Instructor: Matt Fredrikson TA: Tianyu Li

Due date: 02/08/2018 at 11:59pm
Total points: 100

Lab 0

1 Not-so-tiny Server (50 points)

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). 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`. Example for the db file format can be found in the code handout (`example.db`).

Server Structure

- The main server code is written in `tiny.c` where a main server loop sequentially handles incoming connections, parses the query string, and spins up the correct CGI program in a separate process.
- Like tiny server, not-so-tiny server forks a process and calls `execve` on the binary program requested by the client. Command line arguments are passed in the execve call.
- For now, no access control policy needs to be implemented. 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` (see `include/extendible_hash.h` for specification), `cgi-bin/get.c`, `cgi-bin/store.c`

2 Not-so-fast Deployment (35 points)

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.

Task

- Use CBMC to find and fix any memory safety errors in `ubarray.c` and `extendible_hash.c`. Consult the tutorial and CBMC documentation for information on how to configure CBMC to check for this class of bugs.
- 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.

3 Not-so-secure Security (15 points)

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!

...or is it?

Task

- In a text file, write down bugs or security concerns that the model checker is unlikely to find without additional specification in the form of `__CPROVER_assert` specifications, and how they can be problematic. You should assume that the attacker has the ability to interact with the server by sending arbitrary data to it over the network, and that the only acceptable behavior of the server is to process valid requests to execute programs already in `cgi-bin`. Any other behavior that the server can be coerced into executing should be considered a violation of the (not yet precisely-defined) security policy.

  Save your thoughts in `report.txt`. For each issue you take notice of, explain why the model checker is either unable to find it or why doing so would require additional specification, and how other methods such as testing can be utilized to help uncover these issues.

  Note: If you are unable to identify any such issues in the server implementation, you will receive partial credit for identifying potential security issues and explaining why the model checker is unlikely to find them.

- For bonus points (**10 points**), give a query string, or a program that sends query strings, that would exploit one of the bugs to break the server, and explain why or in what sense the attack is of concern. You are not allowed to add cgi programs to make it happen. Said bug must not be introduced through your own code.