# Kargus: a Batched, Parallelizable GPU-Enabled Intrusion Detection System

Muhammad Jamshed, Ji Hyeong Lee, Insu Yun, Sangwoo Moon, Yung Yi and KyoungSoo Park

Department of Electrical Engineering, KAIST

**Abstract.** The rapid growth of the Internet infrastructure has facilitated rise of multiple services that were considered impractical in the past. Unfortunately, the increase in network resources has also allowed more opportunities for widespread botnets to infiltrate target machines. With growing concerns over such attacks, diagnostic tools and network monitoring systems are increasingly being deployed in front of vulnerable server machines that trigger alerts in the wake of attacks. A network intrusion detection system (NIDS) is a popularly used tool that notifies the site administrators of incoming attacks. However NIDS generally do not scale well on faster 10 Gbps bandwidth links. Therefore administrators lean towards a hardware-based solution or turn to farm of IDS monitoring machines. Both alternatives are costly in general.

We introduce Kargus: a software-based IDS framework for a multi-core commodity machine equipped with Graphics Processing Units (GPUs) that scales well on 10 Gbps networks. Kargus outperforms all previous predecessors due to several factors. (i) It exploits parallelism by distributing traffic evenly across each core from the network adapters for analysis. (ii) It uses a batched architecture that reads and processes packets across every sub-module in groups. (iii) In an advent of a flash crowd the IDS opportunistically offloads a proportionate amount of traffic to GPUs to perform attack signature comparisons. Since GPUs are adept at handling multiple packets’ pattern matching tasks in parallel, our batching feature amortizes the overall processing cost. Our evaluation results indicate that a typical machine installed with Kargus shows a performance improvement by a factor of 2.3-6.1 times over the current state-of-art software-based solutions.

Keywords: IDS, PCRE, GPGPU, botnets

## INTRODUCTION

The rapid increase in bandwidth at the edges of the Internet has multiplied the threats posed by botnets that are not showing any signs of recession [3]. Administering network diagnosis in such environments has become more challenging due to the increasing volumes of ingress network traffic. More specifically, traffic monitoring has become an arduous task when dealing with flash crowd as the system tends to drop the surplus network traffic it does not have the resources to analyze.

Site administrators have traditionally used two types of intrusion detection systems: (i) Signature-based IDS solutions [1, 17, 18] rely on deep packet inspection and use attack pattern matching algorithms to detect potential attacks on servers. The system comes with a database of known attack patterns. Each incoming packet is run through the database to check the maliciousness of the incumbent flow. On the flip side, the attack database requires continuous updates as new vulnerabilities are discovered. (ii) Anomaly-based IDS solutions [16] generally rely on abnormal network behavior of flows to detect potential intrusions on target servers. However they are more prone to generating false positives [19]. We feel that the benefits of using signature-based IDS outweigh that of the anomaly-based systems due to the possibilities of generating false alarms if the latter is deployed. Signature based IDS solutions typically depend on a pre-defined rule-set that contains list of patterns and, in some cases, regular expression that span thousands of definitions of known attacks. One of the most widely used rule-set is maintained by Sourcefire Cybersecurity [2] that has become the de facto standard for NIDS signature definitions over the last few years.

Software based NIDS solutions have traditionally performed well on local area networks. However the NIDS face several bottlenecks when they are installed on high bandwidth 10 Gbps links towards the edge of the Internet. Firstly, the network I/O abstraction layer severely throttles the traffic arrival rate. This is because NIDS authors normally use I/O libraries such as Packet Capture (pcap) [4] that are built on top of a large networking stack that prove to be too resource hungry. Secondly, the NIDS normally have huge functional stack trace that cripples the monitoring process each time a packet is passed across all the sub-modules of the system. Thirdly, the NIDS is not designed to fully exploit parallelization that may be introduced for modern machines that host multi-core processors. Fourthly, the software for the NIDS has not originally been written in view of its deployment on 10 Gbps networks and therefore face memory bandwidth throttling effects (for example, due to a large packet structure’s metadata sizes).

IDS vendors and authors have consequently focused more on developing high bandwidth solutions [6–8, 11–13, 15] based on networking hardware customizations that inadvertently increase the cost of the monitoring tools.

MIDeA [20] is the first Snort-based [17] software IDS solution that attempts to resolve some of the deficiencies. It uses PFRING’s [9] Receive-Side Scaling [14] (RSS) based I/O that fairly distributes incoming flows to different cores. Moreover, it offloads processor-intensive pattern matching tasks to multiple GPUs. The Snort framework uses Aho Corasick polynomial time algorithm [5] to detect attack string patterns in packets: a task that can conveniently be parallelized in GPUs if the packets are dispatched in batches. As a result of these optimizations, MIDeA achieves an overall throughput of 7.22 Gbps for 1500 B packets and 1.5 Gbps for 100 B packets. Despite outperforming all previous software-based IDSes, MIDeA lacks a few key features (described in section 2 that can further boost the overall performance of the system).

