" (or proxies) quite straightforward.

These custom solutions solve real business problems often at a fraction of the cost of other potential solutions. This paper will look at a handful of examples where Tcl has been employed in this manner at Intermountain Healthcare.

1 About Intermountain Healthcare

Intermountain Healthcare[3] is a non-profit integrated healthcare delivery system headquartered in Salt Lake City, Utah and serving communities throughout Utah and southeastern Idaho. Intermountain employs over 26,000 people throughout its system of hospitals, clinics and healthplans.

Founded in the 1970s, Intermountain has been pioneering the use of information technology in healthcare since that time. Intermountain is a recognized leader in its uses of information technology in healthcare[4]. In 2005 Intermountain began a 10-year partnership with GE Healthcare to collaborate and jointly design and develop new electronic health record technologies[5][6].

2 Facilitating a corporate rebranding initiative

Prior to November 2005 Intermountain Healthcare was known as "Intermountain Health Care," or more often simply IHC. A major corporate rebranding initiative was launched in the end of that month to remove the space between the words "health" and "care" and eliminate references to IHC as an acronym.[7]. Several months later the health plans division

formerly known as “IHC Health Plans” changed its name to SelectHealth[8] completing the corporate rebranding initiative.

Prior to the rebranding, Intermountain's principal public web site had been powered by Expressroom, a proprietary Java & XML based Content Management System (CMS). Expressroom has had a rocky history, being acquired and then sold by a series of different owners[9]. In short Expressroom was viewed internally as a deprecated technology that needed to be replaced going forward.

When the rebranding initiative was announced internally, the various departments responsible for content realized they would need to modify substantial amounts of content to rid all URLs of any reference to the IHC acronym.

The old home page was located at http://www.ihc.com/xp/ihc/; the new home page was to be located at http://intermountainhealthcare.org/xp/public/.

While standard web technologies (such as Apache's mod_rewrite[10]) could be used to redirect incoming links to legacy URLs two problems would have remained:

1. Extra overhead from absolute links within the existing content that would incur extra redirection overhead
2. The user's browser would still show the now verboten acronym when a user moused over a link

Since the departments responsible for the content did not have the man power to change all of the Expressroom content prior to the date of the public announcement, Intermountain's Enterprise Web Operations team adapted the opensource AOLserver[11] web server to act as a “rebranding proxy.”

For those unfamiliar with the particulars of AOLserver's Tcl API[12], it is (trivial) to delegate the handling of portions of the URL space to specified Tcl procedures (which are invoked once for each request).

```
ns_register_proc GET  / rebranding-proxy
ns_register_proc HEAD / rebranding-proxy
ns_register_proc POST / rebranding-proxy
```

In the above example, any URL requested at or under / (i.e., the whole site) will be processed by a Tcl procedure named “rebranding-proxy.”

The algorithm employed by our rebranding-proxy is quite straightforward:

1. Get the requested URL using *ns_conn url*
2. Use *regsub* to munge the new-style URL (/xp/public) into the old form Expressroom uses (/xp/ihc)
3. Make a connection to the Expressroom application server and request the munged URL
4. If the MIME content type returned by Expressroom matches text/* then use *string map* to fix up any embedded links using the old-style URLs by translating them to the new format
5. Return the response to the user passing along the same HTTP response code and most of the same headers received from Expressroom

It is worth noting that not all response headers should be returned from Expressroom to the user. Specifically, if we've altered any URLs within the HTML (or CSS or Javascript, etc.) then the Content-Length header will absolutely need to be recalculated.

3 A restricted authorization proxy for static content

Intermountain Healthcare has standardized (for the time being) on Vignette's Portal (VAP) and Content Management (VCM) products for audience-focused intranet and extranet dynamic portals[13].

In this model, portlets running within the VAP application server consume dynamic content from the VCM database and render it to HTML. The VAP application servers are front-ended by multiple Apache web servers. Static VCM content, for performance reasons, is pushed out directly to the web servers and served by Apache.

As long as the static content was benign images or content meant for public (unauthenticated) consumption, this approach to managing static content was acceptable. Over time as portal adoption grew, some business units wanted to begin publishing static content that needed to be restricted more tightly than merely through the obscurity of its URL.

