Programming Problem: Secure Log File

Summary

Players will implement a *secure log* to describe the *state of an art gallery*: the guests and employees who have entered and left, and persons that are in rooms. The log will be used by *two programs*. One program, `logappend`, will append new information to this file, and the other, `logread`, will read from the file and display the state of the art gallery according to a given query over the log. Both programs will use an authentication token, supplied as a command-line argument, to authenticate each other; the security model is described in more detail below.

Programs

Players design the log format and implement both `logappend` and `logread` to use it. These programs can be written in any programming language(s) of the contestants' choice as long as they can be built and executed on the Linux VM provided by the organizers. Each program's description is linked below.

- The `logappend` program appends data to a log
- The `logread` program reads and queries data from the log

`logread` contains a number of features that are optional. If you do not implement an optional feature, be sure to **print `unimplemented` to standard output**.

Examples

Look at the page of examples for examples of using the `logappend` and `logread` tools together.

Security Model

The system as a whole must guarantee the privacy and integrity of the log in the presence of an adversary that does not know the authentication token. This token is used by both the `logappend` and `logread` tools, specified on the command line. *Without knowledge of the token* an attacker should *not* be able to:

- Query the logs via `logread` or otherwise learn facts about the names of guests, employees, room numbers, or times by inspecting the log itself
- Modify the log via `logappend`.

- Fool `logread` or `logappend` into accepting a bogus file. In particular, modifications made to the log by means other than correct use of `logappend` should be detected by (subsequent calls to) `logread` or `logappend` when the correct token is supplied

Oracle

An oracle reference implementation is provided to demonstrate the expected output of a series of commands run on `logread` and `logappend`. Contestants may run the reference implementation by going to the team participation page on the website and clicking on "Oracle Submissions". Here is an example of the expected input for the oracle:

```
[
    "logappend -T 1 -K secret -A -E Fred log1",
    "logappend -T 2 -K secret -A -G Jill log1",
    "logappend -T 3 -K secret -A -E Fred -R 1 log1",
    "logappend -T 4 -K secret -A -G Jill -R 1 log1",
    "logread -K secret -S log1"
]
```

Details

- Names only contain alphabetical characters (a-z, A-Z).
- Leading zeros in room IDs should be dropped, such that 003, 03, and 3 are all equivalent room IDs.
- If a single line in a batch file is invalid, print the appropriate error message for that line and continue processing the rest of the batch file.
- Log files may contain underscores.
- Integer inputs, like timestamps, are limited to 32 bits.
- Names are case sensitive. Employees and guests cannot have the same name.

Build-it Round Submission

Each build-it team should initialize a git repository on either github or bitbucket and share it with the `bibifi` user on either of those services. Create a directory named `build` in the top-level directory of this repository and commit your code into that folder. Your submission will be scored after every push to the repository. (Beware making your repository public, or other contestants might be able to see it!)

To score a submission, an automated system will first invoke `make` in the `build` directory of your submission. The only requirement on `make` is that it must function without internet connectivity, and that it must return within ten minutes.

Once make finishes, `logread` and `logappend` should be executable files within the `build` directory. An automated system will invoke them with a variety of options and measure their responses.