Detecting and Flagging Self-Timing Code

Cameron Wong, Joe Doyle

May 11, 2018

Abstract

With the advent of the SPECTRE vulnerability bringing new attention to hardware-level timing-based attacks, much research has gone into mitigating these at a software level. Most of these efforts approach the problem by disabling the attack vector entirely, but have so far shown to have serious performance ramifications. We implemented a source-level static taint analysis of C code as a proof-of-concept for detecting and marking "timed sections", as timed by "timer functions". This analysis might be extended to statically determine whether a program might execute a timing-based attack..

1 Introduction

"Timing attacks" are a broad category of attacks that can be used to covertly transmit information or inspect the internal state of a program by taking advantage of specific properties about the hardware. An important consequence about the physical nature of these attacks is that this can be used to bypass software-level protections by influencing hardware-internal processes and measuring the time taken to execute. The SPECTRE bug, for example, takes advantage of speculative execution and hardware caching effects to leak the contents of kernel memory.

The ability of a well-constructed timing attack to escape software jails and sandboxes is especially concerning in the age of browsers, in which users routinely download and run untrusted scripts and programs (often unknowingly) while browsing the web. Several browsers have implemented internal changes to combat this (such as reducing clock precision and disabling shared memory primitives) that have proven to be somewhat effective. However, there do not appear to be many attempts to statically verify that certain sections of a given program are safe to execute. Most attempts are "all-or-nothing" – mitigation must be used with entire applications or not at all. This can be a dealbreaker in performance- or correctness-critical situations, thus exposing them to severe security vulnerabilities[1].

It is critical, however, that the perpetrator of a timing attack have some capability to measure the relative execution time of a particular procedure. The original SPECTRE paper, for example, uses calls to GCC's builtin `__rtdscp` primitive

[1]See related work

to measure clock cycles taken to perform a particular malicious memory access[1]. Towards that end, we believe that it should be possible to statically detect and delineate between calls and stores of possible "timer functions" and use this data to apply further cache attack mitigation in a more fine-grained manner.

2 Background

2.1 Timing Attacks

The subject of this work is primarily attacks intended to exploit hardware-based caches via analysis of the total execution time of the victim to leak secret information to an adversary[2]. This is typically done by measuring the time taken to perform a large number of memory accesses within attacker-controlled code and inferring the total number of resulting cache hits and misses, which can then be used to guess unrelated information about the victim's internal state. The specifics of how exactly this is done and what information can be gleaned is specific to the attack and is beyond the scope of this work. We are instead concerned with how and where the time is measured, which can be used to demarcate code sections that may be used for a timing-based attack.

2.2 Timing Channels

Timing-based attacks require some way of recording the passage of time. This can be standard wall-clock time, but in practice there are any number of workable replacements. The only information of importance in this case is the *relative time* in some units of necessary precision between two distinct program states, so any monotonically-increasing counter will, in theory, be sufficient[2]. Of other interest is the distinction between concurrent and non-concurrent attacks, which distinguish whether the attacker is able to receive and react to information gained while the victim is executing[3]. Currently, our implementation only properly detects non-concurrent attacks. However, our model can be combined with more extensive analyses to detect concurrent variants of these attacks.

2.3 Taint Analysis

Taint analysis is a technique used to narrow down a source of vulnerabilities by isolating what internal state can or cannot be influenced by untrusted input. This is typically done by examining the *data flow* of a given piece of information – if tainted object X is used to compute the value of some value Y under some execution flow, the resulting value is considered tainted by the untrusted X, and cannot necessarily be trusted itself[4].

We adjusted the definition of "tainted" in this case to refer specifically to objects and values computed using a timer function (as, in theory, the entire program is considered untrusted). However, the core principles still hold.

3 Threat Model

Our model will detects attacks under the following constraints:

- An attacker may perform arbitrary local code execution within a defined environment via a known, untrusted channel. This environment may or may not be subject to external security measures such as operating system permissions, sandboxing, etc.

- The untrusted local code has no access to underlying hardware or operating system. This eliminates the possibility of other side channels, such as power consumption or sound analysis.

