Side-Channel Security 2025
Exercise 1

In this exercise, you will get hands-on experience with a small selection of microarchitectural attacks.

1. This exercise is worth **up to 15 points**.
2. For a positive grade, you need **at least 4 points** on both exercise 1 (this sheet) and exercise 2.
3. Only task 1 is mandatory: if you do not submit it, you will not get a grade.
4. Put all source code, tools, and result files for this submission into folder `ex1` in your repository.
5. Tag your submission with the tag `ex1-1` before the first deadline so we know you will participate. Tagging the second submission (`ex1-2`) is only necessary if you don't want the last commit on main to be used.
6. For all tasks, you can find more information at `https://www.isec.tugraz.at/teaching/materials/scs/exercises/ex1/`

1 Introduction to Cache Attacks – 0.5 Point

Mandatory
Deadline: Thursday, March 20 2025, 8:00am

Produce a Flush+Reload cache hit/miss histogram on at least one machine per team member, and **submit a plot image** into your repository and add a .txt file that specifies the CPU models. Use your histogram to decide on a *good* threshold for each machine. **Both team members** must be able to reproduce their histogram in the exercise interview and explain their choice of threshold.
You may use the histogram demo in the upstream repository, **no programming is necessary** for this task!

2 AES T-Table Attack – 4 Points

Optional
Deadline: Thursday, May 8 2025, 8:00am

We've come across an AES library that doesn't use `AES-NI` instructions but instead falls back to the insecure T-tables implementation of AES found in OpenSSLv3.4. This implementation is susceptible to a simple first-round cache attack, which can leak the upper 4 bits of all 16 key bytes.

The underlying mechanism of this attack is that in the first round, the four T-Tables Te_{0-3}, consisting of 256 4-byte entries each, are accessed depending directly on the key. More specifically, in the first round the tables are accessed according to the XOR of the plaintext and key bytes:
$Te_0 : P_{\{0,4,8,12\}} \oplus K_{\{0,4,8,12\}}$
$Te_1 : P_{\{1,5,9,13\}} \oplus K_{\{1,5,9,13\}}$
$Te_2 : P_{\{2,6,10,14\}} \oplus K_{\{2,6,10,14\}}$
$Te_3 : P_{\{3,7,11,15\}} \oplus K_{\{3,7,11,15\}}$

Accesses also happen in later rounds, but we know one thing: when we fix a plaintext byte, the XOR for this byte is *always* accessed, regardless of everything else. Since we are looking at cache lines, we lose some information: entries are 4 bytes wide, while cache lines contain 64 bytes. This is equivalent to the upper 4 bit of each XOR operation. With this resolution in mind, we can calculate the probability that a given cache line in the T-tables was accessed after an entire AES encryption (given random plaintexts): $1 - \frac{15}{16}^{40} = 92.43\%$. Since we know that one cache line for each plaintext byte we fix will always be accessed (100%), we can measure a lot of encryptions until we see this difference for a given cache line (e.g., 92 accesses out of 100 when $P_n \oplus K_n$ does not land in the cache line we measure vs 100/100 when it does).

See section 3.2 in Osvik et. al. [1] for a more detailed description.

Your task is to recover these 64/128 bits using fewer than 16000 encryptions.

2.1 Bonus Points

- +1 – you need fewer than 500 encryptions
- +3 – full key recovery with any method, e.g., with last-round attack (no restriction to number of encryptions) [2]

3 Cache Covert Channel – 5.5 Points

Optional
Deadline: Thursday, May 8 2025, 8:00am

Implement a *fast* (see Table 1) and reliable cross-core covert channel based on a cache attack (Flush+Reload, Prime+Probe or Flush+Flush). Your channel must be able to take arbitrary *binary* files as an input parameter (not hardcoded), and output to a file. Transmit a file large enough that the transmission takes 60 seconds, with random (`/dev/rand`) or meaningful non-uniform content (`png`, `jpeg`, `mp3`, etc.). Compute the true channel capacity T as

$$T = C \cdot (1 + ((1-p) \cdot \log_2 (1-p) + p \cdot \log_2 (p))),$$

where C is the *raw capacity* and p is the *bit error ratio*.
You do not need to implement any error correction.

Capacity T (kiB/s)	> 0	7.5	13	20	27	35	45	55	67	**80**	130	200	500
Points	1	1.5	2	2.5	3	3.5	4	4.5	5	**5.5**	6	6.5	7

Table 1: True channel capacity to points.

3.1 Bonus Points

- +1 – Your group has the fastest channel[3].
- +1/2 – your channel uses Prime+Probe (+2) or Flush+Flush (+1).
- +1 – your Prime+Probe channel needs no access to physical addresses.
- +1 – your Prime+Probe channel works across virtual machines.

A scaling factor of 2/3 will be applied to the capacity requirement for Prime+Probe channels.

4 `Spectre` Attacks – 2.5 Points

Optional
Deadline: Thursday, May 8 2025, 8:00am

We have found a library that we think is susceptible to a Spectre-PHT attack[4]. We have provided it to you in your repository. You can access anything defined in the library's header to extract the secret string. You may not change the library or read the secret directly from memory. Most modern processors are affected by Spectre, but we recommend you use Intel or AMD. During the exercise interview, you will be given a new shared object file with a different secret, so make sure to test if your solution works for any string (though you can assume ASCII 63-95). Points will be awarded according to Table 2/Table 3 (Intel/AMD), though for this task your hardware and your explanation during the exercise interviews will also have an influence.

time to secret (s)	<160	<100	<70	<50	<**10**	<1	<0.5
Points	0.5	1	1.5	2	**2.5**	3	3.5

Table 2: Time to recovery of the (correct) secret to points on Intel.

time to secret (s)	<160	<100	<70	<50	<**15**	<2	<1
Points	0.5	1	1.5	2	**2.5**	3	3.5

Table 3: Time to recovery of the (correct) secret to points on AMD.

4.1 Bonus Points

- +1.5 – you can mistrain the target branch from a different process[5], without calling the library function in that process.

5 KASLR is bad, please break it – 2.5 Points

Optional
Deadline: Thursday, May 8 2025, 8:00am
Break kernel address space layout randomization using one of the methods presented in the lectures. Your goal is to find the starting virtual address of the Kernel within the possible address range. The simplest method is timing of prefetch instructions, but you can earn extra points for using Data Bounce[6]. If you use prefetch instructions, investigate which ones (or combinations thereof) work best for finding the kernel. You can also earn extra points for especially fast or robust implementations[7]. Please add a sample visualization (plot) of your program's output to your repo. For simplicity, we recommend using an Intel CPU for this task.

Notes

[1]Osvik et. al., Cache atacks and countermeasures: the case of AES, `https://eprint.iacr.org/2005/271.pdf`

[2]Irazoqui et.al., Wait a minute! A fast, Cross-VM attack on AES, `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=3514a3424fd6141c5f35d701c1023679cc8528f7`, Section 5.1

[3]Send a discord message to Lukas (redrabbyte) with *raw capacity*, *bit error ratio*, and any extra details you want included to update the speed record page `https://www.isec.tugraz.at/teaching/materials/scs/exercises/ex1/`. Final rankings are based on the exercise interviews.

[4]`http://spectreattack.com/`

[5]On newer hardware, you may need to deactivate STIBP, ask in Discord if you're unsure. `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/single-thread-indirect-br html`

[6]As described here `https://arxiv.org/pdf/1905.05725.pdf`

[7]Send a discord message to Lukas (redrabbyte) with *method*, *run time*, and *reliability*, to update the KASLR record Wiki page `https://www.isec.tugraz.at/teaching/materials/scs/exercises/ex1/`.