Leveraging Apache Spark for Real-Time Regex Matching on Bro Log Data

Cadets Sean Deaton, David Brownfield, Leonard Kosta, and Bob Zhu
Advisor: Dr. Suzanne J. Matthews

Motivation

- **Network Intrusion Detection Systems (NIDS)** are critical to maintaining situational awareness on enterprise networks.
- While tools like Bro NIDS are widely used, the Department of Defense (DoD) maintains a set of custom regular expressions (regexes) to additionally scan data for threats.
- To maintain pace with future increases in network traffic volume, the DoD must process log events at a rate of 1,250 events per second and up to 2,500 events per second during brief spikes of network activity.
- **Research Question:** Can we leverage Apache Spark to develop novel algorithms that can perform regex matching on log data at a rate of 2,500 events per second?

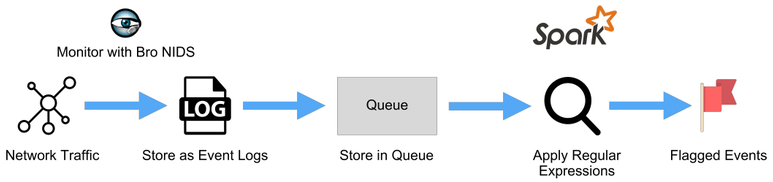

Figure 1: Anomaly Detection Work Flow. Data from network access points is monitored by Bro NIDS. and stored in event logs and in an event queue. The goal is to apply custom regular expressions to events streaming from the queue to detect malicious Uniform Resource Identifiers (URIs) in real-time.

Background

- NIDS on DoD High-Performance Computing (HPC) Systems log gigabytes of events daily, producing event streams that must be further scanned for malware.
- Exploit kit and ransomware authors often develop custom string mutation algorithms to generate malicious Uniform Resource Identifiers (URIs) in HTTP requests.
- The DoD maintains and updates its own custom set of regexes for detecting malicious URIs.
- To instill confidence that the network was not compromised, these regexes are applied to a stream of events that have already been passed through Bro NIDS.
- **Apache Spark** is an open source parallel computing framework that is widely considered the successor to Apache Hadoop. Its advantages over Hadoop include:
  - A speedup over Hadoop of at least 10 times.
  - Support for design flows beyond MapReduce.
  - Enabling speedup on multicore systems.
  - Leveraging Resilient Distributed Datasets (RDDs) to conduct fault-tolerant parallel operations.
  - Supporting lazy evaluation and distributed in-memory processing of intermediates.
  - Allowing a user to specify the number of partitions with which to split input data.

Event ID	Source IP	Dest IP	Method	Host	URI
Ch8Ilv4F	10.0.0.4	192.168.1.48	GET	b.scorecard.com	/ns_site=bloomberg&type=hidden
Clz5TR2B	10.0.0.41	192.168.1.184	HEAD	pontiac2.mil	/Software/SiteStat.xml
C54Ekr28	10.0.0.137	192.168.1.184	HEAD	pontiac2.mil	/Software/SiteStat.xml
CclH3B2u	10.0.0.137	192.168.1.184	GET	pontiac2.mil	/Software/EPOAGENT/Catalog.z
C1D8pnn2	10.0.0.4	192.168.1.3	GET	z.cdn.turner.com	/xslo/cvp.swf?profile=expansion

Figure 2: Excerpt from an event stream. Each event is on a separate line in the NIDS event stream.

Unclassified, Distribution D

Contributions

- We designed a novel parallel algorithm that leverages Apache Spark for parallel regex matching.
- We ran our implementation on ERDC's High Performance Computer, Topaz.
- The work described in this poster is the result of our year-long capstone research project. We (the student authors) wrote all the code, ran all the experiments, and produced all the figures described in this work. Our faculty advisor was a valuable source of mentorship and guidance.