Based on the success of using AOLserver as a "rebranding proxy" Intermountain's Enterprise Web Operations team constructed a similar proxy, dubbed "portal-rproxy" to restrict access to static content to logged in authenticated users. Rather than having Apache retrieve files directly from the local filesystem we used mod_proxy[14] to point Apache to an AOLserver instance running on a non-privileged port bound to the local loopback interface.

Users who are authenticated to VAP will have a session id cookie. This cookie is only transmited over SSL between the user and the web server to prevent eavesdropping attacks. Because the SSL connection has already been terminated by Apache, *portal-rproxy* has access to it in cleartext. We leverage this fact to call forward to the VAP application server to see if the session is still valid and active; only if it is, do we serve up the static content from the file system (using AOLserver's *ns_returnfile* API call).

To make it possible for system administrators to change access restrictions without needing to modify Tcl code, we created two configuration files that are read at runtime: *noauth.conf* and *restrict-per-site.conf*.

The *noauth.conf* file contains a list of regular expression patterns. Blank lines, lines made up of only whitespace and lines where the first non-whitespace character is a # (i.e., comments) are ignored. Such a file would look like:

```
# Allow all access to .css stylesheets
\.css$

# Logos, etc. for the public areas of the portal
^/Public/Images/
```

The *restrict-per-site.conf* file contains URL patterns and a list of portal subsite(s) the logged in user must have access to in order to retrieve the content. A hypothetical entry in such a file would look like:

```
restrict-url "^/Surgery/Schedules/" to physicians
```

For performance, the results of a succesful check of the validity of a session id can be cached for a few minutes (to reduce the overhead of repeatedly checking while retrieving multiple static assets referenced from a single portal page). We recommend keeping this cache window fairly low (no more than several minutes) to limit the amount of time static content could be refreshed after a user has logged off.

As with any web application that uses session cookies if an attacker can obtain the session cookie (i.e., via an Cross Site Scripting (XSS) attack[15]) they effectively become the user.

Our portal-rproxy neither widens this risk (as compared to having the application server handle the static content) nor does it mitigate it any.

4 Pseudo source-NAT'ing with tcpsymlinks

The basic skeleton of a functioning man in the middle proxy in Tcl can easily be written to fit on a single printed page[16]. For example:

```
socket -server accept listeningPort

proc accept {client addr port} {
  if {[catch {socket -async destHost destPort} server]} then {
        shutdown $client
    } else {
     fconfigure $client -blocking 0 -buffering none -translation binary
     fconfigure $server -blocking 0 -buffering none -translation binary
      fileevent  $client readable [list glue $client $server]
      fileevent  $server readable [list glue $server $client]
    }
}

proc glue {src dst} {
    if {[catch {puts -nonewline $dst [read $src]}] ||
        [eof $src] || [eof $dst]} then {
        shutdown $src $dst
    }
}

proc shutdown {args} {
    foreach sock $args {catch {close $sock}}
}

# Enter the event loop
vwait forever
```

Just as a symbolic link (symlink) in a file system serves as "a special type of file that serves as a reference to another file"[17], we introduce the notion of a "tcpsymlink" which is just a listening port on a particular IP address that proxies traffic to another address and port.

Our *tcpsymlinkd* daemon looks in a *ports/* directory for files named either *description.port* or *description.port.interface*. The *description* portion of the filename serves merely as documentation for those administering the server the daemon is running on. The daemon listens on the specified port on either all interfaces or the one specified.

Each *ports/* file is expected to contain one line containing the hostname (or IP address) followed by a space and a port number. When a new connection comes in a new outgoing connection is made to this location and the two sockets are "glued" together in much the same way as the code skeleton above shows.

The daemon polls periodically (typically every fifteen seconds) to see if any of the *ports/* configuration files have changed, been deleted, or added. Changes only affect future connections; existing connections are not disturbed. If a *ports/* file has been deleted, the listening

socket is closed preventing future connections. Likewise if a new *ports/* file has been created, a new listening socket will be opened.

On occasions when the daemon needs to listen on a prvileged port, it must be started as root. In these cases it is recommended to drop root privileges as soon as the initial listening sockets are opened. Either the TclX extension[18] or a small C extension (such as the one included with TclHttpd[19] or one written using Critcl[20]) can be used to *setuid* to a non-privileged user.

