Keeping a Faulty Apartment Secure:
Safety and Cost Analysis of a Realistic Apartment Structure

EE595 (A) Secure Computer Architecture, Project 1

20218116 (name kept anonymous per request)

I. Introduction

In this section a brief summary of the goal of this project (the way I understand it) is provided. There is an F-story apartment with N households on each floor, resulting in an apartment consisting of F×N households in total. This apartment is faulty in that an opening (and thus a closing, or a slamming) of a door at room #f0n (f-floor, n^{th} household) results in the doors of its upper and lower households (rooms #(f+1)0n and #(f-1)0n) becoming loose. A door breaks as the screws become loose upon the T^{th} slamming it is affected by.

A security guard is hired for this specific apartment. The guard may target a certain household or a pool of certain households to "reset" their doors to maintain security. This comes at a cost, which is detailed further in Section II.

II. Assumptions

There are a few assumptions that must be established prior to conducting any experiments. Below are some of these assumptions that have been made about the apartment's situation.

1. *The households are distributed to people at random and all houses must be entered by their residents only.*

Malicious users, also known as 'robbers', can manipulate their access patterns to certain households (e.g., entering rooms 105 and 305 to break into 205) but because the security guard assigns rooms at random and entering a door requires one of its corresponding residents, it is unlikely for a malicious user to exploit the apartment's faultiness to gain access to a specific room. Given this assumption, a malicious user would attempt to move to a house that is capable of attacking his/her target house, which leads to the second assumption.

2. *The residents in the apartment are not allowed to move.*

The apartment restricts residents from moving for two reasons. First, a malicious user could move in and out repeatedly until a room that can attack the target household is assigned. Moving is forbidden to get rid of this idea. Although the idea of restricting residents from moving is unlikely in real world scenarios, the apartment committee found the moving fee inconceivably high and has banned all moving activities from taking place.

3. *The security keeps an access log to track of how many times a group's doors have been affected by slams.*

The access log can be kept in four manners, depending on how the group is determined. When a certain level of threshold is reached, the doors of all rooms inside the group are 'reset' by the security, being able to tolerate another T slams by their vertical neighbors.

A. PER_APARTMENT refers to the apartment being kept track of as a whole. This is the simplest log to keep track of, but is also expected to be the least secure in that it fails to provide any details on which room has been accessed – just a simple number showing how many times the residents have been affected by slams. While most slams cause the log's count to increase by 2, the top and bottom floors' doors being slammed would increment the log's count by 1.

B. PER_FLOOR refers to the log being kept at a floor-level. A slam at a floor f would result in the logs of floors (f-1) and (f+1) being affected, lest f is the bottom of top floor, in which case it would only affect one. Both of the logs for floors (f-1) and (f+1) will be incremented.

C. PER_LINE is the line[1] version of PER_FLOOR. It is cheaper to keep track of, given that a realistic

[1] *line:* the vertical group of households in the apartment who are above/below each other. For example, rooms 105, 205, 305, …, 2005 are in the same line.

apartment would have more floors than lines. A door slam at line l would result in the log's count being increased by 1 or 2, depending on which floor the slam occurred at.

D. PER_DOOR is the most expensive type of log which keeps track of every room/door. This method is designed to be 100% secure, if the log's threshold is equal to the door's threshold.

4. *The access patterns of the residents of the apartment are unknown.*

In this project we test four access patterns, but in real world scenarios there are an infinite number of accesses, depending on the type of residents that dwell within the apartment.

A. UNIFORM is the simplest case where all accesses are uniformly distributed across the whole apartment.

B. TWO_QUADRANTS is where if the apartment is divided into four quadrants (the x-axis being parallel to the ground and y-axis denoting the distance to the ground), only two quadrants that are diagonally located are accessed. The accesses to each of the rooms in the active quadrants are uniformly distributed. This is essentially identical to having two separate identical (or similarly sized) apartments and the accesses being completely uniform. In this project the second and fourth quadrants have been selected (where either the floor number or the line number is over half of its maximum value) but the experiment would be the same for the first and third quadrants as well, by symmetry.

C. FEW_HOT is where the top 10% of the residents take up 90% of the accesses. Upon each access simulation, the chance to roll on a hot access is 90%, then uniformly chooses one household from the top 10% hot group. Upon a cold access roll, the household to be accessed is chosen uniformly from the 90% cold group. Effectively, this would leave a hot resident be more likely to be accessed than a cold resident by 81 times.

