International Journal of Computer & Information Technologies (IJOCIT)
www.ijocit.org & www.ijocit.ir
ISSN = 2345-3877

A Cooperative GPU-Based Approach for Alert Aggregation

Masoud Narimani Zaman Abadi[1], Alireza Nowroozi[2], Payam Mahdinia[3]
IT Security Institute, ICT Department[1,2], Electrical and Computer Engineering Department[3]
MalekAshtar University of Technology, Tehran[1, 2], Isfahan University of Technology[3]
narimani.msd@gmail.com[1], nowroozi@mut.ac.ir[2], p.mahdiniaalvar@ec.iut.ac.ir[3]

Abstract: - Alert aggregation classified as a similarity-based alert correlation which fuses and clusters similar alerts. Alert aggregation increases meaning of alerts and reduces incoming alerts simultaneously; this process requires lots of computing resources. Limitation of computing resources, like CPUs, makes such systems not satisfactory. Graphic processing units (GPUs) are a potential option to solve this. In recent years, GPUs have been used in various fields, however, due to the dynamic nature of processing and data structures in alert correlation, correlation algorithms have not been implemented on GPU. In this paper, we present a cooperative model that uses the processing power of graphics processing unit (GPU) to aggregate security alerts and transform the time complexity from the second power to the linear one. Evaluations illustrate the proposed method for 600,000 alerts in time window will improve the processing speed by 26 times. In the proposed algorithm, in spite of main algorithm, the system performance at best, average and worst cases are the same.

Keywords*:* Alert aggregation, alert correlation, security alert, graphics processor, time window.

1. Introduction

Alert correlation analysis is one of the core functions of security operations center, which can avoid false and duplicate report, contribute to find some potential threats and improve the efficiency and security of the network [1]. The performance of the system should be such that it can process incoming alerts online. Similarity-based algorithms are a subcategory of correlation algorithms which is widely used. It utilizes similarity metrics to correlates alerts. Techniques used in this subcategory are classified as filtering or aggregation. Filter-based techniques perform fixed task on each alert and according to the

acquired results make decision about alerts. Alert verification and prioritization are in this class. The goal of aggregation technique is fusing or clustering incoming alerts. This technique utilizes similarity metrics and put similar alerts into one category. The algorithms that use aggregation technique require a lot of computing resources and this restriction is as a bottleneck in correlation system [2]. The time complexity of aggregation algorithms is $O(r^2)$ in which r is the rate of input alerts. With the increase in processing power, either it can process alerts with a higher rate or it can keep more alerts in the alert queue and these improve quality of correlation.

Graphics processors have developed very rapidly in recent years. On each new generation, additional features are introduced that move the GPUs one step closer to wider use for general purpose computations [3]. The use of a GPU beside a CPU to perform general-purpose computations is known as General Purpose computing on Graphics Processing Units (GPGPU). Lots of research have been done to improve the performance of variant algorithms in GPUs. This paper presents a prototype for CPU-GPU cooperative model that can integrate the computing power of CPU and GPU to perform alert aggregation more efficiently. In this model, aggregation system selects best platform to process alerts. The movement between CPU and GPU platforms depends on the alert queue size. Evaluations show that this prototype can aggregate security alerts 26X faster than CPU-based model when the alert queue size is 600,000.

The rest of this paper is organized as follow: First, a brief discussion of related work around correlation and GPU is given in section II. Section III provides a template for serial aggregation algorithm and a proposed template for GPU-based parallel aggregation algorithm. Section IV describes proposed model and finally, Section V draws conclusions and outlines future work.

2. Related Work

Most of proposed solutions presented in correlation field such as [4–8] are offline tools that can be run periodically on a database of

alerts and cannot operate with good performance. In addition, the solutions such as [9–11] that do operate online have problem in performance and are only able to operate in real-time on datasets with a low alert-rate. The work described in [9] considers the performance of alert correlation by utilizing a Column-oriented Database and an In-Memory Alert Storage in connection with Improved Algorithms using Memory-based Index Tables. Valeur in [11] uses a hash table to perform the lookup in the state, but the problem with hash tables is that when the table fills up the performance of search and deletion decreases. In addition, in Valeur model, indexes consume valuable memory and can be costly to update. The aim of [10] is to develop an automated alert correlation algorithm using attack type graphs which is suitable for deployment in a real-time environments. [2] Implies to limitations of correlation systems in terms of performance and proposes to pre-filter information at the source. Such a filter will reduce the amount of alerts collected. To do pre-filtering two factors are used: structural specifications and security policy. This approach has two main drawbacks: The first is the difficulty to maintain such distributed filters, and the second one is the lack of universality concerning security related events on the systems. There are many works related to GPU that utilize this processor to improve algorithm performance, but there is not any effort on correlating alert via GPU power.