We believe this to be a realistic threat model, as it can be fitted to common attack vectors in real systems, such as

- Javascript injection into websites, executed via browser
- Malicious code published via a large package or dependency manager
- Malicious third-party code running in a sandboxed environment such as a cloud-hosted virtual machine

4 Solution

Let E be the set of expressions within our language.

Suppose that we have defined some trusted *timer detection function* $f : E \rightarrow \{\texttt{true}, \texttt{false}\}$ that can be used freely within the analysis[2].

For the purposes of this analysis, we will consider an *output* of a function to be any observable, concrete result (such as a print statement, a server response, direct return value, etc).

We can define our taint flow as follows:

- Any expression e is considered tainted by itself if $f(e)$.
- If a subexpression e' of e is tainted, then e is tainted by e'.
- For an assignable x, if there exists a reachable command $n = assign(x, e)$ where e is tainted, then x is tainted by e.

Finally, we consider an output of the procedure to be a potential leak if it is tainted by at least two unique nodes. We define the *critical section* of this value to be the possible control flow paths between any two taint sites. Once the dataflow analysis is complete, we can output these sections ourselves.

5 Results

Our proof-of-concept implementation is written in Haskell to analyze C; a more broadly practical version might instead target arbitrary Javascript or x86. We did not implement a particularly sophisticated timer analysis function f; we instead did a simple string comparison on function call name.

We wrote two implementations of common cache-based timing attacks as input data (see Appendix A):

- `cache-size.c`

  Our analysis correctly reports the critical sections between lines 43 and 52 by identifying the relevant dirty variable `time_taken` used to compute the output. It also identifies the section between lines 73 and 78.

  Our analysis currently also (erroneously) marks sections between 43 and 73, 52 and 73 (etc) as possible

[2]Our implementation uses simple variable-name comparison, but can be theoretically expanded to arbitrary analyses

critical sections. This is technically correct under the analysis, but not necessarily useful in this case. Improving the accuracy of the results will be a concern moing forwards.

Finally, we were able to correctly identify the variables `t1`, `t2`, `t3` and `t4` as variables containing timer state.

- `nproc.c`

  We correctly detect the critical timed section between lines 39 and 46, identifying `t1` and `t2` as the relevant timer variables.

All of our source code can be found on Github in the repository
jpdoyle/static-analysis-project

6 Limitations and Future Work

Timer Function Detection.

A major limitation is the necessity for a sophisticated timer detection function. An extension of this system would be to detect monotonically increasing shared variables for use as timers. We believe that current standard dataflow techniques are capable of detecting such, but this was slightly beyond the scope of our project.

Intraprocedural Analysis.

Our work is currently limited to analyzing single procedures. This is unrealistic for real-world use cases, but there is nothing inherent to our analysis that prevents it from being expanded to work intraprocedurally as well.

7 Related Work

Cache Side Channel Elimination.

There has been significant work into developing lightweight methods to eliminate cache-based timing side channels entirely via lazy cleansing of shared state before handing control over to untrusted processes[3]. We believe our work to be orthogonal to this in that Braun et al's work is primarily for mitigation at a blue-team (defending) source level, whereas our work is targeted at detecting red-team (attacking) efforts at the source level. The proposed mitigations require the ability to inspect and update the source code of security-critical applications to insert the necessary source annotations to demarcate secret sections. Our approach, while possibly less effective, is more conducive to higher-level defense layers that can be applied on top of applications for which adding the proposed source annotations is infeasible or impossible.

Retpoline and other software approaches

Google has released a mitigation strategy known as retpoline[5] that can inject software constructs into otherwise-vulnerable code to undirect speculative execution and avoid the cache-related issues. Similar work has been done by various compilers and service manufacturers[6] [7]. Much of this work, however, must be done at compile time or requires extensive other measures. Similar to above, we expect extensions of our work to instead be used to implement an "outer layer" around sensitive applications without necessitating modifying the application itself.

Robust Static Cache Analysis.

