

CSE 365: Introduction to Information Assurance

[SCHEDULE](#) [SYLLABUS & POLICY](#) [ASSIGNMENTS](#)

Assignment 2

Assignment 2 is *due 2/3/21* on or before 11:59:59pm MST.

Part 1 — Bandit v2 (10 points)

Continuing on [Assignment 1 Part 2 Bandit v1](#): The goal is to solve levels 5–10 (in other words reach level 11) on the [overthewire.org](https://overthewire.org/bandit-challenges/) Bandit challenges.

Please [consult Assignment 1 Part 2 Bandit v1](#) if you're unfamiliar with how to create a wechall user, link it to overthewire, etc.

Submission Instructions

[Submit on GradeScope](#) the plain text file `README` to gradescope. **Note:** you must include the following line in your README (replace the `INSERT_WECHALL_NAME_HERE` with your wechall username), or else the autograder won't be able to give you a grade and you will get a 0:

```
wechall name: INSERT_WECHALL_NAME_HERE
```

Part 2 — Secure this house (90 points)

Your goal is to write, in any language, a program which implements the given security policy. The security policy will be based on our in-class discussion of the security policy for a house.

The name of your house simulator will be called `secure_house`.

Policy

Only users with an `authorized key` can enter the house. To enter the house, the user must first:

1. Insert their key in the lock
2. Turn the key in the lock
3. Enter the house

Note that testing if a key is valid is done only when the key is turned.

For turning the key and entering the house, it must be the same user that puts the key in the lock, turns the key, and enters the house.

A house can be rekeyed (this means that the old keys are no longer useable) with new keys only by the owner, and only if the owner is inside the house.

Firefighters can enter with the secret key (literal string) `FIREFIGHTER_SECRET_KEY`.

There is only one lock and one door. The lock will always be accessed in the following way:

1. insert key
2. turn the key
3. enter the house

Other commands can be issued in between insert, turn, and enter.

For example, the following situations will never happen:

- insert, enter
- insert, turn, insert
- turn, enter
- insert, turn, turn, enter

Interface

You must implement the following command-line interface for your server:

```
./secure_house <owner_name> <key_1> <key_2> ... <key_n>
```

where `<owner_name>` is the name of the owner, and `<key_1>` through `<key_n>` are all authorized keys for the house.

All inputs to the program (keys and names) will be `[a-zA-Z0-9_\-]` (alphanumeric, underscore, and dash). All matching is case-sensitive.

The input to your program (on [standard input](#)) will be a series of events separated by a newline. Your program must track these events and respond appropriately, while enforcing the security policy.

Every input will end in a [newline](#), and every response must end in a newline.

Your program must continue to process input until there is no input left. How you do this will vary by programming language, try Googling for “End of file” (or EOF) and your programming language.

You can simulate EOF on the command line by pressing CTRL-D (more info [on this stackoverflow](#)

post).

```
INSERT KEY <user_name> <key>
```

<user_name> inserts key <key> into the door. Response should be: **KEY <key> INSERTED BY <user_name>**

```
TURN KEY <user_name>
```

<user_name> turns the key in the door. Possible responses are: **SUCCESS <user_name> TURNS KEY <key>** or **FAILURE <user_name> UNABLE TO TURN KEY <key>**

```
ENTER HOUSE <user_name>
```

<user_name> enters the **house**. Possible responses are: **ACCESS DENIED** or **ACCESS ALLOWED**.

```
WHO'S INSIDE?
```

Who is currently inside the house? Response must be a comma-separated list of user names, ordered by access time (earlier access first): **<user_name_1>, <user_name_2>, <user_name_3>...** or **NOBODY HOME** if there are no users in the house.

```
CHANGE LOCKS <user_name> <key_1> <key_2> ... <key_n>
```

<user_name> wishes to rekey the house with new given keys **<key_1>, <key_2>, ..., <key_n>**. Possible responses are: **ACCESS DENIED** or **OK**

```
LEAVE HOUSE <user_name>
```

<user_name> leaves the **house**. Possible responses are: **OK** or **<user_name> NOT HERE**

If any events are received that are not according to this specification, the response must be: **ERROR**.

Example

Running the program as follows:

```
./secure_house selina foobar
```

Given the input:

```
INSERT KEY adam key
TURN KEY adam
ENTER HOUSE adam
INSERT KEY pat foobar
TURN KEY pat
ENTER HOUSE pat
WHO'S INSIDE?
```

The program must produce the following output:

```
KEY key INSERTED BY adam
FAILURE adam UNABLE TO TURN KEY key
ACCESS DENIED
KEY foobar INSERTED BY pat
SUCCESS pat TURNS KEY foobar
ACCESS ALLOWED
pat
```

Implementation

Your program must work on [Ubuntu 18.04 64-bit](#) with the default packages installed.

In addition to the default packages, the following packages for languages are also installed:

- C (`gcc`)
- C++ (`g++`)
- Python 2 (`python`)
- Python 3 (`python3`)
- Java (`default-jre` and `default-jdk`)
- Rust (`rustc`)
- Ruby (`ruby-full`)
- Node.js (`nodejs` and `npm`)
- Mono (Custom install)
- Go (1.15.6, custom install)

If there's a package that you need, please [ask on the course piazza](#) and I'll have it installed for everyone. Java is already installed.

If you need to set up a virtual machine for your development: [VirtualBox](#) is a free and open-source VM system.

We've created a [test script](#) called `test.sh` to help you test your program before submitting.

1. Download [test.sh](#) to the directory where your code lives (including `README` and

`Makefile`).

2. Ensure that `test.sh` is executable: `chmod +x test.sh`
3. Run: `./test.sh`

There is also a [test_debug.sh](#) that gives you the output of your program. This can help you with debugging when the program appears to work from the command line, but not in the `test.sh` script (it's happened before).

Your program must be able to accept arbitrarily large input (and this will be tested by the autograder).

Submission Instructions

[Submit on GradeScope](#) your source code, along with a `Makefile` and `README`. The `Makefile` must create your executable, called `secure_house`, when the command `make` is ran. Your `README` file must be plain text and should contain your name, ASU ID, and a description of how your program works.

CSE 365: Introduction to Information Assurance

Adam Doupé and Tiffany Bao

Website Design Acknowledgement: Carolina Zarate and Zilin Jiang