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 five 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 random 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.

E. SEQUENTIAL is when the access is sequential, going from (0, 0) to (0, N – 1) to (1, 0) to (1, N – 1), all the way to (F – 1, N – 1) which wraps around to (0, 0), where (f, l) denotes the f^{th} floor l^{th} line being accessed.

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

Only the 'price overhead', or spatial overhead, is considered. That is, there is virtually no temporal overhead coming from resetting the doors. A slam or reset does not affect the next slam, should there be one.

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 0.04 to 0.4, denotes how much more secure the system will become, at the expense of a higher reset overhead. 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. Because the PER_DOOR method is slamproof and others are not, they need to trade off cost for higher security. When the tradeoff variable is smaller than 1.0, security is enhanced at the cost of more frequent resets, and if the tradeoff variable is larger than 1.0, security is handled more loosely while incurring less reset overhead. At first, tradeoff variable was swept from 0.1 to 2.0, but it was later determined that the notion of 'batching' several doors into one log made it infeasible for tradeoff variable to go beyond 0.4, and was retested with finer granular tradeoff variables, from 0.04 to 0.4.

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 10th 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 for PER_FLOOR, PER_LINE, and PER_APARTMENT.

Each configuration of the three sweep parameters was run 10 times, with 100,000 slams each.

The Python code consists of 432 lines of code across two modules and one main script. 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. Evaluation

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 the `out/parsed/` 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.

1. *Simulation results*

A. PER_DOOR analysis (200 logs)

Access type	Cheapest secure TV	Average survived slams (tradeoff variable)	Average survival resets (normalized value)
UNIFORM	-	-	18,412.8 (18,412.8)
TWO_QUADRANTS	-	-	18,458 (18,458.0)
FEW_HOT	-	-	18,193.7 (18,193.7)
PSEUDOGAUSSIAN	-	-	19,618.1 (19,618.1)
SEQUENTIAL	-	-	18,500 (18,500.0)

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 0.4, 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 17,678.6 to 19,618.8. This is as expected because some doors can be left loose but not completely broken, and there should be an average of a little under 2 resets in every 10 slams when the slams go on indefinitely, and the results showed that all average survival resets were a little smaller than 20,000.

Compared to other runs, PSEUDOGAUSSIAN showed a little higher number of average survival resets. This is due to the fact that in PSEUDOGAUSSIAN accesses are centered and make it highly unlikely for accesses to happen at edges and corners of the apartment, where the number of affected households affected by a single slam is small. However, the number itself does not vary too far off from the other configurations' numbers.

B. PER_APARTMENT analysis (1 log)

Access type	Cheapest secure TV	Average survived slams (tradeoff variable)	Average survival resets (normalized value)
UNIFORM	0.08	(TV = 0.12)	1,155.6 (231,120)
TWO_QUADRANTS	0.04	55,600 (TV = 0.08)	2,311.5 (462,300)
FEW_HOT	N/A (30% @ 0.04)	33,882.14 (TV = 0.04)	N/A
PSEUDOGAUSSIAN	N/A (90% @ 0.04)	37,800 (TV = 0.04)	N/A
SEQUENTIAL	0.40	N/A (100% @ 0.40)	256 (51,200)

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, achieving only a 30% security (3 successful runs out of 10) even with a very low tradeoff variable of 0.04. This

is 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. In general, PER_APARTMENT fails to provide benefit over the PER_DOOR configuration. In FEW_HOT and PSEUDOGAUSSIAN access types, PER_APARTMENT fails to achieve security. In TWO_QUADRANTS, PER_APARTMENT only achieves 100% security at the lowest tradeoff variable of 0.04.

On the other hand, in SEQUENTIAL, all swept tradeoff variables were 100% secure due to the fact that because the accesses are sequential and therefore 100% spread out, resetting the apartment because of a door reaching the threshold meant that other apartment doors were very near (1 away from) reaching the threshold as well. In fact, we can expect PER_APARTMENT to reach 100% security for SEQUENTIAL until TV is 1.0. The results from SEQUENTIAL varies greatly from UNIFORM because while they are both 'spread out' equally, there is no randomness in SEQUENTIAL and the random-access behavior of UNIFORM hinders the PER_APARTMENT performance greatly.

C. PER_FLOOR analysis (20 logs)

Access type	Cheapest secure TV	Average survived slams (tradeoff variable)	Average survival resets (normalized value)
UNIFORM	0.16	70,003 (TV = 0.20)	11,556.2 (115,562)
TWO_QUADRANTS	0.08	57,060 (TV = 0.12)	23,118.4 (231,184)
FEW_HOT	0.08	186 (TV = 0.12)	22,950.9 (229,509)
PSEUDOGAUSSIAN	0.12	29,439.67 (TV = 0.16)	16,406.9 (164,069)
SEQUENTIAL	0.40	N/A (100% @ 0.40)	5,124 (51,240)

Table 3. Simulation results for PER_FLOOR logging method

The simulation results for PER_FLOOR logging method are shown in Table 3. As with PER_APARTMENT, the FEW_HOT access type is the hardest access pattern to deal with. SEQUENTIAL is also 100% secure in PER_FLOOR because there is no randomness involved, and would probably work fine up to the tradeoff variable being 1.0, effectively reducing the normalized reset overhead to somewhere around 20,000.