3. Aggregation Algorithm

Alert aggregation is a class of alert correlation that uses similarity metrics. This class of correlation includes the fusion, session reconstruction, thread reconstruction, focus recognition, and multi-step correlation algorithms [11]. The main feature of these algorithms is need to time window. Time window keeps incoming alerts at an alert queue in order to process them later. In order to implement an aggregation component, two different functions are needed: a matching function and a merging function. A matching function takes two arguments: a set of related

alerts, and a single alert to be matched with the set. The matching function compares the incoming alert to be matched with all the alerts in the set (the alert queue). If the alert belongs to the set (i.e., the function's similarity criteria are satisfied), then the alerts in the set and the matched alert should be aggregated and the function returns true. On the other hand, if the incoming alert is not similar to the alerts in the alert set, then false is returned and alert is added to the alert queue. A merging function takes as input a set of alerts that have previously been matched by the matching function and returns a hyper-alert representing the aggregate of the set. If no aggregate can be calculated, the function returns null. The alerts within the time window are stored in a time-ordered queue. Serial and parallel algorithms are presented in the following sections:

A. Serial Algorithm

When a new alert arrives, serial algorithm compares it with the alerts in the queue, starting with the first alert in the queue and moves towards the end of the queue.

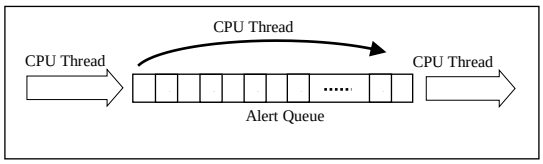


Figure 1: Alert processing in serial mode

Figure 1 shows process. Upon finding a match, the two alerts are merged, the resulting hyper-alert replaces with the matched alert in the queue, and the operation is terminated. If no match is found after searching through the whole queue, the alert is inserted into the queue, to be considered for matching with future alerts. If the rate of incoming alerts was *r*, the time window has in average *r*window_size* alerts and the system has to compare *r*r*window_size* alerts per second. As a result, the time complexity of the matching phase becomes $O(r^2)$. With this time complexity, correlation system cannot process alerts at a high rate [11]. Therefore, in these conditions, the system is forced either to reduce the algorithm details or to remove additional alerts from input buffer; anyway, this decreases the quality of the resulting output. Window size is considered in [12] 120 seconds,

so it is susceptible to a big alert queue. Until the alert queue size is less than a threshold, algorithm performance is acceptable. However, with growing size of the queue, the algorithm performance is sharply reduced. Therefore, researchers use various methods such as keeping small window size or reducing the rate of incoming alerts to hide the problem. The critical part of this algorithm is the loop that compares new alert with alerts in the queue.

B. GPU-based Parallel Algorithm

In our proposed approach, we have used GPU parallel processing power to handle alerts. Graphics processors have many cores that work simultaneously and are capable to run thousands of threads concurrently. This capability can remove comparing loop in serial algorithm. Figure 2 illustrates GPU-based parallel processing. Here, the alert queue should be transferred to device (GPU) memory. Each thread is responsible for one alert in the alert queue and has to process it.

