Name: SGT Schoenwald-Oberbeck, Jesse
Date: 27JUN2017
Current Module: Networking using C
Project Name: ashti

Project Goals:

Build a web server which listens on a port based on UID. The server should be able to serve html to netcat or a browser regardless of the source being a flat html file or the result of a cgi script.

Considerations:

- o Transmissions should be handled by TCP.
- o C networking can be tricky.
- o File sizes can vary dramatically..
- o Differing file types will need to be handled appropriately.
- o The correct directory will need to be used.

Initial Design:

The network functionality will primarily be handled by ashti.c. This will build, maintain, and close the connections to clients appropriately. Filehandler.c will extract the file name requested from the GET request, and send the proper data piece by piece.

Connections are handled by forks in ashti.

Data Flow:

A TCP connection is built by ashti, and the data and socket is handed over to filehandler, which accesses, breaks apart, and sends the requested files in easily managed chunks, which should likely never need fragmenting.

Communication Protocol:

TCP over bound port. Specifically on loopback.

Potential Pitfalls:

- o Networking in C
- o Sending variable sized files.
- o Being able to work with both netcat and a standard browser.

Test Plan:

User Test:

Checks for access to both html and cgi from both netcat and the Chrome browser.

Test Cases:

All test cases completed with correct output.

Conclusion:

Networking with C is tedious and full of potential for all manner of error. Fortunately TCP inherently handles a good number of issues, and much more appropriate for this use since the files can potentially be very large.

The difference between HTML and CGI was actually much easier to handle than anticipated thanks to popen, which conveniently had the same parameters as fopen.

Buffer size was set to an arbitrary/easily-handled low number, and files were broken apart and sent according to this buffer size. This was to handle files too large for a single packet (presumably most files…) and came together easily.

I ran into a fair number of small, but hard to hunt down errors, including the need to send two newlines after the Content-type line, which seems arbitrary and weird. Further difficulties came from all the possible variants of the GET request, including the simple “/” sent by a browser when no file is specified.