Although the size of a group was reduced from 200 to 10 compared to the PER_APARTMENT logging method, the normalized reset cost is generally much higher than that of PER_DOOR configuration's, which is due to the fact that in logging groups and not individual rooms, rooms that are not close to reaching the threshold are also reset.

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	0.20	28741.67 (TV = 0.24)	4619.4 (92388)
TWO_QUADRANTS	0.08	80847.5 (TV = 0.12)	11562.8 (231256)
FEW_HOT	0.04	15749.8 (TV = 0.08)	23202.6 (464052)
PSEUDOGAUSSIAN	0.12	26864.8 (TV = 0.16)	8203.5 (164070)
SEQUENTIAL	0.40	N/A (100% @ 0.40)	2560 (51200)

Table 4. Simulation results for PER_LINE logging method

PER_LINE configuration's results in Table 4 are somewhat similar to those of PER_FLOOR's, except a little more robust (and thus cheaper) in UNIFORM and requiring more reset overhead in FEW_HOT.

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 is infeasible or too costly, PER_FLOOR and PER_LINE are alternatives that would serve the purpose of keeping the apartment secure enough (except under pathological access patterns) with 10 or 20 times (number of lines in a floor / floors in a line) a smaller number of logs, albeit

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/PER_LINE 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.

2. *Sensitivity*

In this section, we try adding even more faultiness to the apartment by assuming that a slam could affect more doors.

In TRIPLE faulty scale, a door slam not only affects the room right on top of the slammed door's household, but also the two rooms next to them. This would make the slam affect at most 6 rooms. For door slam at room #f0n, rooms #(f+1)0n, #(f+1)0(n-1), #(f+1)0(n+1), #(f-1)0n, #(f-1)0(n-1), and #(f-1)0(n+1) are affected. In ALL faulty scale, a door slam at floor f would affect all doors to become looser at floors (f-1) and (f+1). In our configuration, a single door slam could affect 10 or 20 rooms, depending on which floor the slam was at.

In Table 5 we display the cheapest secure TV values for different faultiness scales. Because the average survival resets values are (as can be seen from Section IV's results) approximately reversely proportional to the cheapest secure TVs, we omit the average survived slams and average survival resets. Furthermore, we show how the cheapest secure TV values have changes from SINGLE (baseline results in Section IV) to TRIPLE and ALL. Also, because PER_DOOR is always 100% secure and not affected by different tradeoff variable values, we only show PER_APARTMENT, PER_FLOOR, and PER_LINE logging methods' results.

Logging method	Access type	Cheapest secure TV changes		
		SINGLE	TRIPLE	ALL
PER_APARTMENT	UNIFORM	0.08	0.12	0.16
	TWO_QUADRANTS	0.04	0.04	0.16
	FEW_HOT	N/A	N/A	0.08
	PSEUDOGAUSSIAN	N/A	0.04	0.12
	SEQUENTIAL	0.40	0.40	0.08
PER_FLOOR	UNIFORM	0.16	0.16	0.40
	TWO_QUADRANTS	0.08	0.12	0.40
	FEW_HOT	0.08	0.08	0.40
	PSEUDOGAUSSIAN	0.12	0.16	0.40
	SEQUENTIAL	0.40	0.40	0.40
PER_LINE	UNIFORM	0.20	0.16	0.12
	TWO_QUADRANTS	0.08	0.08	0.12
	FEW_HOT	0.04	0.04	0.08
	PSEUDOGAUSSIAN	0.12	0.08	0.12
	SEQUENTIAL	0.40	0.40	0.08

Table 5. Effect of more severe faultiness (SINGLE < TRIPLE < ALL)

We can see that for PER_APARTMENT, going from SINGLE to TRIPLE to ALL actually helps with security stability, raising the cheapest secure TV values. While counterintuitive, this can be attributed to the fact that as more rooms are affected by a single door slam, the affected doors become more spread out, reducing the deviation of the access locations and lessening the negative effect randomness has on stability. In case of SEQUENTIAL, it drops to 0.08 in ALL because in ALL faultiness scale with SEQUENTIAL access type, A single door can be affected 20 times in 30 slams (10 effects from 10 slams in lower floor, 0 effects from 10 slams in its floor, then 10 effects from 10 slams in upper floor), which is not enough slams to trigger the resets for the whole apartment in PER_APARTMENT with a high enough tradeoff variable.

For PER_FLOOR, for similar lower randomness reasons, its stability increases and manages to reach all cheapest secure TV to become 0.4 (highest TV simulated), making it the best choice if the apartment's faultiness

is equal to TRIPLE or ALL, over other non-PER_DOOR methods.

In PER_LINE, the UNIFORM access's stability fell as the faultiness grew, which is as expected. The stability of SEQUENTIAL fell dramatically in ALL because of the same reason as PER_APARTMENT. For other access types, it is hard to conclude from the results of Table 5.

Thus, in this subsection we conclude that as the faultiness of the apartment becomes more severe and a single slam affects more doors in the neighboring floors, the stability for different logging methods (especially PER_FLOOR) grow, narrowing their reset overhead with the most secure PER_DOOR logging method. One can conduct similar experiments under their faultiness level and compare the results of PER_DOOR and PER_FLOOR, compute their logging and reset overhead costs, choosing the most secure and least expensive method.

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 an analysis using a slam-level simulator on a realistic apartment configuration