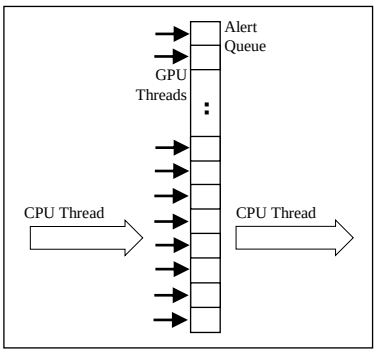


Figure 2: Alert processing in parallel mode

Once host (CPU) receives a new alert, after pre-processing it, shifts alert to device and introduce it to device threads. The threads simultaneously compare new incoming alert with the alert that are responsible for. If a thread finds a match, it calls merging function to aggregate two alerts. Matching process is such that the new alert is matched with no alerts or be matched with only one alert. Therefore, upon detecting a matching case in a thread, we can call the merging function, because it sure which does not find any matches at other threads. When all threads terminate and do not find any matches, a flag announces the status to the host. The host checks the flag and if finds that no match occurred, adds new alert into the alert queue.


```
Parameter alert_queue_ size, block_size, merge_flag, t1, t2
Global alert_queue
Aggregation (alert) {

    Remove all ra: Alert from alert_queue where ra.queue_index
    > alert_queue_size

    Pass removed alerts to next correlation component

    If (t1 < alert_queue_ size < t2)
        Keep previous process state
    Else if (alert_queue_ size ≤ t1)
        If (previous process state is not serial)
            Move the alert queue to host
        Serial_aggregation (alert)
    Else
        If (previous process state is not parallel)
            Move the alert queue to device
        Parallel_aggregation (alert)                    }

Serial_aggregation (alert) {

    For each a: Alert in alert_ queue
        If (alert.attributes == a.attributes)
            Send a and alert to Merge_Template phase

        Else
            Add alert to alert_queue
        }
Parallel_aggregation (alert) {

    kernel_compare_merge <<<alert_queue_size/ block_size,
    block_size>>> (alert)

    If (merge_flag is set)
        kernel_add<<<1, 1>>> (alert)                    }

kernel_compare_merge (alert) {

    Set this thread to a: Alert from alert_queue

    If (thread_index is less than alert_queue_size)
        If (alert.attributes equal a.attributes)
            Send a and alert to Merge_Template phase
            Set merge_flag                              }
```

Figure 3: Pseudo code of the proposed model

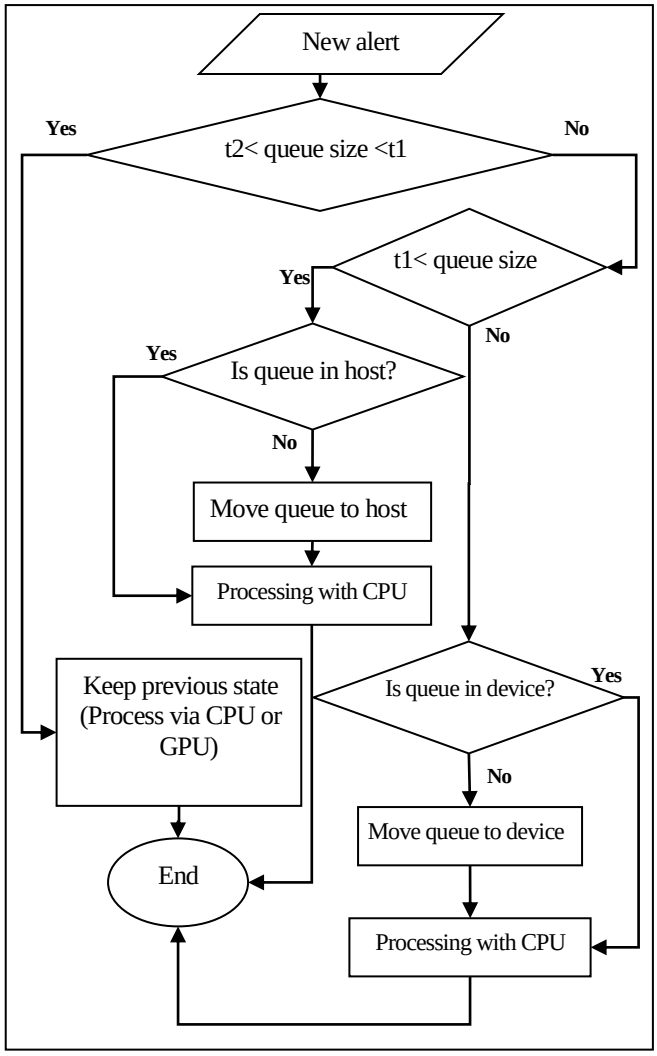


Figure 4: Flowchart of the proposed model

With the arrival of a new alert, host creates a kernel code – a code that executes on device - and transfers control to the device. Device runs kernel, which compares alerts in GPU, and when finished, returns control to the host. In this step, the host checks status flag. If the flag shows that no match found, the host creates other kernel code to add new alert into the alert queue located in device memory. Control switching between host and device imposes overhead on the system. When the alert queue length is small, switch overhead is significant, but when the queue length exceeds a threshold, this overhead is negligible and can be ignored.