5 A Tcl web server with a One Track Mind

OTM[21] is a web server written in Tcl that answers all requests in exactly the same way. This turns out to actually be a useful feature, especially when combined with tcpsymlinks.

Instead of having a web server configured to front-end an application server, we instead have the web server talk to a tcpsymlink which in turn talks to the application server. When it comes time to do periodic scheduled maintenance, a helpful downtime message can be served up using OTM and the tcpsymlink can be temporarily repointed away from the application server. When the downtime is over, the tcpsymlink can be changed back. All of this can be done without changing any of the configuration settings of either the web server or the application server and is potentially less error prone.

6 Front-ending an existing system with SSL

Another man in the middle use where Tcl shines is the ease with which existing applications can be extended to support SSL connections using the TLS extension[22].

In the simplest case, an application that already calls Tcl's native *socket* command need only call *::tls::socket* instead, possibly specifying some additional SSL-specific configuration options.

One recent implementation of an SSL-enabled proxy at Intermountain dealt with a new radiology image viewer. Several hundred physicians (not directly employed by Intermountain but who have admitting rights at Intermountain hospitals) and their offices and clinics have hardware VPN tunnels that allow them to access certain components of Intermountain's Electronic Medical Record (EMR) software.

The existing EMR system, written in Java, added significant overhead to the transmission of images because it would download the entire image from the radiology system, buffering it in memory before it would send any data back to the web server (which would only then begin to transmit the data back to the end user's browser). This added a roughly 10x penalty compared to directly accessing the radiology system.

Ideally the images would be transfered via an SSL connection over the hardware tunnel. Although the hardware VPN provides encryption for the data as it traverses the public internet, at the other end of the VPN tunnel the traffic is no longer protected to any eavesdroppers on the local LAN. End-to-end SSL encryption thus provides additional security against eavesdropping.

Using the existing SSL-enabled Apache web servers that front-end the EMR to proxy the radiology images provided somewhat better performance than having the EMR handle the

images directly, but the speed was still 2x to 3x slower than retrieving them directly from the radiology system (without SSL).

Since the VPN tunnels require configuration on both ends of the tunnel, adding a new address (say that of the radiology system itself) would require coordination between both Intermountain engineers and the (often contracted) IT staff that the affiliated physicians employ to manage their computer equipment. Coordinating and implementing such a change could easily have consumed 500-man hours of labor.

Instead, using Tcl code basically equivalent to the man in the middle skeleton shown previously, with *socket -server* replaced with *tls::socket -server* running on the same web servers on a high port (so no VPN tunnels needed to be reconfigured) affiliates were able to access the radiology image viewer using SSL. The TLS package was compiled to take advantage of the hardware SSL acceleration cards already present in the web servers. The net result is that there is no noticeable difference between requesting an image through our SSL-enabling Tcl proxy compared to accessing the radiology system (non-SSL) directly.

7 Deterministic load balancing

Our final example of using Tcl as a man in the middle proxy involves deterministic load balancing. If we have a pool of N application servers we choose an application server to route to by taking the final octet of the requester's IP address mod N and using the corresponding application server. For example:

```
set servers [list host1 host2 host3 host4]
set octet   [lindex [split $ip_addr .] end]
set choice  [expr {$octet % [llength $servers]}]
set backend [lindex $servers $choice]
```

Typically an application server vendor will supply a "plugin" for various web servers[23][24]. The job of the plug-in is to spread the load across the various backend application server instances. Since these plug-ins are traditionally proprietary and closed source, their algorithmic decision making process is somewhat opaque.

At Intermountain, we are phasing out our use of the WebLogic plug-in, replacing it with a Tcl man in the middle proxy we call "wlpr-proxy" (short for "WebLogic Plugin Replacement Proxy"). With the vendor's plug-in, active users would occasionally be redirected to a different application server instance for no apparent reason. This was very frustrating for end users (clinicians hate to have to repeat entry of lengthy patient notes merely because some piece of software routed them to the wrong destination). Since the problem was intermittent and not reproducible on demand, it posed a frustrating challenge for both quality assurance (QA) and support personnel alike.

WebLogic's plug-in can be configured to log debug data. On a busy web server, however, all the debug data from various connections quickly becomes intermingled and is nearly impossible to disentangle. Thus, a design requirement for our wlpr-proxy replacement was the ability to log each connection individually. We chose to have the capability of logging all client request headers (for GET, HEAD and POST requests) and server response headers (for GET and HEAD requests). For privacy reasons, we do not log any form data that the user POSTs or any of the servers response headers to the POST request.