We present Kargus: a flexible and highly scalable signature-based IDS framework that successfully resolves all the weaknesses we have discussed so far.

## DESIGN

Kargus has been developed while keeping the following features into consideration.

**High Speed Packet I/O:** Kargus employs PacketShader’s [10] efficient packet I/O engine that uses NIC’s multicore aware receive queues with Receive-Side Scaling (RSS) [14]. RSS evenly distributes packets across multiple receive queues by hashing the five fields of the IP/TCP headers: source and destination addresses, transport layer port numbers, and the transport protocol number. Each receive queue in a NIC maps to a single CPU core, and the corresponding CPU core accesses the queues exclusively, eliminating cache bouncing and lock contention that can be caused by shared data structures. The randomness of the RSS hash function ensures that the incoming traffic load is evenly distributed across all cores. The engine delivers packets to the user-space application in batches. This helps the application in amortizing the cost of processing packets (by incurring less context switches between user space and kernel space).

**Function Call Batching:** The Kargus framework is set up so that each core runs its own IDS engine. Each sub-module (preprocessing, flow management, and analysis) within an engine propagates the packets in batches - a concept that is enforced from the packet I/O engine. This reduces the overhead of frequent function calls when compared with packets that are passed individually.

**Single Process/Multi-Threaded Model:** Kargus is based on a single process multiple threaded (running independent analyzing engines) model. Each engine is core-affinitized and processes its own quota of flows that are read from its RSS receive queue. The single process model ultimately assists the IDS framework for fine-grained load balancing where one overly stressed engine can offload task to its sibling engine.

**GPU Capable:** The default batching approach across all sub-modules facilitates efficient opportunistic offloading of packets to GPUs for parallel attack strings pattern matching. The offloading process only activates when the system experiences heavy incoming networking load. Each GPU is managed by a dedicated proxy engine thread that is affinitized to an arbitrary core. All other engines responsible for processing incoming packets enqueue the respective payloads to the proxy threads for evaluation. Unlike MIDeA, the GPU proxy thread is also capable of accepting PCRE evaluation requests.

**Load Balancing:** We have devised an efficient load balancing technique that opportunistically offloads packets for analysis to the GPU proxy threads only if the engine is under heavy load. The load balancer is programmed to decrease the offloading rate once it detects that the GPU is also overloaded with tasks.

By implementing these techniques, we have managed to create a software-based IDS framework that is capable of analyzing at a rate higher than 30 Gbps for all packet sizes. Although the current version of Kargus only monitors HTTP-based web traffic for the time being, it can easily be extended to cover more protocols and can eventually achieve the same functionality as Snort’s.

## EVALUATION

|  |  |
| --- | --- |
| *a) CPU vs GPU – Non-Malicious Packets* | *b) CPU vs GPU – Malicious Packets* |

*Figure 1: Kargus Performance – CPU vs CPU/GPU (Sending Rate: 40 Gbps)*

Our testbed consists of a Kargus NIDS system that is an Intel Xeon E5680 3.33 GHz 12 core (6 cores per NUMA socket) machine with 12 MB L3 cache and 24 GB of RAM. The NIDS machine has two dual-port Intel 82599 10 Gbps network while we also install two NVIDIA GTX 580 GPUs. Each NIC and GPU is installed on separate NUMA nodes.

Our attacker machine is a customized extension of the PacktShader’s packet generator [10] that is capable of interpreting and generating synthetic attack traffic from Snort rules. The attacker machine is an Intel Xeon E5660 2.80 GHz 12 core machine with 12 MB L3 cache and 24 GB of RAM.

We implement our IDS in two modes. The CPU only mode has a total of twelve engines where each instance reads from two network interfaces (with 6 RSS receive queues per NIC). The CPU/GPU mode has ten engine threads that read from two network interfaces each (5 RSS receive queues per NIC). Two engine threads serve as proxies to the GPUs. We evaluate our performance with the following two experiments.

## Synthetic Non-Malicious Traffic

We evaluate Kargus by sending non-malicious traffic at 40 Gbps through our packet generator with varying packet sizes. Figure 1(a) shows the results. Both modes of Kargus clearly outperform MIDeA for 128 and 1514 bytes packets. MIDeA did not report how it fared against packets of remaining sizes. The histogram indicates that the CPU only mode outperforms CPU/GPU setup for lower packet sizes. This is because the host to GPU memory copy bandwidth severely throttles the analyzing rate. The CPU mode shows exceptionally high performance for 64 B packets because the analyzer module (that is scanning for attack patterns) only has to read at most 10 bytes of payload (the Ethernet, IP and TCP headers make up 54 bytes of the packet).

