Firegaze: Processing and Visualizing Firewall Logs in the Cloud

R. van Tonder and W. Visser
Department of Computer Science
University of Stellenbosch, Private Bag X1, Stellenbosch 7600
Tel: +27 21 808 4232, Fax: +27 21 882 9865
email: {rvantonder,visserw}@sun.ac.za

Abstract-This project aims to visualise packet counts filtered by iptables at the network layer, and allows for performing network forensics in a distributed environment. For example, anomalies such as bandwidth spikes and port scans are exposed and quickly identifiable. Naturally, there are a host of tools which already perform this function. The twist with this project is that it should operate on a scalable cloud infrastructure—Nimbula Director is used as a test bed to this end. Intrusion Detection Systems and full-blown Security Information and Event Management (SIEM) solutions have their merits but are often too bulky. Cloud infrastructures rely principally on correctly configured firewalls for network-layer security. As such, Firegaze is a prototype solution which serves as a supplement to network layer security by visualizing firewall activity; it does not perform any analysis, but rather leaves it up to the system administrator to identify anomalous activity. Typically, log files are only needed once an incident occurs, or in the event of system failure. The idea behind Firegaze was to provide a solution for visualizing iptables logs in real-time, or on a historical basis. The challenge of doing this in an environment which scales has influenced the implementation greatly; logs are propagated among nodes in a hierarchical manner, and logs are inserted into a sharded MongoDB database according to a pre-aggregated reports pattern.

Index Terms—cloud, logging, firewall,visualisation

I. INTRODUCTION

Logs provide a wealth of information on system activity, but are often ignored as system administration or development is prioritized in an enterprise environment. There is a growing need to utilize the available information and present it concisely, and to do so on a massive scale.

Log analysis can be decidedly valuable for optimising system performance, reporting, profiling, and security. However, one of the difficult challenges encountered in the field of log analysis is the distributed nature of log-generating components [24]. In the cloud, it is non-trivial to monitor events based on logs which originate from multiple sources. This is especially relevant when one considers the value of logs in a security context: an incident typically takes place on a single node, or on a small subset of nodes, within a network. Nevertheless, the benefits for addressing the problem are clear; Oliner et al. [25] asserts that logs can help to reconstruct events following a security incident, or identify anomalous activity.

In particular, network security in a cloud environment can be strengthened by providing a detailed account of firewall activity, provided that relevant logs are taken into account. Firewall logs contain evidence of anomalous activity, and can be used as forensic evidence during event reconstruction. This project addresses the need for large-scale management, processing, and visualisation of Linux firewall logs; a subset of system logs generated by the well-known iptables application. Nimbula Director [21], a proprietary cloud platform, is chosen to provide basic infrastructure for deployment and testing within a distributed environment.

II. LITERATURE SURVEY

The literature survey highlights works related to scalable log processing, and the use of firewall logs to improve network security.

A. *Log Processing*

There is a growing desire to address the need of handling log analysis in distributed systems and cloud-based environments. Wei et al. [19] advocate the use of a NoSQL solution (MongoDB [5]) for log storage, aggregation, and query purposes. It optimizes border bandwidth distribution among ISPs, and provides network traces based on logs. Aggregation of IP-group traffic is done at 10-minute intervals, using MongoDB's MapReduce capability. MapReduce-based log analysis for system anomaly detection has been explored by Liu et al. [14]. Hadoop's [2] MapReduce framework is used as a basis for log analysis, with separate clusters for data aggregation and analysis.

A few examples in industry are putting these technologies into practice. Loggly [4], a San Fransisco startup company, provides a cloud based "logging as a service" platform which supports a scalable way of managing customer logs. Loggly's servers process and index customer logs, and aims to provide application intelligence and ease of troubleshooting. Apache Solr [1] serves as Loggly's search platform, and MongoDB is used to store aggregation statistics such as log counts. All log visualizations are produced by querying MongoDB, and text searches are directed to Solr. Importantly, logs themselves are persisted on the filesystem, while aggregation results are stored in the database. Both Solr and MongoDB scale horizontally, and so the architecture proves effective. Splunk [22] is another example of a company which indexes customer logs, allowing powerful search functionality and visualizations. While Splunk's architecture is mostly custom-built, it makes use of the MapReduce framework and a time-series database.