Doychev and Kopf implemented an automated static countermeasure against cache-based side channel attacks in executable code[8]. However, they appeared to be focused on analyzing the flow of cache memory itself, checking reads or writes to potentially cached memory addresses to determine what information may be leaked. As their implementation is a whole-program analysis, we belive that our work can be used to supplement theirs by directing attention to relevant fields.

8 Conclusion

Modern computer systems run untrusted code, sandboxed code at speeds that allow the sandboxed code to infer fine-grained information about the external system via precise timing. Many current mitigations (e.g. clock jitter, stronger isolation, removing concurrency features) are incomplete and often have significant negative impacts on performance and other.

By directing efforts towards automatic, course-grained security verification of untrusted code, the human burden of securing dependencies can be somewhat alleviated. Our program serves as a strong proof of concept for identifying possibly-untrusted timed code sections. If such a system is integrated into a web browser or hypervisor, timing-based attacks can be flagged and mitigations can automatically be taken. A mature version of this system could be used to decide whether or not performance-compromising mitigations need to be taken, allowing untrusted code to be run faster while remaining safe.

References

[1] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *CoRR*, vol. abs/1801.01203, 2018. [Online]. Available: http://arxiv.org/abs/1801.01203

[2] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, Apr 2018. [Online]. Available: https://doi.org/10.1007/s13389-016-0141-6

[3] B. A. Braun, S. Jana, and D. Boneh, "Robust and efficient elimination of cache and timing side channels," *CoRR*, vol. abs/1506.00189, 2015. [Online]. Available: http://arxiv.org/abs/1506.00189

[4] E. Barbosa, "Taint analysis," 2009, university Presentation. [Online]. Available: http://web.cs.iastate.edu/~weile/cs513x/2018spring/taintanalysis.pdf

[5] P. Turner, "Retpoline: a software construct for preventing branch-target-injection," blog Post. [Online]. Available: https://support.google.com/faqs/answer/7625886

[6] R. Biner, "Gcc 7.3 released," 2018. [Online]. Available: https://lwn.net/Articles/745385/

[7] M. Corporation, "Mitigating speculative execution side channel hardware vulnerabilities," 2018. [Online]. Available: https://blogs.technet.microsoft.com/srd/2018/03/15/mitigating-speculative-execution-side-channel-hardware-vulnerabilities/

[8] G. Doychev and B. Köpf, "Rigorous analysis of software countermeasures against cache attacks," *SIGPLAN Not.*, vol. 52, no. 6, pp. 406–421, Jun. 2017. [Online]. Available: http://doi.acm.org/10.1145/3140587.3062388