Algorithm and Implementation

- Our algorithm utilizes the Map-Reduce paradigm, which is divided into two distinct phases: Map and Reduce.
- Map Phase: Events are examined in parallel using regexes. Events that match a regex are "flagged" and passed to the combiner with regex info.
- Reduce Phase: A union operation is performed on each event's set of regex information, which is outputted to the user.
- In the example below, we consider n events and 3 partitions. Event 1, 5, and n will trigger our set of regexes

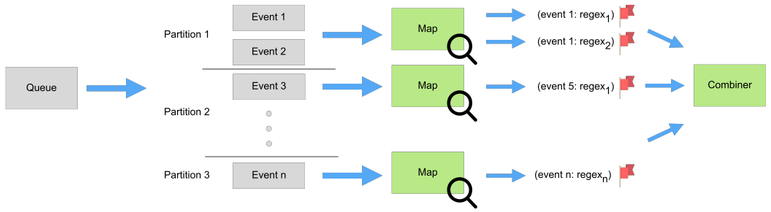

Figure 3: Map Phase. Each mapper receives a partition of events and runs the set of regular expressions on it. If there is a match with a regular expression, the event id is passed to the combiner, along with the information for that match, including the regexID, and fieldID. For simplicity, we represent the information associated with regex 1 as regex1.

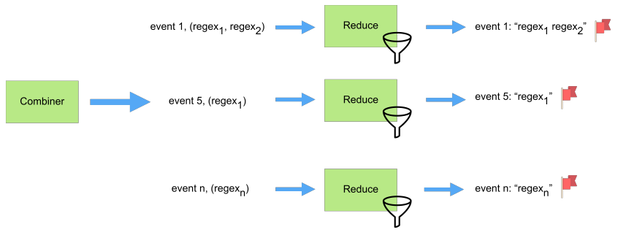

Figure 4: Reduce Phase. Each reducer receives from the combiner an eventID and a set of associated matches. For each event, the reducer simply combines all the matches into a single string.

Experimental Testing

- **Experimental Data:** The DoD provided a dataset consisting of 31 million events collected from Ft. Hood, TX and 569 sample regular expressions.
- **Experimental Platform:** The DoD provided access to Topaz, a DoD HPC Cluster consisting of 3,468 nodes each with 36 cores @ 2.3GHz and 117GB of RAM.
- The DoD cluster is used for many other purposes other than regex matching; as a result, we want to use as few resources as possible to achieve real-time performance (e.g., 2500 events per second).

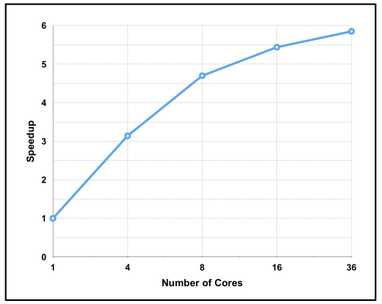

Figure 5: Speedup vs Number of Cores. Increasing the number of the cores increases the speedup of our approach.

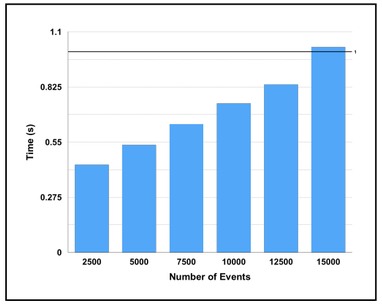

Figure 6: Time vs Number of Events. Our approach can process just under 15,000 events per second using 36 cores to meet the defintion of real-time.

	Number of Cores				
Events	1	4	8	16	36
1250	1.372	0.516	0.418	0.364	0.385
2500	2.557	0.813	0.544	0.470	0.437

Table 1: Timing Events. Average time in seconds to process 1,250 and 2,500 events on n cores.

	Number of Events					
Metric	2500	5000	7500	10000	12500	15000
Time (s)	0.437	0.536	0.640	0.743	0.837	1.024

Table 2: Event Processing Scalability. Average time in seconds to process n events on 36 cores.

- **Coverage Experiments:** We tested our approach on 12,205 random samples of 2,500 events from the Ft. Hood dataset and observed that our approach takes an average of 1.57 seconds to proccess each sample.

Impact & Future Work

- ERDC processes approximately 108 million events per day.
- We are able to process 108 million events within 12 hours, exceeding ERDC's given definition of real-time.
- This will rapdily increase the rate at which ERDC is currently able to process network traffic.
- The project will be delivered to ERDC inside a Docker container, allowing it to be used across Army installations for detecting malicious traffic.
- For improvements, we seek to obtain a set of log events tagged as malcious or benign to perform supervised machine learning and evaluate the correlation between URIs and malicious network traffic.
- We also would like to study how the complexity of the regex can impact load-balancing. Specifically, using the First Fit Decreasing Algorithm to group regexes by peformance time might improve current runtimes.