## Synthetic 100% Malicious Traffic

We next test Kargus by sending 100% malicious traffic at line rate with similar settings. Figure 1(b) shows that the CPU/GPU mode performs better than CPU only mode for packets sizes greater than 128 bytes. The analyzing performance progressively degrades with larger attack packet sizes since the analyzing detection engine triggers detailed post-processing (that involves performing remaining Snort rule options evalutions) once the Aho Corasick module confirms presence of the attack packets.

## CONCLUSION

We discuss, Kargus, a highly efficient network intrusion detection system that exploits features that are available in modern multi-core systems such as parrallelization (with the assistance from RSS queues from NICs). Kargus is also capable of using GPUs if they are present in the system which, in fact, enhances the performance as it further parrallelizes the work. In order to fully facilitate GPU framework, the incoming packets to core engines are passed in batches that, in turn, also amortizes the cost of per-packet analysis. Our experiments show that Kargus outperforms state-of-the-art NIDS solution by a factor of almost 2:3. Although Kargus currently only supports HTTP flows, introducing more protocols detection plugins is merely an incremental task.

##### REFERENCES

1. Suricata Intrusion Detection System. http://www.openinfosecfoundation.org/index.php/download-suricata.
2. Sourcefire Cybersecurity, 2012. http://www.sourcefire.com/.
3. Symantec intelligence report: January 2012, 2012. http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&ved=0CF0QFjAG&url=http%3A%2F%2Fwww.symanteccloud.com%2Fmlireport%2FSYMCINT\_2012%\_01\_January\_FINAL-en.pdf&ei=x1h0T9HgCIytiQeGruzjDw&usg=AFQjCNGUE5s5JrqdCffO3jAxbRUOyFOfJg.
4. Tcpdump/libpcap, 2012. http://www.tcpdump.org/.
5. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. Commun. ACM, 18:333–340, June 1975.
6. Z. K. Baker and V. K. Prasanna. Time and area efficient pattern matching on fpgas. In In The Twelfth Annual ACM International Symposium on Field-Programmable Gate Arrays (FPGA 04), pages 223–232. ACM Press, 2004.
7. C. Clark, W. Lee, D. Schimmel, D. Contis, M. Kon, and A. Thomas. A hardware platform for network intrusion detection and prevention. In In Proceedings of the 3rd Workshop on Network Processors and Applications (NP3), February 2004. 178.
8. C. R. Clark and D. E. Schimmel. Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns. In In Proceedings of 13th International Conference on Field Program, pages 956–959, 2003.
9. L. Deri. Improving Passive Packet Capture: Beyond Device Polling. In 4th International System Administration and Network Engineering Conference (SANE), 2004.
10. S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated Software Router. In Proceedings of ACM SIGCOMM, 2010.
11. J. Lee, S. H. Hwang, N. Park, S.-W. Lee, S. Jun, and Y. S. Kim. A high performance nids using fpga-based regular expression matching. In Proceedings of the 2007 ACM symposium on Applied computing, SAC ’07, pages 1187–1191, New York, NY, USA, 2007. ACM.
12. R.-T. Liu, N.-F. Huang, C.-H. Chen, and C.-N. Kao. A fast string-matching algorithm for network processor-based intrusion detection system. ACM Trans. Embed. Comput. Syst., 3:614–633, August 2004.
13. C. R. Meiners, J. Patel, E. Norige, E. Torng, and A. X. Liu. Fast regular expression matching using small tcams for network intrusion detection and prevention systems. In Proceedings of the 19th USENIX conference on Security, USENIX Security’10, pages 8–8, Berkeley, CA, USA, 2010. USENIX Association.
14. Microsoft. Scalable networking: Eliminating the receive processing bottleneckintroducing rss. In WinHEC (White paper), 2004.
15. A. Mitra, W. Najjar, and L. Bhuyan. Compiling pcre to fpga for accelerating snort ids. In Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems, ANCS ’07, pages 127–136, New York, NY, USA, 2007. ACM.
16. Paxson, Vern. Bro: A System For Detecting Network Intruders In Real-Time. Computer Networks, (31):2435–2463, 1999.
17. M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In 13th Systems Administration Conference - LISA, 1999.
18. C. Smutz and A. Stavrou. Ruminate: A Scalable Architecture for Deep Network Analysis (Technical Report). 2010. http://cs.gmu.edu/˜tr-admin/papers/GMU-CS-TR-2010-20.pdf.
19. R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In IEEE Symposium on Security and Privacy, pages 305–316, 2010.
20. G. Vasiliadis, M. Polychronakis, and S. Ioannidis. Midea: a multi-parallel intrusion detection architecture. In Proceedings of the 18th ACM conference on Computer and communications security, CCS ’11, pages 297–308, New York, NY, USA, 2011. ACM.