Appendix A Source code listings

cache-size.c:

```c
#include <pthread.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <stdlib.h>

#define N_BYTES (1<<26)
#define AVG_ITS (100)

struct timespec get_time() {
    struct timespec ret;
    clock_gettime(CLOCK_MONOTONIC,&ret);
    return ret;
}

int main(void) {
    uint8_t* data = malloc(N_BYTES);
    if(!data) {
        printf("Couldn't␣malloc␣data␣array\n");
        return -1;
    }
    size_t n_bytes_used;
    size_t i,j;

    double time_taken = 0;
    double prev_miss_rate = 0;

    for(n_bytes_used = 1; n_bytes_used < N_BYTES; n_bytes_used *= 2) {

        size_t misses = 0;
        double total_avg_time = 0;

        for(j=0; j<AVG_ITS; ++j) {

            for(i=0; i<n_bytes_used; ++i) {
                ++data[i];
            }

            {
                struct timespec t1,t2;

                t1 = get_time();

                {
                    volatile uint8_t x = 1;
                    for(i=0; i < n_bytes_used; ++i) {
                        x *= data[i];
                    }
                }

                t2 = get_time();

                time_taken = 1000.0*t2.tv_sec + 1e-6*t2.tv_nsec
                            - (1000.0*t1.tv_sec + 1e-6*t1.tv_nsec);
            }

            double avg_time;
            avg_time = time_taken/n_bytes_used;
            total_avg_time += avg_time;
            for(i=0; i<n_bytes_used; ++i) {
                ++data[i];
            }

            {

#define CHUNK_SIZE 128

                volatile uint8_t x = 1;
                for(i=0; i < n_bytes_used; i += CHUNK_SIZE) {
                    struct timespec t3,t4;
                    size_t nxt = i+CHUNK_SIZE;
                    t3 = get_time();

                    for(; i < n_bytes_used && i < nxt; ++i) {
                        x *= data[i];
                    }
                    t4 = get_time();

                    int missed;
                    missed = (1000.0*t4.tv_sec + 1e-6*t4.tv_nsec
                      - (1000.0*t3.tv_sec + 1e-6*t3.tv_nsec) > 1.1*CHUNK_SIZE*avg_time);
```

```
 83                     misses += missed;
 84                 }
 85             }
 86 
 87         }
 88 
 89         double miss_rate;
 90         miss_rate = (misses/(double)AVG_ITS)
 91                         /(n_bytes_used/(double)CHUNK_SIZE);
 92 
 93         printf("␣␣%010lu␣bytes:␣%lf␣ms/byte,␣~%lf␣miss␣rate␣(%lu␣misses)\n", n_bytes_used,
 94                 total_avg_time/AVG_ITS, miss_rate, misses);
 95 
 96         if(misses > 3 && miss_rate > 1.3*prev_miss_rate) {
 97             printf("Probable␣cache␣size:␣%lu\n", n_bytes_used/2);
 98         }
 99 
100         prev_miss_rate = miss_rate;
101     }
102 
103     printf("Done!\n");
104     return 0;
105 }
```