An important point should be made that while the MapReduce framework is highly scalable, it is largely a batch-oriented paradigm. For this reason, real-time analytics, monitoring, and alerts may be slower if they depend on the MapReduce model.

Visualization of log statistics is the most important part of conveying the information that system administrators are interested in. Aptly summarized, "a picture is worth a thousand log entries" [15]. Ganglia is an example of a distributed monitoring system for clusters and grids. It uses RRDtool [6] to display effective time-series data pertaining to network traffic, CPU utilization, running processes, and other metrics. Moreover, Loggly's solution supports charting data with Highcharts.

The tools discussed thus far make use of custom visualizations, but all maintain a web front-end to display appropriate charts; these form part of a centralized monitoring solution.

B. *Distributed Network Security*

PSAD (Port Scan Attack Detector) is a Linux tool that performs intrusion detection and log analysis using iptable logs. PSAD operates on the central idea that a lot of data relating to intrusion detection may be gleaned from firewall logs, since log entries maintain fields of almost every aspect of packet headers. PSAD detects anomalous activities that manifest themselves at the network and transport layer [18]. Moreover, PSAD can be configured to send e-mail alerts when a threat is detected. The major limitations with PSAD is that it does not cater for a distributed environment, nor does it provide visualization of iptables logs. Consequently, the wealth of information exposed from fine-grained detail of firewall logs go to waste.

Snort is another popular intrusion detection system (IDS), and interacts with a firewall by applying rulesets. Brennan [8] and Kumar et al. [13] support the idea that IDS such as Snort can be used in a distributed environment. Individual nodes are set up to detect suspicious activity using Snort, and alerts are propagated to centralized nodes on the grid. As with PSAD, this idea does not give an oversight of the entire cluster, nor does it allow inspection of firewall activity by visualization.

While IDS are not obsolete, Gartner [16] makes a valid point that "Firewalls are the most effective defense against cyberintruders" and that enterprises are plagued by false-positives generated by IDS. The emphasis is that investment should be on "firewalls that block attacks, rather than alert administrators to them." Indeed, an IDS should not be a tool that is relied on to fend off attacks, but as a supplement to a correctly configured firewall. It is therefore all the more worthwhile for the system administrator to have an indication of traffic activity at the firewall level, rather than relying on black-box IDS analysis.

III. Design

A. *Overview*

Management, processing, and visualization of firewall logs in a distributed environment is demonstrated by implementing a prototype logging tool, henceforth referred to as Firegaze, on Nimbula Director. This entails making use of scalable technologies and architectures, as the number of nodes may order in the tens of thousands. The goal is to provide a scalable solution which provides an overview of firewall activity on a fine-grained level: counts of data packets associated with IPs and ports that pass through the firewall on a minutely, hourly, daily, weekly, and monthly basis. This information forms a foundation for anomaly detection and log visualisation.

Serving as a supplement to a properly configured firewall, the tool assists the user in identifying anomalies based on packet counts and port destinations. System administrators benefit from charts detailing real-time metrics of firewall traffic, contributing to security intelligence. In addition to real-time updates, the user is also able to query and observe traffic in the past, at varying granularities.

Filtering can be performed on IP addresses via regular expressions, affording flexibility. This allows system administrators to pinpoint precisely the traffic behaviour that they're interested in—something not readily available when unprocessed logs reside on a local file system. As a final feature, a history of raw logs are archived on the distributed file system. The tool is designed for two purposes: to meaningfully persist firewall log counts at scale, and to provide visualisation of log statistics.

B. *Log Storage in the Cloud*

In addressing the persistence of firewall logs, two possibilities present themselves. Namely, logs may be persisted in the Hadoop File System (HDFS) or in MongoDB. While the former possibility scales, it would not be meaningful to persist log counts. In order to generate log counts from HDFS, the raw log files would need to be parsed and processed in a batch-like MapReduce manner. Rather, HDFS is nominated to store raw logs for auditing purposes, but not for processing and visualization purposes. Each node within the cloud stores its logs locally on HDFS.

MongoDB presents an elegant solution when used with prudence. It is infeasible to store raw log lines in MongoDB, firstly because it would incur sizeable writes, and secondly because the raw logs contain redundant fields. Instead, the approach is taken to parse raw log lines and update log counts by relevant fields in MongoDB. Furthermore, MongoDB scales horizontally to meet the needs of distributed systems, making it a suitable solution. In the cloud infrastructure, a single MongoDB is sharded over all nodes in the network (See Figure 1).

