Microarchitectural Attacks — Project 1

Due: 4:59pm, Wednesday, June 14, 2023

In this project you will perform the L1-D Prime+Probe attack on the final round of AES-128 encryption. The project consists of three tasks. The first task is to implement and test a generic Prime+Probe attack. The second task is to use your attack against a T-table implementation of AES. The third task is to analyze the attack results and recover the key. These tasks are further detailed below.

Work is in groups of two students. See submission instructions at the end of this document.

Task 1: Implement Prime+Probe (40%)

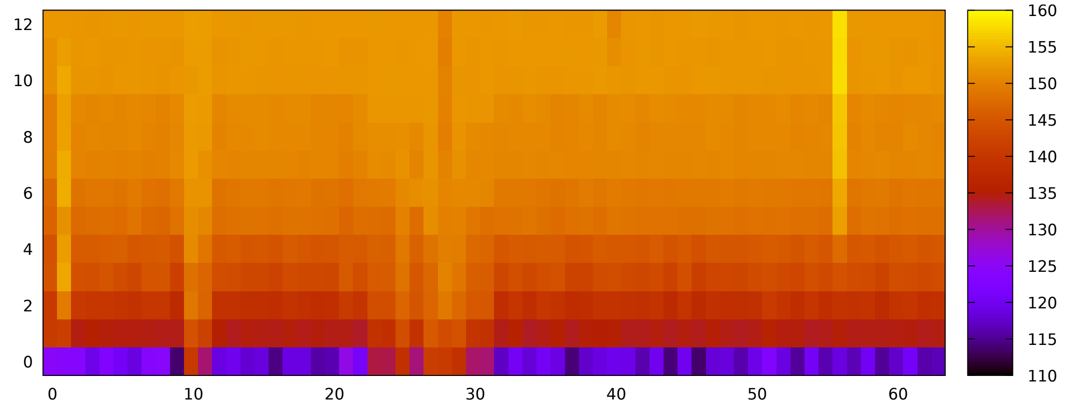
The code for Prime+Probe consists of three main functions. The function `void ppinit()` initializes all the data structures required for implementing the attack, including the eviction sets. It further ramps up the CPU speed by looping for at least a billion cycles. A possible implementation of the ramp up step is:

```
int dummy;
uint64_t start = rdtscp(&dummy);
while (rdtscp(&dummy) - start < 1000000000) ;
```

The other two functions are `void prime()`, which primes all cache sets in the L1 cache, and `void probe(uint64_t result[64])`, which probes the L1 cache and returns the probe time of cache set s in `result[s]`. Note that your code must handle the prefetcher, e.g. by randomizing the order of scanning the sets.

To test the implementation, create a fake victim that takes two arguments: a cache set number $0 \leq s < 64$, and a number of lines $0 \leq l \leq 12$. Then, use this victim to calculate the average probe time in each of the cache sets $0 \leq s < 64$ when accessing $0 \leq l \leq 12$ lines in that set. To stabilize the results, you want to repeatedly choose random s and l, execute a test, and record the results. You also need to remove outliers. Otherwise, the numbers will take a really long time to converge.

Please submit the code for a test program that outputs the average probe times, as well as a heatmap figure that shows the results on your machine. The heatmap shows the cache sets in the horizontal axis, the number of accessed elements in the vertical, and uses a shade that corresponds to the average probe time of the set when the victim accesses the number of lines in the set. See an example of such a heatmap below. Do not submit object files, binaries, or any other generated data.



Task 2: Attacking AES (30%)

Download the implementation of AES from `http://www.qwerty.co.il/AES.tgz`. Use it to write a program that takes two command line arguments: a 32-digit hexadecimal number, that represents an AES key, and a number of encryptions n. The program generates n random plaintexts, and performs Prime+Probe when encrypting each. The output of the program is n lines of text. Each line has the following format:

```
plaintext ciphertext cs0 cs1 ...cs63
```

Where `plaintext` is the random plaintext, `ciphertext` is the result of encrypting the plaintext with the key, and cs$i$ is the probe time for cache set i after executing the encryption. You can see an example of such a file in `https://www.qwerty.co.il/data.txt`. (Warning, the size of the file is approximately 30 MB.)

For submission, please include all source code for your program, and the result of encryption 1000 plaintexts with the key `00112233445566778899aabbccddeeff`.

Task 3: Final Round Correlation Attack (30%)

The aim of the last task is to run a correlation attack to recover the final round key of the encryption. For that, write a program that takes two arguments. The first is a name of a file that contains the output from Task 2, and the second is the ciphertext byte to target (between 0 and 15).

The program uses the correlation attack described in the lecture, and outputs the 10 guesses with the highest absolute correlation.

Note that because you do not know the cache location on the relevant T-table, you need to guess both the relevant key byte and the position of the relevant table. Thus, the total number of guesses is 256×64.

For each such guess, given the byte of the plaintext, you can compute the probability that each cache set is accessed during the encryption. The correlation attack calculates the correlation between the probe time of a cache set and the probability that the set is accessed. Summing this over all cache sets gives the correlation between the guess and the probe results. Given enough probe results, we expect that the guess with the highest absolute correlation is the correct guess. Don't forget to ignore outliers.

Please submit the code you developed to find the key, and the final round key used to produce the example output above. Note that the code can be in C, Python, or any other language that the marker allows.

Submission

Please submit a tar archive that has three folders, named `task1`, `task2`, and `task3`. Each folder has a `README` file that explains how to use the code and where to find the requested results. Please make sure to include your names and IDs.