Analysis:

```
data/cache-size.c:
===== main =====

("data/cache-size.c": line 43):
        Ident "t1" 6388 (NodeInfo ("data/cache-size.c": line 43) (("data/cache-size.c": line 43),2) (Name {nameId =
              18694}))
        Ident "missed" 243083090 (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) (Name
             {nameId = 18894}))
        Ident "misses" 243085010 (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) (Name
             {nameId = 18931}))
        Ident "total_avg_time" 408095412 (NodeInfo ("data/cache-size.c": line 60) (("data/cache-size.c": line 60)
             ,14) (Name {nameId = 18784}))
        Ident "avg_time" 414543957 (NodeInfo ("data/cache-size.c": line 59) (("data/cache-size.c": line 59),8) (
             Name {nameId = 18776}))
        Ident "prev_miss_rate" 428921654 (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100)
             ,14) (Name {nameId = 19002}))
        Ident "time_taken" 439625400 (NodeInfo ("data/cache-size.c": line 54) (("data/cache-size.c": line 54),10) (
             Name {nameId = 18739}))
        Ident "miss_rate" 487943857 (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) (
             Name {nameId = 18947}))


("data/cache-size.c": line 52):
        Ident "t2" 6516 (NodeInfo ("data/cache-size.c": line 52) (("data/cache-size.c": line 52),2) (Name {nameId =
              18732}))
        Ident "missed" 243083090 (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) (Name
             {nameId = 18894}))
        Ident "misses" 243085010 (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) (Name
             {nameId = 18931}))
        Ident "total_avg_time" 408095412 (NodeInfo ("data/cache-size.c": line 60) (("data/cache-size.c": line 60)
             ,14) (Name {nameId = 18784}))
        Ident "avg_time" 414543957 (NodeInfo ("data/cache-size.c": line 59) (("data/cache-size.c": line 59),8) (
             Name {nameId = 18776}))
        Ident "prev_miss_rate" 428921654 (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100)
             ,14) (Name {nameId = 19002}))
        Ident "time_taken" 439625400 (NodeInfo ("data/cache-size.c": line 54) (("data/cache-size.c": line 54),10) (
             Name {nameId = 18739}))
        Ident "miss_rate" 487943857 (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) (
             Name {nameId = 18947}))


("data/cache-size.c": line 73):
        Ident "t3" 6644 (NodeInfo ("data/cache-size.c": line 73) (("data/cache-size.c": line 73),2) (Name {nameId =
              18851}))
        Ident "missed" 243083090 (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) (Name
             {nameId = 18894}))
        Ident "misses" 243085010 (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) (Name
             {nameId = 18931}))
        Ident "prev_miss_rate" 428921654 (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100)
             ,14) (Name {nameId = 19002}))
        Ident "miss_rate" 487943857 (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) (
             Name {nameId = 18947}))


("data/cache-size.c": line 78):
        Ident "t4" 6772 (NodeInfo ("data/cache-size.c": line 78) (("data/cache-size.c": line 78),2) (Name {nameId =
              18881}))
        Ident "missed" 243083090 (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) (Name
             {nameId = 18894}))
        Ident "misses" 243085010 (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) (Name
             {nameId = 18931}))
        Ident "prev_miss_rate" 428921654 (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100)
            ,14) (Name {nameId = 19002}))
        Ident "miss_rate" 487943857 (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) (
            Name {nameId = 18947}))



section:
 (NodeInfo ("data/cache-size.c": line 43) (("data/cache-size.c": line 43),1) Name {nameId = 18698})
^
v
 (NodeInfo ("data/cache-size.c": line 52) (("data/cache-size.c": line 52),1) Name {nameId = 18737})
  timed by:
        (NodeInfo ("data/cache-size.c": line 54) (("data/cache-size.c": line 54),10) Name {nameId = 18741})
        (NodeInfo ("data/cache-size.c": line 59) (("data/cache-size.c": line 59),8) Name {nameId = 18777})
        (NodeInfo ("data/cache-size.c": line 59) (("data/cache-size.c": line 59),10) Name {nameId = 18779})
        (NodeInfo ("data/cache-size.c": line 60) (("data/cache-size.c": line 60),14) Name {nameId = 18786})
        (NodeInfo ("data/cache-size.c": line 60) (("data/cache-size.c": line 60),8) Name {nameId = 18788})
        (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) Name {nameId = 18895})
        (NodeInfo ("data/cache-size.c": line 82) (("data/cache-size.c": line 82),8) Name {nameId = 18927})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18933})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18935})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) Name {nameId = 18948})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),6) Name {nameId = 18950})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),14) Name {nameId = 18972})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),9) Name {nameId = 18976})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),6) Name {nameId = 18978})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),6) Name {nameId = 18982})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),9) Name {nameId = 18986})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),14) Name {nameId = 18989})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),14) Name {nameId = 19005})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),9) Name {nameId = 19007})

section:
 (NodeInfo ("data/cache-size.c": line 43) (("data/cache-size.c": line 43),1) Name {nameId = 18698})
^
v
 (NodeInfo ("data/cache-size.c": line 73) (("data/cache-size.c": line 73),1) Name {nameId = 18855})
  timed by:
        (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) Name {nameId = 18895})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18933})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18935})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) Name {nameId = 18948})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),6) Name {nameId = 18950})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),9) Name {nameId = 18976})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),6) Name {nameId = 18978})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),6) Name {nameId = 18982})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),9) Name {nameId = 18986})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),14) Name {nameId = 18989})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),14) Name {nameId = 19005})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),9) Name {nameId = 19007})

section:
 (NodeInfo ("data/cache-size.c": line 43) (("data/cache-size.c": line 43),1) Name {nameId = 18698})
^
v
 (NodeInfo ("data/cache-size.c": line 78) (("data/cache-size.c": line 78),1) Name {nameId = 18887})
  timed by:
        (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) Name {nameId = 18895})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18933})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18935})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) Name {nameId = 18948})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),6) Name {nameId = 18950})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),9) Name {nameId = 18976})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),6) Name {nameId = 18978})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),6) Name {nameId = 18982})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),9) Name {nameId = 18986})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),14) Name {nameId = 18989})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),14) Name {nameId = 19005})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),9) Name {nameId = 19007})

section:
 (NodeInfo ("data/cache-size.c": line 52) (("data/cache-size.c": line 52),1) Name {nameId = 18737})
^
v
 (NodeInfo ("data/cache-size.c": line 73) (("data/cache-size.c": line 73),1) Name {nameId = 18855})
  timed by:
        (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) Name {nameId = 18895})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18933})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18935})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) Name {nameId = 18948})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),6) Name {nameId = 18950})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),9) Name {nameId = 18976})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),6) Name {nameId = 18978})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),6) Name {nameId = 18982})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),9) Name {nameId = 18986})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),14) Name {nameId = 18989})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),14) Name {nameId = 19005})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),9) Name {nameId = 19007})

section:
 (NodeInfo ("data/cache-size.c": line 52) (("data/cache-size.c": line 52),1) Name {nameId = 18737})
^
v
 (NodeInfo ("data/cache-size.c": line 78) (("data/cache-size.c": line 78),1) Name {nameId = 18887})
  timed by:
        (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) Name {nameId = 18895})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18933})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18935})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) Name {nameId = 18948})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),6) Name {nameId = 18950})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),9) Name {nameId = 18976})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),6) Name {nameId = 18978})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),6) Name {nameId = 18982})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),9) Name {nameId = 18986})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),14) Name {nameId = 18989})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),14) Name {nameId = 19005})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),9) Name {nameId = 19007})

section:
 (NodeInfo ("data/cache-size.c": line 73) (("data/cache-size.c": line 73),1) Name {nameId = 18855})
^
v
 (NodeInfo ("data/cache-size.c": line 78) (("data/cache-size.c": line 78),1) Name {nameId = 18887})
  timed by:
        (NodeInfo ("data/cache-size.c": line 81) (("data/cache-size.c": line 81),6) Name {nameId = 18895})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18933})
        (NodeInfo ("data/cache-size.c": line 83) (("data/cache-size.c": line 83),6) Name {nameId = 18935})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),9) Name {nameId = 18948})
        (NodeInfo ("data/cache-size.c": line 90) (("data/cache-size.c": line 90),6) Name {nameId = 18950})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),9) Name {nameId = 18976})
        (NodeInfo ("data/cache-size.c": line 94) (("data/cache-size.c": line 94),6) Name {nameId = 18978})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),6) Name {nameId = 18982})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),9) Name {nameId = 18986})
        (NodeInfo ("data/cache-size.c": line 96) (("data/cache-size.c": line 96),14) Name {nameId = 18989})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),14) Name {nameId = 19005})
        (NodeInfo ("data/cache-size.c": line 100) (("data/cache-size.c": line 100),9) Name {nameId = 19007})
```