4. Cooperative Model

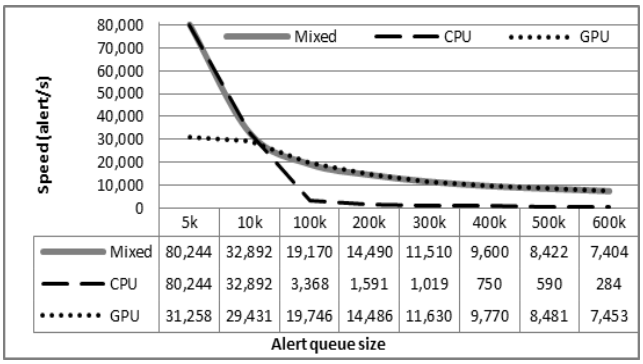
When the size of the alert queue is smaller than threshold (t0), performance of the serial algorithm is better than parallel case. Although, with the increase of the queue length process time of the serial algorithm increases drastically, while the parallel algorithm does not. In parallel case, with increase of the queue length, the curve is a little steep rises (Figure 5). Therefore, in order to take advantage of both algorithms, we use a hybrid model. This model defines two thresholds, t1 and t2, in such that t1<t0<t2. As long as the queue length is smaller than t1, CPU is responsible for processing incoming alerts. When the queue length is greater than t2, processing is transferred to the GPU. During processing an alert on a platform, the alert should be within that platform. Therefore, concurrent with the transfer of control from one platform to another platform, the alert queue must be copied in its memory. Transferring data between the host (CPU) and the device (GPU) has an overhead to the system. Thus, to reduce unnecessary transfers when the queue size threshold is near t0, we will define a safety margin (the distance between t1 and t2). In this interval, the process is maintained in previous state. This means that if the queue length is smaller than t1 initially, with entering the size of the queue to the safety margin, the host still will process alerts. As well as, if the queue length is bigger than t2, with entering the size of the queue to the safety margin, the device will process alerts. Figure 4 shows pseudo code of our approach and in Figure 6, we depict flowchart of our model. For our experiments, we used 11,000 as the threshold determined heuristically by choosing a size and then analyzing the results of the aggregation process. With 9 percent safety margin, t1 and t2 will be 10,000 and 12,000.

5. Experimental Results

In this evaluation, the performance of alert processing in serial and parallel is calculated. Figure 5 shows output results in terms of processing time. As shown in the figure, when the queue length is about 11,000 processing

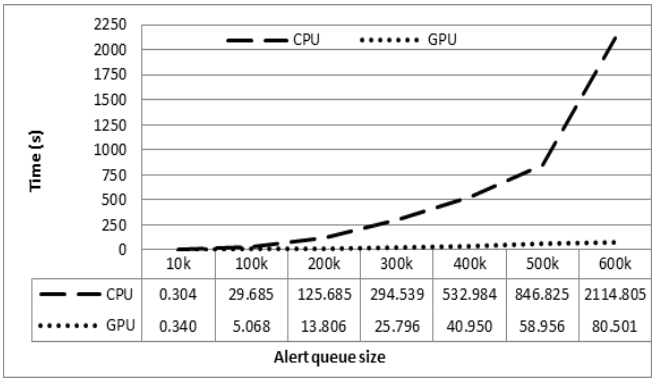
speed of the host and the device is almost equal. When the queue size reaches 600,000 alerts, processing speed of the host is the average of 284 alerts per second, while the processing speed of the device is equal to 7,453 alerts per second. It means achieving a speedup of up to 26. In Figure 6, we show results in term of processing rate. The gray curve in Figure 6 (Mixed mode) shows the processing speed of the proposed model.

