Instructors: Matt Fredrikson TA: Tianyu Li

Due date: 02/01/2018 at 11:59pm

Lab 0

1 Not-so-tiny Server

After taking 15-213, you are very excited about the server technology presented to you. You are convinced that this will be the next big thing in tech. To get a head start on your peers, you have decided to “borrow” the 213 code for tiny server and add some cool features to it. This way, you can beat them to market and achieve some creative disruption.

Recall that tiny server serves both static and dynamic contents. However, static contents are boring. You decided to delete all static content related code and focus on the dynamic part. Tiny server came with support for Common Gateway Interface(cgi) programs. How cool would it be if you can exchange information with your friends on a tiny server? You decided to extend tiny server’s capabilities with a persistent key-value store and the appropriate programs to access it.

Key-Value Store

A key-value store is a simple database structure that is perhaps better known as dictionary or map. It has been decided that string to integer mapping is the only mapping that people will ever need. Perhaps the easiest way to implement a key-value store is through the use of an extendible hash table. For simplicity’s sake, you did not optimize the data structure for disk performance, but instead serializes the table to a db file in human readable format (⟨key⟩ ⟨value⟩, with one space character in between, one entry on each line). Example can be found in the code handout.

Where? Give filename.

Server Structure

Caps What does this mean?

- the general server architecture “references” tiny server. The main server code is written in `tiny.c` where a main server loop sequentially handles incoming connection, parses the query string, and spins up the correct cgi program in a separate process. ~~The server does not handle multiple connections concurrently because concurrency is hard.~~

  plural Caps

- Like tiny server, not-so-tiny server forks a process and calls `execve` on the binary program requested by the client. Arguments are passed on the command line. exec doesn't have cmd line

  Be more precise. Server accepts all connections, doesn't have authentication

- For now, anyone can accesss your server and everyone shares the same database backed by the same file. To achieve full functionality, the only thing you need to do is to implement two new cgi programs, `get` and `store`, reading from and writing to the underlying database, respectively.

Task

You have already scoped out the architecture for your not-so-tiny server, all that remains is to write the code! Fill in all the functions unimplemented in `cgi-bin/extendible_hash.c`, `cgi-bin/get.c`, `cgi-bin/store.c`

Didn't talk about this before. Need to specify functions and informal contracts, otherwise it will be hard to grade.

*Comment the functions in these files with descriptions of their behavior. Even the ones whose code is provided...

2 Not-so-fast Deployment

You are very excited after you are done, and decided to start testing your new server.

Alas, somebody on the 15316 staff was feeling evil and inserted random bugs into your data structure code when you were not looking. Writing a thorough test suite seems like quite a project, and tests only show the presence, not the absence of bugs anyway. A slightly better approach, of course, is to use what you learned in 15316 to complement testing.

Fortunately, the 15316 staff is not completely evil and left you with a bounded model checker along with a tutorial on how to use it. Give Filename

Task

- Use CBMC to find and fix any memory safety errors and general bugs in `ubarray.c` and `extendible_hash.c`, and verify your fixes up to a reasonable bound. ~~A tutorial to CBMC can be found in cbmc.pdf.~~ If you are not familiar with extendible hash maps or unbounded arrays, read `https://en.wikipedia.org/wiki/Extendible_hashing` and `https://en.wikipedia.org/wiki/Dynamic_array`. This should go in the previous section.
- Your finished artifact should be two code files `ubarray_verified.c` and `extendible_hash_verified.c` with fixes and the code you used to verify the files. Also, record the command you used to verify them along with the console output in `ubarray_output.log` and `extendible_hash_output.log`, respectively.
- ~~You may want to use the command line argument `--slice-formula` to speed model checking. `--trace` provides extra information for any bugs found.~~ Move this to the tutorial. They should read the whole thing anyway.

3 Not-so-secure Security

You are very happy that cbmc gives you the final check mark on your server. Your not-so-tiny server is finally ready to roll!

Caps

...or is it?

Task

- In a text file, write down bugs or security concerns that the model checker is unlikely to find out about and how they can be problematic. Name this `report.txt`. For each class of issue you take notice of, explain how other methods, such as testing, can be utilized to help uncover these issues.

- For bonus points, give a query string, or a program that sends query strings, that would exploit one of the bugs to break the server, or touch things it's not supposed to touch. You are not allowed to add cgi programs to make it happen. Said bug must not be introduced through your own code.

We didn't say what shouldn't be touched. Have them explain why, or in what sense, their attack is concerning.