**Disclaimer:** We have tested it with many different cache side-channel PoCs, and carefully tuned thresholds, But some classes of side-channel attacks like *flush + flush attacks* or utilizing the branch target buffer are not detected - due to the fact that only cache characteristics are analysed.

**Idea: try to detect CPU characteristics, which can be traced back to side-channel attacks with a very high probability.** If these are detected, report the event to the system log. For example, print a trigger warning when these messages are detected.

# Usage and Example

Installation and Execution

cargo build  
$(pwd)/target/debug/detector -v

Detected Cache Side-Channel Attacks are printed to STDOUT or into systemd’s journal. See the example output for a spectre detection which is logged with log level error into systems journal.

Possible cache side-channel attack on CPU 0 detected!  
Cache miss/ref ratio 96.43% above trigger threshold of 90.00%  
Within 6 recorded seconds on CPU 0, 42230182 cache references where detected and 40724557 cache misses

# Implementation Aspects

This tool analyzes all available logical CPUs for abnormalities over a certain period of time. In doing so, it iterates over time and in a pseudo-random over the cache registers HPC to make countermeasures more difficult. Then Barnowl analyzes the cache reference and cache missrate for a certain amount of time - again pseudo-randomly. Here especially the last level cache characteristics. Basically the following Cache Side-Channel attacks should be detectable:

* Flush+Reload on AES
* Prime+Probe on AES
* Flush+Reload on RSA
* Spectre Attack

As noted earlier, flush+flush or Branch History Attacks are not detectable by this method.

For this purpose Detector tool uses the so-called Counting Mode of the Performance Monitor Unit (PMU) of the processor. In contrast to the Sampling Mode, the Counting Mode has defacto no measurable overhead.

The implementation is based on the perf subsystem of the Linux kernel which is designed around two aspects: flexibility and performance. In fact, perf can be seen as a command system call, which efficiently exchanges data between kernel space and user space using a ring buffer. So ideal conditions if these analyses are to be made with the goal of lowest overhead.

# Documentation for Detection Tool Code file in Rust

This Rust code is a tool for detecting potential cache side-channel attacks on a system by monitoring CPU cache activity. Below is the documentation detailing its functionality, structure, and usage. Overview

## The tool:

Monitors L2 cache references and cache misses on a specific CPU.  
Calculates the cache miss to cache reference ratio.  
Logs warnings if the ratio exceeds a defined threshold, indicating a potential side-channel attack.  
Provides options to run in verbose mode and as a daemon.

## Dependencies

The tool depends on several crates:

clap for command-line argument parsing.  
log for logging.  
nix for Unix-specific functionality.  
perf\_event for accessing hardware performance counters.  
rand for random number generation.  
systemd\_journal\_logger for logging to the systemd journal.

## Constants

CACHE\_MISS\_REF\_RATIO\_THRESHOLD: The ratio threshold for detecting a potential attack (87.6%).  
CACHE\_MISS\_IGNORE\_THRESHOLD: Minimum cache references per second to consider the data valid (10,000).  
RECORDING\_TIME\_MIN: Minimum recording time in seconds (2 seconds).  
RECORDING\_TIME\_MAX: Maximum recording time in seconds (5 seconds).