The ability to reconstruct the precise sequence of end user requests and responses has proven to be a useful resource for both QA testers and developers. Several instances of obscure

corner-case bugs have already been identified via the enhanced logging that wlpr-proxy provides. In the past, because we lacked visibility, some of these bugs would have slipped through testing and into production.

8 Two caveat to keep in mind

When writing line-oriented proxies with Tcl versions prior to 8.5, it helps to keep in mind the potential Denial-of-Service (DoS) condition discussed on the Tcl'ers Wiki by Donald Porter and George Peter Staplin[25]. In non-blocking I/O mode a readable fileevent will trigger when new data is available on a channel even if an entire line is not available.

A malicious user could send excessively long lines (without ever sending a newline) forcing Tcl to eventually exhaust all of its available memory, panic and abend.

With the inclusion of TIP #287[26] in Tcl 8.5, a new subcommand of *chan pending* can be used to introspect how much buffered data is available to be read and enforce application-specific limits appropriately. For those interested (and some may not be since this problem is largely theoretical and rarely seen in the wild), prior to Tcl 8.5 several different mitigation techniques are possible:

1. Write a small C extension to expose the existing Tcl_InputBuffered and use that to introspect the amount of unread data
2. Set an event some number of seconds into the future using *after* and take some action (i.e., using *read* instead of *gets* or aborting the connection) if a complete line has not been read by then
3. Rewrite your application to use *read* instead of *gets*

The second caveat to keep in mind is that it is worth remembering to call *fblocked*: "The fblocked command returns 1 if the most recent input operation on channelId returned less information than requested because all available input was exhausted[27]."

If you are writing a proxy that inspects HTTP requests the end of the client's request headers is signified by a blank line[28]. When *gets* returns a blank line it could be because the line was blank (end of request headers) or there wasn't a complete line in the buffer. Calling *fblocked* (or in Tcl 8.5 *chan blocked*) allows the program to distinguish these two cases and avoid a failure to parse subsequent request headers.

For an example showing the use of both *chan blocked* and *chan pending* together, see this[29] December 2006 thread on the comp.lang.tcl newsgroup.

9 Not just IPv4 and TCP

Although the Tcl core itself only supports TCP IPv4 sockets various extensions exist which provide support for other protocols.

- TclUDP[30] provides UDP sockets and is available for both Windows and Unix systems
- IOCPSOCK[31] is a Windows extension providing faster IPv4 TCP sockets as well as IPv6 TCP and IrDA sockets
- Ceptcl[32] is a Unix-centric extension providing UDP, IPv6 and raw IP sockets
- hping3[33] is a low-level packet assembler scriptable with Tcl

10 Conclusion

We have seen that custom application-specific proxies can be quite easily written in Tcl. These Tcl solutions solve real business problems. Because of Tcl's powerful event-driven I/O model, Tcl solutions tend to be small and fairly easy to reason about.

References

[1] Poindexter, Tom, Keith Vetter, and Don Libes. "SockSpy." <http://sourceforge.net/projects/sockspy/>

[2] Laird, Cameron. "Sockspy Knows TCP/IP" Sys Admin December 2002. <http://www.samag.com/documents/s=7732/sam0212b/0212b.htm>

[3] About Intermountain: Serving Our Communities. Intermountain Healthcare. <http://intermountainhealthcare.org/xp/public/about-intermountain/>

[4] Intermountain Healthcare. "Report to the Community 2006." <http://intermountainhealthcare.org/xp/public/documents/corp/annualreport.pdf>

[5] Kozek, Andrea. "GE Healthcare & Intermountain Health Care To Provide Wide-Reaching IT System." 17 February 2005. <http://www.gehealthcare.com/company/pressroom/releases/pr_release_10225.html>

[6] Cowley, Daron. "GE Healthcare & IHC establish new research center to develop electronic health record technologies." 6 July 2005. <http://intermountainhealthcare.org/xp/public/about-intermountain/news/article26.xml>

[7] Intermountain Communications. "Intermountain Healthcare updates logo." 29 November 2005. <http://intermountainhealthcare.org/xp/public/about-intermountain/news/article6.xml>