When the queue size is smaller than the threshold, host processes alerts and when it is larger than the threshold, the device is used for processing alerts. The experiments were executed locally on an Intel CPU with a 2.93GHz clock speed. The Graphics Processing Unit used was the NVIDIA GTX 580 with 512 cores and the operating system used was an Ubuntu 11.10. Consider that the details associated with the operation of both algorithms are the same, so the results are comparable.



	5k	10k	100k	200k	300k	400k	500k	600k
Mixed	80,244	32,892	19,170	14,490	11,510	9,600	8,422	7,404
CPU	80,244	32,892	3,368	1,591	1,019	750	590	284
GPU	31,258	29,431	19,746	14,486	11,630	9,770	8,481	7,453

Figure 5: Processing time of algorithms in CPU and GPU



	10k	100k	200k	300k	400k	500k	600k
CPU	0.304	29.685	125.685	294.539	532.984	846.825	2114.805
GPU	0.340	5.068	13.806	25.796	40.950	58.956	80.501

Figure 6: Processing speed of algorithms

References

[1] S. Yuan and C. Zou, "The security operations center based on correlation analysis," 2011 IEEE 3rd International Conference on Communication Software and Networks, pp. 334–337, May 2011.

[2] R. Bidou, "Security operation center concepts & implementation," avalable at http://www. iv2-technologies. com/, 2005.

[3] R. Wu and B. Zhang, "Clustering Billions of Data Points Using GPUs," UCHPC-MAW'09, pp. 1–5, 2009.

[4] S. Xiao, Y. Zhang, X. Liu, and J. Gao, "Alert Fusion Based on Cluster and Correlation Analysis," 2008 International Conference on

Convergence and Hybrid Information Technology, 2008.

[5] H. Waita Njogu, "Using Alert Cluster to reduce IDS alerts," 2010 3rd International Conference on Computer Science and Information Technology, Jul. 2010.

[6] A. Valdes and K. Skinner, "Probabilistic Alert Correlation," SRI International, pp. 54–68, 2001.

[7] D. Li, Z. Li, L. Wang, and M. Roesch, "Reducing false positives based on time sequence analysis," Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007), 2007.

[8] H. T. Elshoush and I. M. Osman, "Reducing false positives through fuzzy alert correlation in collaborative intelligent intrusion detection systems ," International Conference on Fuzzy Systems, Jul. 2010.

[9] S. Roschke, F. Cheng, and C. Meinel, "A Flexible and Efficient Alert Correlation Platform for Distributed IDS," Fourth International Conference on Network and System Security, 2010.

[10] G. Tedesco and U. Aickelin, "Real-Time Alert Correlation with Type Graphs," in Proceedings of the 4th International Conference on Information Systems Security, 2008.

[11] F. Valeur, "Real-time intrusion detection alert correlation," Ph.D. dissertation, Univ. Santa Barbaration, 2006.

[12] F. Valeur, G. Vigna, C. Kruegel, and R. a. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 3, Jul. 2004.

Authors profile

Masoud Narimani completed his undergraduate educations in information technology at Payam-Noor University in 2011. Then he passed the bachelor education in information technology security at Maleke-Ashtar University of Technology and then arrived to graduate skills from 2013. He has one paper about using GPU power in accelerating alert correlation algorithms in international conferences. His favorite research fields are: information security management, security evaluation, network security and parallel processing.

Dr. Alireza Nowroozi is assistant professor in Maleke-Ashtar University of Technology. His research studies are mainly focused on IT Security, Crisis Management and Decision Making. He earned his BS in Software Engineering from the Ferdowsi University of Mashhad and his MS in Computer Science from Sharif University of Technology. He holds his PhD in Computer Science in Amirkabir University of Technology. He earned the highest

GPA during MS and PhD education. He stood first in Azad University's MS entrance exam in AI and ranked second in state universities' MS entrance exam in Sharif University of Technology in CS. His PhD and MS students are now active in security evaluation, network security and penetration testing.

Payam Mahdinia completed his undergraduate educations in hardware engineering at Isfahan University in 2010. Then he passed the bachelor education in computer architecture at Isfahan University of Technology and then arrived to graduate skills from 2013. He has three papers about using GPU power in accelerating intrusion detection systems in international conferences. His favorite research fields are: parallel processing, network security and computer architecture.