D. PSEUDOGAUSSIAN is where the floor number and the line number are both decided independently by a Gaussian distribution. The mean is the center of the floor/line, and the standard deviation of the distribution is chosen in advance so that 99% of the 'rolls' of the accesses for each the floor and the line would be in a feasible range (non-negative and under the maximum floor/line). When the distribution does hit an invalid number, the access is directed towards the nearest value, which is the bottom or top (start or end), depending on the rolled number.

5. *The reset cost does not hinder the residents' behavior. Instead, such overhead is considered a price*.

Such price overhead incurred by resets is considered in analysis when determining the cheapest method.

III. Methodology

A slam-level simulator was built in Python to evaluate all access patterns and logging methods. While sweeping for access patterns and logging methods, a third parameter called the tradeoff variable was also swept.

The tradeoff variable, varying from 0.1 to 2.0, denotes how much cheaper the reset overhead will be, in comparison to the PER_DOOR method. While the PER_DOOR method is extremely tedious to keep track of, it is the most secure in that the doors cannot be broken – the log of the door about to be broken will trigger the security to reset it. That is, because the PER_DOOR method is slamproof, the other methods should be given different levels of tradeoff (when the tradeoff variable is smaller than 1.0) between security and reset cost to provide a reason to use them over PER_DOOR. When the tradeoff variable is larger than 1.0, it lowers security levels but reduces the reset overhead.

The apartment is configured to resemble that of a realistic one, with 20 floors and 10 lines each, resulting in a total of 200 households. The threshold for the doors has been set to 11 so that they are safe up to 10 slam effects. The log reset timer is the threshold of the security to reset the log group's doors. It is set to 10 in PER_DOOR to trigger reset at any door's 10^{th} slam effect. In other logging methods, they are increased linearly depending on the group size increase. Then the tradeoff variable takes effect and is multiplied to the log reset timer, once for PER_FLOOR and PER_LINE, and twice for PER_APARTMENT.

Each configuration of the three sweep parameters was run 10 times, with 100 thousand slams each.

The Python code consists of 363 lines of code across two modules and one main file. The source is available

on GitHub here, *but be warned that the GitHub account name reveals the author's name, which was asked to be kept anonymous.*

IV. Results

In this section, only some results out of the 320 different configurations are shown. The full results of all other tested configurations can be viewed along with the source code at the git provided above, inside the files located in `out/parsed/100k_slams__10_runs/parsed/$tradeoff_variable/` directory.

We denote the *cheapest secure TV* as the largest tradeoff variable that leads to 100% security. We denote the *average survived slams* as the average number of slams of the simulation when the tradeoff variable was the next smallest variable to the cheapest secure TV. We denote the *average survival resets* as the average number of resets that occurred in the simulations under the cheapest secure TV configuration.

A. PER_DOOR analysis (200 logs)

Access type	Cheapest secure TV	Average survived slams (tradeoff variable)	Average survival resets (normalized value)
UNIFORM	-	-	9918.2 (9918.2)
TWO_QUADRANTS	-	-	9916.4 (9916.4)
FEW_HOT	-	-	9960.4 (9960.4)
PSEUDOGAUSSIAN	-	-	9918.7 (9918.7)

Table 1. Simulation results for PER_DOOR logging method

Table 1 displays the simulation results for the PER_DOOR logging method. Unlike the other three logging methods, the tradeoff variable does not affect the PER_DOOR configuration. The average survival resets values are from the tradeoff variable 1.0, but the difference within different configurations are statistically insignificant. PER_DOOR achieves perfect security as expected for all configurations. From 100,000 slams, the average number of resets spanned from 9915.1 to 9961.7. This is as expected because some doors can be left loose but not completely broken, and there should be an average of 1 reset in every 10 slams when the slams go on indefinitely, and the results showed that all average survival resets were a little smaller than 10,000.

B. PER_APARTMENT analysis (1 log)

Access type	Cheapest secure TV	Average survived slams (tradeoff variable)	Average survival resets (normalized value)
UNIFORM	0.3	40,324.3 (TV = 0.4)	555 (111,000)
TWO_QUADRANTS	0.2	20,297.4 (TV = 0.3)	1,250 (250,000)
FEW_HOT	0.1	4,749 (TV = 0.2)	5,000 (1,000,000)
PSEUDOGAUSSIAN	0.2	18,330.1 (TV = 0.3)	1,250 (250,000)

Table 2. Simulation results for PER_APARTMENT logging method

As can be seen from Table 2, the FEW_HOT simulation configuration is the most expensive one to deal with, because the gap between the cold and hot residents' access frequency is too large to cover, leading to very early infiltrations compared to other configurations' average survived slams. The only reason that the cheapest secure TV exists for FEW_HOT is because the value of 0.1 means that the apartment has employed 100 times higher reset rate compared to the PER_DOOR logging method.