[8] Intermountain Communications. "IHC Health Plans has a new name--SelectHealth." 3 April 2006 <http://intermountainhealthcare.org/xp/public/about-intermountain/news/article10.xml>

[9] "Expressroom Lives On..." CMS Watch. 21 July 2003. <http://www.cmswatch.com/Trends/224-Expressroom-Lives-On...>

[10] "Module mod_rewrite URL Rewriting Engine." The Apache Software Foundation. <http://httpd.apache.org/docs/1.3/mod/mod_rewrite.html>

[11] Davidson, Jim. "Tcl in AOL Digital City: The Architecture of a Multithreaded High-Performance Web Site." 16 February 2000. <http://www.aolserver.com/docs/intro/tcl2k/html/>

[12] AOLserver Tcl API Reference. <http://www.aolserver.com/docs/devel/tcl/api/>

[13] Smith, Ryan. "Intermountain Healthcare: Audience-Focused Dynamic Portals." 24 October 2006. <http://www.vignettevillage.com/Austin/ConferenceSessionDetails.html>

[14] "Apache module mod_proxy." The Apache Software Foundation. <http://httpd.apache.org/docs/1.3/mod/mod_proxy.html>

[15] Fogie, Seth, Jeremiah Grossman, Robert Hansen, Anton Rager, and Petko D. Petkov. Cross Site Scripting Attacks: XSS Exploits and Defenses. Syngress, 2007.

[16] Cleverly, Michael A. "The skeleton of a man in the middle." Weblog Entry. Cleverly Blogged. 12 March 2007. <http://blog.cleverly.com/permalinks/285.html>

[17] "Symbolic link." Wikipedia: The Free Encyclopedia. 17 August 2007. <http://en.wikipedia.org/wiki/Symbolic_Link>

[18] Lehenbauer, Karl et al. "TclX." <http://tclx.sourceforge.net/>

[19] Welch, Brent. "TclHttpd." <http://tclhttpd.sourceforge.net/>

[20] Landers, Steve. "Access C library functions using Critcl." Tcl'ers Wiki. 5 April 2004. <http://wiki.tcl.tk/11227>
[21] Cleverly, Michael A. "OTM: A web server with a *One Track Mind*." Weblog Entry. Cleverly Blogged. 26 June 2005. <http://blog.cleverly.com/permalinks/158.html>
[22] Newman, Matt et al. "TLS extension." <http://tls.sourceforge.net>
[23] "Using Web Server Plug-Ins with WebLogic Server." BEA WebLogic Server 8.1 Documentation. 2003. <http://e-docs.bea.com/wls/docs81/plugins/>
[24] Cocasse, Sharad and Makarand Kulkarni. "Understanding the WebSphere Application Server Web server plug-in." October 2003. <http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/pdf/WASWebserverplug-in.pdf>
[25] Staplin, George Peter and Don Porter. "Using gets with a socket is a BAD IDEA." Tcl'ers Wiki. 24 October 2001. <http://wiki.tcl.tk/1183>
[26] Cleverly, Michael A., Donal K. Fellows, ed. "TIP #287: Add commands for Determining Size of Buffered Data." Tcl Improvement Proposal. 26 October 2006. <http://www.tcl.tk/cgi-bin/tct/tip/287.html>
[27] "fblocked manual page." <http://www.tcl.tk/man/tcl8.4/TclCmd/fblocked.htm>
[28] Fielding, et al. "Hypertext Transfer Protocol--HTTP/1.1." RFC 2616. June 1999. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5>
[29] Cleverly, Michael A. "Re: TIP #287: Add a Commands [sic] for Determining Size of Buffered Data." comp.lang.tcl newsgroup. 14 December 2006. <http://groups.google.com/group/comp.lang.tcl/msg/e5e4d9cf8842a092>
[30] Thoyts, Pat and Xiaotao Wu. "TclUDP." <http://tcludp.sourceforge.net/>
[31] Gravereaux, David. "IOCPSOCK." <http://iocpsock.sourceforge.net/>
[32] Cassoff, Stuart. "Ceptcl." <http://www3.sympatico.ca/stuart.cassoff/software/>
[33] Sanfilippo, Salvatore. "Hping3." <http://wiki.hping.org/>