C. Log Propagation in the Cloud

While all nodes can theoretically make a connection to MongoDB within the cloud, it is infeasible for each node to write to MongoDB. This approach would generate much overhead and MongoDB would continually have to keep the database consistent. Consequently, it can be expected that performance will suffer greatly. Instead, we opt for an approach which minimizes the amount of connections needed to write to MongoDB, and a form of log propagation is used.

In general terms, cloud architectures are composed of controller and compute nodes. OpenStack is one such example [23]. Controller nodes typically expose cloud application programmer interfaces (APIs) for launching compute instances on compute nodes. In the interest of high availability, these responsibilities can shift if a particular node fails. The cloud infrastructure of Nimbula follows this model, and is such that a hierarchy of node responsibilities exist in three tiers. In reference to Figure 1, site nodes reside at the top of the hierarchy (purple), followed by cluster nodes (blue). These are analogous to controller nodes. Standard nodes, analogous to compute nodes, are labeled in green.

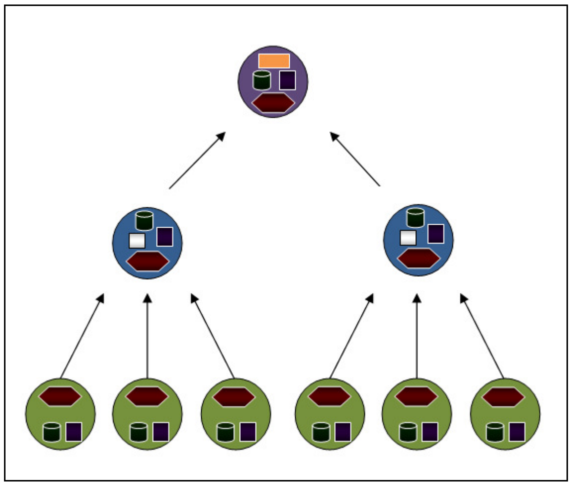
Figure 1: Log propagation architecture

In this configuration, all nodes log local system messages, and only warning logs are propagated to higher tiers in a bottom-up manner. Firegaze makes use of this log propagation model to forward raw iptables logs, with a few modifications. Cluster nodes (in the middle tier) carry the responsibility of parsing, filtering, and inserting local and received firewall logs into MongoDB. A single cluster node suffices for servicing hundreds to thousands of standard node logs, which drastically lowers the amount of connections required to the database. Figure 2 illustrates the logging services we expect on cluster nodes. It maintains a logging daemon, an instance of the sharded MongoDB, an instance of the shared HDFS, and a log handler. Standard nodes run the same services, but do not process logs.

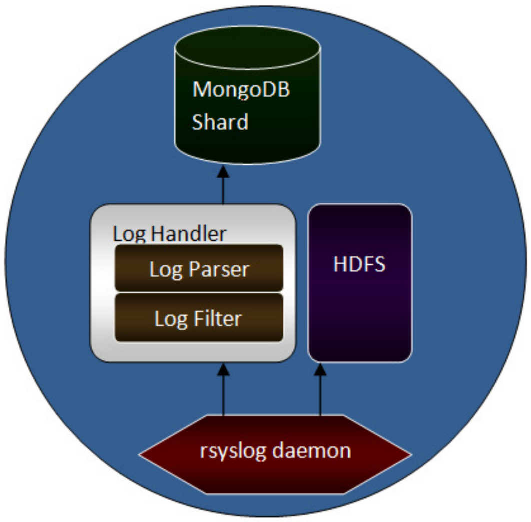

Figure 2: Cluster node services

Site nodes (at the top of the hierarchy) are relieved from performing the role of processing logs, since they are responsible for critical administrative operations in the cloud. Therefore, site nodes propagate their iptables logs to an appropriate cluster node. Site nodes run the same services as standard nodes, with the addition of providing a web front-end for Firegaze (see Figure 3). The web front-end is composed of multiple views, with tabular and graphical representations of firewall metrics.

