COMPX304 Assignment 1

Ethan McKee-Harris

# Abstract

The opportunity to gain information about systems as a side affect of non-malicious code is commonly known as a side channel attack. In this report we detail a side channel attack used in order to determine the size of a machines L3 cache through the usage of array lookup times.  
The methods detailed in this report provide simple and scalable ways to aggregate the information a system leaks during program execution in order to find the size of the L3 cache. Further, various methods were used to varying successes throughout this assignment. I found that utilising data gathered by timing the overall time to do arbitrary actions on an array yielded better results than timing each individual arbitrary action. After computation, I can deduce that the methods used through this report outputted data which correlated with the true sizes of the test machines L3 cache.

# Implementation Size Baseline & Cache lines

Text

Description automatically generatedWhen attempting to time side channel-based attacks, we need some form of reliable baseline in terms of size for our data. In both of the implementations in this report all sizes are in mb, so we also need to match that with both approaches discussed in the report. On a more technical note, this is implemented through the usage of a method which returns the required array size in order to create an array of int’s which matches the requested size.

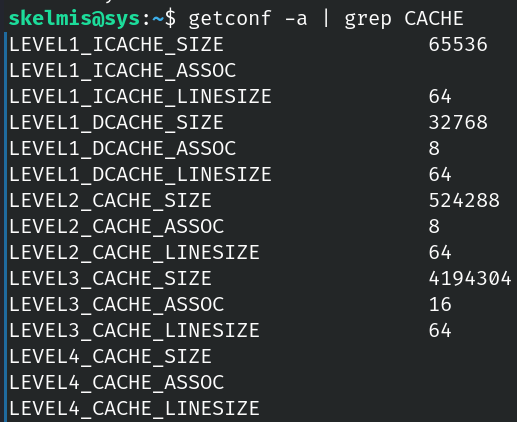
Text

Description automatically generated

Psudocode showing the relative usage of the getArraySizeFor method.

Given both approaches utilise the ability to know your target machines cache lines in advance, it is relevant to discuss it here. In computing, a cache line can more commonly be thought of as simply the amount of memory that is transferred to the cache for each ‘block’. Given the information the system leaks is going to be related to these lines, it makes sense to optimize access to use a new cache line for every arbritarty action taken within our loop. This means that the code is going to be Text

Description automatically generatedaccessing/populating new cache lines as much as possible rather than simply performing actions on already populated cache lines. This optimization means our code is more productive, as it is activelly attempting to load as many elements into the L3 cache as possible which results in better data output.

Within both of our approaches, we perform the abritary action on a different cache line for every iteration of the loop. This aims to maximise our actions and usage of the machines cache.

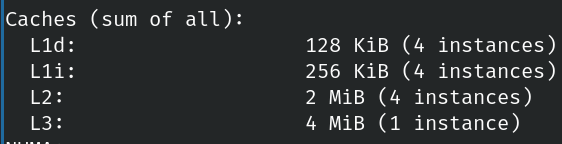
In order to find he cache line size for your machine, refer to the image on the right side.

# Chart, histogram Description automatically generatedInitial Approach

Initially, I decided that we would be able to retrieve more accurate data if the program timed how long each arbitrary action took, before graphing that data. This resulted in clear spikes as can be seen in the graph, however, they do not appear to be conclusive for the L3 cache.

Text

Description automatically generatedWhen viewing the output of lscpu (Bottom right image), we can see that the graph on the right correlates well to the 512KiB of per core L2 cache however it does not appear to meet any of the L3 cache values which we would expect near the end of the graph.  
I believe that the reason for this inconsistency is down to the method used for testing, which was timing individual access times rather then the entire array. This belief would turn out to be true when working on the final approach which is discussed later in this report.



# Final Implementation

Text

Description automatically generatedBased on the results of the previous approach, I figured it would be better to simply time the access for the overall loop as shown in the right hand image.   
Further to this, we are also now properly utilising the cache lines in order to optimize this loop and by extension, the side channel attack.

On a side note, both approaches attempt to reduce the amount of ‘noise’ within the data output by the program by calling the garbage collector prior to the loop itself. This should in theory mean our data is not polluted with any left-over values from previous iterations or Java overhead. However, this only affects the program itself and other items running on the target machines system may influence the output and that is out of our control.

Chart, line chart

Description automatically generated

As we can see in the above graph, this approach has yielded noticeable results, with somewhat clear jumps at the relevant cache levels. However, even with this approach we still struggle to find clear cut segments between the various cache levels so further refinement would be necessary.

You will also note that in this graph, we can see that the time for each array is actually just increasing for most iterations regardless of the cache being used. This behaviour is expected, as the size every iteration is increasing so we can expect it to take longer to perform the arbitrary action on the entire array.

For further accuracy there are two possible avenues which should be explored.   
Firstly, the end user should attempt to make sure that there are no other processes running on the target machine. By removing other running processes, you can ensure that the data returned from this program is accurate as other processes may be using the cache themselves which distorts the results.  
Secondly, the program could be tailored to smaller intervals then half a mib. This would allow the generated plot to be more accurate which could result in a better guess at the exact size of the L3 cache. I did not use a smaller value however as during my research I found most machines to use intervals in the mib range so I felt that going up by half an mib every iteration would yield accurate enough data.

# Conclusion

To conclude, this report showcased two methods which can be used to gain information about systems as a side effect of non-malicious code. These side channel attacks showed to varying degrees of accuracy, the size of the target machines L3 cache utilising methods which are easily scalable for any target machine.

This report also detailed the differences between approaches, and how timing the overall time the program took to do an arbitrary action on the entire array yielded better results than timing each individual arbitrary action. When using data from both methods, we could conclude that the target machines L3 cache appeared to be between 2.5mib and 4mib with reliable accuracy, however, we could not get a more concise answer as the data was likely affected by noise from other programs on the target machine who were also utilising the machines caches.