nproc.c:

```c
#include <pthread.h>
#include <time.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>

struct timespec get_time() {
    struct timespec ret;
    clock_gettime(CLOCK_MONOTONIC,&ret);
    return ret;
}

void work() {
    volatile double d = 0;
    for (int n=0; n<10000; ++n) {
        for (int m=0; m<10000; ++m) {
            d += d*n*m;
        }
    }
}

typedef struct {
    pthread_mutex_t lock;
    double work_time;
    size_t num_samples;
    int keep_running;
} timing_data;

void* timing_thread(void* thd_data) {
    timing_data* data = thd_data;
    struct timespec t1,t2;

    double outer_time = 0, inner_time = 0;

    while(data->keep_running) {
        t1 = get_time();

        work();

        t2 = get_time();
        pthread_mutex_lock(&data->lock);
        inner_time = 1000.0*t2.tv_sec + 1e-6*t2.tv_nsec
                     - (1000.0*t1.tv_sec + 1e-6*t1.tv_nsec);
        data->work_time += inner_time;
        ++data->num_samples;
        pthread_mutex_unlock(&data->lock);
    }
    return NULL;
}

#define MAX_N_THREADS 100

int main(void) {
    size_t i = 0;
    timing_data timings[MAX_N_THREADS];
    memset(timings,0,sizeof(timings));
    pthread_t threads[MAX_N_THREADS];
    memset(threads,0,sizeof(threads));
    size_t n_threads;

    for(i = 0; i < MAX_N_THREADS; ++i) {
        pthread_mutex_init(&timings[i].lock,NULL);
    }

    int nproc_found = 0;
    double prev_max_time = 0;
    double min_time = 100, max_time = 0;

    while(!nproc_found) {
        if(n_threads >= MAX_N_THREADS) {
            return -1;
        }

        size_t ix = n_threads++;
        timings[ix].keep_running = 1;
        pthread_create(&threads[ix],NULL,&timing_thread,&timings[ix]);

        usleep(50000);

        min_time = 100000;
        max_time = 0;
        for(i=0; i < n_threads; ++i) {
            pthread_mutex_lock(&timings[i].lock);
            while(!timings[i].num_samples) {
                pthread_mutex_unlock(&timings[i].lock);
                usleep(50000);
                pthread_mutex_lock(&timings[i].lock);
            }

            double on_time = timings[i].work_time/timings[i].num_samples;
            timings[i].work_time = 0;

            timings[i].num_samples = 0;

            pthread_mutex_unlock(&timings[i].lock);


            if(on_time > max_time) {
                max_time = on_time;
            }
            if(on_time < min_time) {
                min_time = on_time;
            }
        }

        printf("%lu␣threads,␣Min␣time:␣%lf,␣Max␣time:␣%lf\n",
                n_threads, min_time, max_time);
        if(max_time > 1.2*min_time) {
            nproc_found = 1;
        }
    }

    printf("Found!␣nproc␣=␣%lu\n",n_threads-1);
    printf("Joining␣timing␣threads...\n");

    for(i=0; i < n_threads; ++i) {
        timings[i].keep_running = 0;
    }

    for(i=0; i < n_threads; ++i) {
        pthread_join(threads[i],NULL);
    }

    printf("Done!\n");
    return 0;
}
```