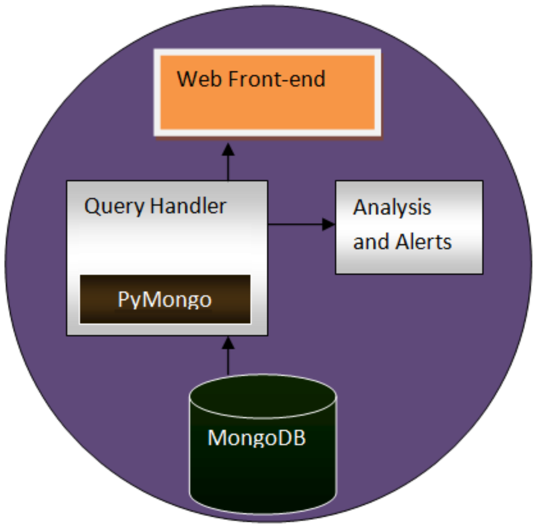

Figure 3: Site node front-end services

In summary, all nodes in the cloud (including standard nodes) share a single, sharded MongoDB instance. However, cluster nodes are the only entities which ever write to MongoDB, and site nodes are the only entities which query MongoDB and visualises data.

D. Data Information Model and Visualisation

The design of log processing and storage enables the visualisation of log attributes, the most pertinent of which include

- IPs with the highest number of packet counts

- IPs with the highest number of ports in use
- Packet counts for specific IPs by destination and source port
- The number of ports in use for specific IPs.

Additionally, Firegaze allows the user to query these criteria by varying time granularities and direction. Conceptually, queries select relevant dimensions of the data model presented in Figure 4.

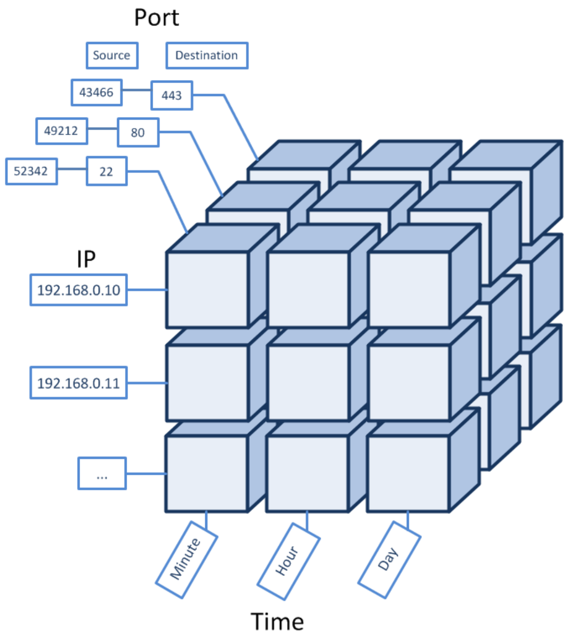

Figure 4: Dimensional breakdown of log data

IV. Implementation

The implementation comprises a front-end which issues queries and performs visualisation, as well as a back-end which maintains log collection, processing, and serving queries. The back-end is implemented in Python, and interacts with MongoDB through the PyMongo API. Rsyslog is the logging daemon which runs on each node, and performs log forwarding. The front-end is written in HTML, PHP, and JavaScript. Data visualization is performed by Highcharts.

A. Back-end

Configuration files for Rsyslog are modified for each node to allow the propagation of iptables log messages, which are identified by a string tag. All packets passing through the ethernet interfaces of the nodes are assigned this tag by additional rules in the INPUT and OUTPUT chains of iptables. Logs received at the cluster node are sent to a named pipe, at which point logs are processed by a Python module. This module strips unnecessary data, extracts the relevant fields, and writes to the appropriate documents in the database.

Database document design forms a critical part of the solution, and MongoDB affords a novel way to construct a document schema according to a pre-aggregated reports design pattern, which lends itself particularly well to storing log counts. To elaborate further, MongoDB is a non-relational, document store database. In discussing the construction of database documents, consider the following raw log:

```
Nov 1 15:25:15 user kernel: [16479.817936] FWLOG=OUT
IN= OUT=eth0 SRC=10.0.2.15 DST=146.232.20.10
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF
PROTO=UDP SPT=2666 DPT=53 LEN=40
```

We are interested in storing counts for important fields, for example:

- FWLOG, the direction of the packet
- SRC, the source IP address
- DST, the destination IP address
- SPT, the source port
- DPT, the destination port,
- PROTO, the protocol

With pre-aggregated reports, attributes in the documents are preemptively generated for varying time granularities, whose counts are incremented as logs are processed. The incentive for doing so is to maximize performance for queries on a per-document basis.