C. PER_FLOOR analysis (20 logs)

Access type	Cheapest secure TV	Average survived slams (tradeoff variable)	Average survival resets (normalized value)
UNIFORM	0.1	16,950 (TV = 0.2)	9,991.7 (99,917)
TWO_QUADRANTS	0.1	7,521.8 (TV = 0.2)	9,991.6 (99,916)
FEW_HOT	0.1	145.9 (TV = 0.2)	9,992.8 (99,928)
PSEUDOGAUSSIAN	0.1	15,945.6 (TV = 0.2)	9,991.8 (99,918)

Table 3. Simulation results for PER_FLOOR logging method

The simulation results for PER_FLOOR logging method are shown in Table 3. It's interesting for

PER_FLOOR that the cheapest secure TV is fixed to 0.1 and the average survival resets are near 10,000. When the average survival resets are normalized so that it reflects the cost of updating multiple logs at once, it becomes clear that PER_FLOOR logging method is essentially 10 times (number of lines in a floor) more expensive than PER_DOOR logging method when it comes to reset overhead. This shows that due to all of the floors being reset at once when only a few doors need fixing is reflected directly in the reset overhead.

A different type of PER_FLOOR logging method, which stems from the idea that the odd floor door slams affect doors located on the even floors and that the even floor door slams affect doors on the odd floors, keeping two logs (one for even floors and one for odd floors) could be a reasonable logging method. This method was not simulated in this work.

D. PER_LINE analysis (10 logs)

Access type	Cheapest secure TV	Average survived slams (tradeoff variable)	Average survival resets (normalized value)
UNIFORM	0.1	18,898.3 (TV = 0.2)	4995.1 (99,902)
TWO_QUADRANTS	N/A (10% @ TV = 0.1)	22,024 (TV = 0.1)	N/A
FEW_HOT	N/A (0% @ TV = 0.1)	148.2 (TV = 0.1)	N/A
PSEUDOGAUSSIAN	0.1	6,928.7 (TV = 0.2)	4,995 (99,900)

Table 4. Simulation results for PER_LINE logging method

PER_LINE configuration's results in Table 4 shows how dangerously insecure the logging method is. It is the only logging method that has failed to achieve any security even at very low TV of 0.1.

The results clearly show that because the overhead of keeping multiple logs is not large enough, PER_DOOR is the cheapest (and 100% secure) logging method for this faulty apartment under the previously stated assumptions. However, if keeping multiple logs was infeasible or too costly, PER_FLOOR is an alternative that would keep the apartment secure enough (except under pathological access patterns) with 10 times (number of lines in a floor) a smaller number of logs at a higher resetting cost.

Because the optimal logging method (depending on the cost of multiple logs and reset costs) is either PER_DOOR or PER_FLOOR regardless of the access types, an adaptive logging method that would adjust to the cheapest method as the slams occurred is unnecessary. Although not in line with our assumptions, *if* there were malicious users, they would be able to manipulate the learning algorithm and attack the apartment security system.

V. Limitations

Our slam-level simulator fails to analyze deeper details of the slams, and assumes that there is no time delay but only cost overhead in resetting the doors, which is far from a realistic scenario. The assumptions that essentially make it impossible for malicious users to break into an apartment household is also unrealistic, and defending against the faultiness of the apartment is much easier when there are no malicious parties involved. This work also fails to address a good enough configuration that would beat the obvious solution of PER_DOOR logging method, in terms of security and maintenance cost.

If there were signs that a FEW_HOT access pattern was taking place, the security could apply hybrid logging methods to keep track of the hot access in a PER_DOOR logging method and the rest in PER_APARTMENT logging method. However, due to lack of time, this work does not employ such adaptive or learning mechanisms.

Another seemingly promising configuration this work fails to accommodate is the EVEN_ODD_FLOOR configuration. Because an even floor's slam only affects the odd floors' doors, and vice versa, this method may serve as a cheaper version of a PER_FLOOR logging method that is just as secure. However, this configuration was not modeled in our simulator due to lack of time.

Nonetheless, to the best of our knowledge, this work is the first to address the faultiness of an apartment, where a slam of a door affects its neighboring floors' doors. in depth and conduct a slam-level analysis on a realistic apartment configuration.