Analysis:

```
data/nproc.c:
===== timing_thread =====

("data/nproc.c": line 36):
        Ident "t1" 6388 (NodeInfo ("data/nproc.c": line 36) (("data/nproc.c": line 36),2) (Name {nameId = 16123}))
        Ident "data" 205336804 (NodeInfo ("data/nproc.c": line 44) (("data/nproc.c": line 44),4) (Name {nameId =
             16180}))
        Ident "inner_time" 435755592 (NodeInfo ("data/nproc.c": line 42) (("data/nproc.c": line 42),10) (Name {
             nameId = 16149}))


("data/nproc.c": line 40):
        Ident "t2" 6516 (NodeInfo ("data/nproc.c": line 40) (("data/nproc.c": line 40),2) (Name {nameId = 16133}))
        Ident "data" 205336804 (NodeInfo ("data/nproc.c": line 44) (("data/nproc.c": line 44),4) (Name {nameId =
             16180}))
        Ident "inner_time" 435755592 (NodeInfo ("data/nproc.c": line 42) (("data/nproc.c": line 42),10) (Name {
             nameId = 16149}))



section:
 (NodeInfo ("data/nproc.c": line 36) (("data/nproc.c": line 36),1) Name {nameId = 16127})
^
v
 (NodeInfo ("data/nproc.c": line 40) (("data/nproc.c": line 40),1) Name {nameId = 16138})
  timed by:
        (NodeInfo ("data/nproc.c": line 35) (("data/nproc.c": line 35),4) Name {nameId = 16120})
        (NodeInfo ("data/nproc.c": line 41) (("data/nproc.c": line 41),4) Name {nameId = 16144})
        (NodeInfo ("data/nproc.c": line 42) (("data/nproc.c": line 42),10) Name {nameId = 16151})
        (NodeInfo ("data/nproc.c": line 44) (("data/nproc.c": line 44),4) Name {nameId = 16182})
        (NodeInfo ("data/nproc.c": line 44) (("data/nproc.c": line 44),10) Name {nameId = 16186})
        (NodeInfo ("data/nproc.c": line 45) (("data/nproc.c": line 45),4) Name {nameId = 16190})
        (NodeInfo ("data/nproc.c": line 46) (("data/nproc.c": line 46),4) Name {nameId = 16198})
```