In MongoDB, multiple documents form a collection. Consider that we have a collection which stores incoming packet counts for all IPs at an hour and minute level granularity. In the collection, there exists one document per unique destination IP (DST) per day. Thus, the document key is composed jointly of the DST, and the current day. In this document, we wish to record the packet counts according to varying destination ports (DPT), and source IPs (SRC). This is done via nested attributes for every hour and minute of the day. Figure 5 presents such a document.

```
{ "IP" : "192.168.1.10",
  "_id" : "192.168.1.10/20121023",
  "date" : ISODate("2012-10-23T00:00:00Z"),
  "hour" : { "13" : { "ports" : { "total" : 151,
                                  "443" : 151 },
                      "srcs" : { "197-142-111-2" : 151 } } },
  "minute" : { "13" : { "57" : { "ports" : { "total" : 99,
                                             "443" : 99 },
                      "srcs" : { "197-142-111-2" : 99 } },
                        "59" : { "ports" : { "total" : 52,
                                             "443" : 52 },
                      "srcs" : { "197-142-111-2" : 52 } } } }
}
```

Figure 5: MongoDB document schema for iptables logs

For example, from the document in Figure 5, we can derive that 99 packets were received by 192.168.1.10 on port 443 from a single source 197.142.111.2, at 13:57. Importantly, if an incoming log line is parsed which matches the document key, the count of the destination packet is incremented via an upsert operation.

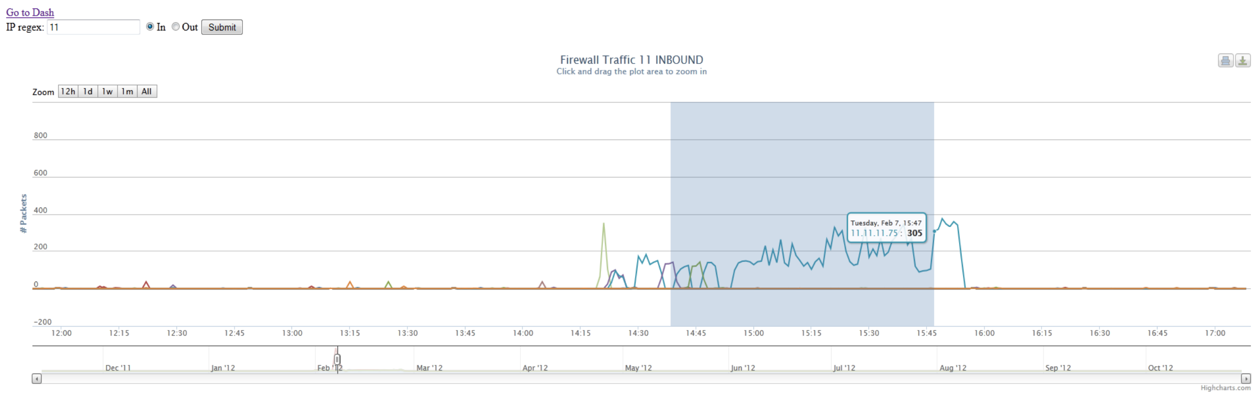

Figure 6: Firegaze visualisation of anomalous activity

Note that the minute attribute is split into hourly fields for optimization. Due to MongoDB's implementation, which stores values sequentially, supplying an hour field allows it to skip over a maximum of 23 values when updating a minute value, as opposed to a maximum of 1439 values if we stored a separate field for each minute of the day.

Firegaze extends the idea of the document design to cater for monthly counts, and out-going packets. Documents are therefore segregated into four collections, which contain incoming and outgoing packet counts for daily and monthly time granularities. Separate collections allow for querying fewer documents when visualising metrics.

B. *Front-end*

The web-based front-end provides tabular representations of packet and port counts, as well as a dynamic graph which plots metrics at varying granularities (enabled by zooming). This functionality is made available by Highcharts, a Javascript charting engine, in conjunction with asynchronous loading of data points via queries to MongoDB.

Asynchronous loading allows for a flexible, efficient way to view millions of data points over varying time resolutions. For example, it is infeasible to plot data points at a minute-level granularity for a time-span of a month, as this would require in excess of 2.5 million data points to be rendered on a single chart. However, by imposing resolution constraints, the visualisation technique afforded by asynchronous loading makes viewing millions of data points manageable. By zooming and panning across a chart in Firegaze, a user can view

- Minutely data for a time-span of two days
- Hourly data for a time-span of two weeks
- Daily data for up to two years

V. Evaluation

A. *Application in the Cloud*

As a prototype solution, results on the cloud proved promising. The test system consisted of 5 nodes running Nimbula Director 2.0. The nodes operated in a restricted IP space on Amazon EC2. After configuration, the back-end system performed reliably without incident for two weeks.

The resource footprint encountered during this time revealed significant insights. Namely, it is very effective to store log counts and metrics in a compressed form in MongoDB, as opposed to analyzing raw log formats.

Quantitatively, raw IP log generation averaged about 200 MB per hour per node, equating to 5 GB per day. Over a period of two weeks and 5 nodes, this implies a size of approximately 350 GB of logs. However, because only log counts are stored, this data becomes massively compressed after processing: the total MongoDB size converted to BSON format is roughly 20 MB in size. This small size is due in part to the fact that only 5 significant IPs were monitored, with low interaction from the Internet in general. However, it should be expected that the high ratio of compression is maintained in proportion to the amount of bandwidth across interfaces, and not in proportion to the number of unique IPs.

In terms of limitations, consideration should be given to the bandwidth usage when propagating logs. While Rsyslog allows compression of logs, compression on a single log line may not justify incurring CPU overhead. With the figures above, a single node would use around 0.5 Mbps, and at first impression, this doesn't seem like much on a 100 Mbps connection. However, it is expected that the bandwidth requirement will increase as more nodes are added to the network. One method of mitigating this is to filter logging of known high-bandwidth ports such as those used by Rsyslog and MongoDB.

B. *Application for finding Anomalies*

Firegaze performed admirably with logs of known incidents made available by The Honeynet Project. One of the scans [7] were used for this purpose. IPs of highest threat could be clearly identified, as well as the ports that lead to compromise of certain hosts. For example, the graph in Figure 6 reveals the date and nature of compromise on one such host.

Anomalous activity was also identified while running Firegaze in real-time on a single EC2 node. It was observed that at one point a brute-force SSH (Secure Shell) attack bombarded port 22 for approximately 3 hours. Similar attempts were found to originate from various IP sources at different times.

VI. Conclusion

Firegaze demonstrates the value that can be derived from simple iptables logs at a relatively low cost. It presents a strong argument for processing of logs whereby counts are stored in a compressed manner in a distributed, cloud environment. Efficient querying across varying time granularities is afforded by an appropriate document schema. While time-series data is, and will continue to be, a challenging form of data, it was demonstrated that it is worthwhile processing and visualising firewall logs. Further use of Firegaze as a forensic tool is also effective, as it can assist in identifying and visualising network activity in response to incidents.

VII. References

[1] Apache Solr. lucene.apache.org/solr/. Accessed: 5/6/2013.

[2] Hadoop. hadoop.apache.org/. Accessed: 5/6/2013.

[3] Highcharts. http://www.highcharts.com. Accessed: 5/6/2013.

[4] Loggly. www.loggly.com. Accessed: 5/6/2013.

[5] MongoDB. www.mongodb.org/. Accessed: 5/6/2013.

[6] RRDtool. oss.oetiker.ch/rrdtool/. Accessed: 5/6/2013.

[7] Scan of the Month, Honeynet. http://old.honeynet.org/scans/scan30/. Accessed: 5/6/2013.

[8] Michael P. Brennan. *Using Snort for a Distributed Intrusion Detection System*. SANS Institute, 2002.

[9] Nathan Einwechter. *An Introduction to Distributed Intrusion Detection Systems*. Symantec, 2001.

[10] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *Data Mining, 2009. ICDM '09. Ninth IEEE International Conference on*, pages 149–158, dec. 2009.

[11] T. Katic and P. Pale. Optimization of firewall rules. In *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on*, pages 685–690, june 2007.

[12] R. Khattak, S. Bano, S. Hussain, and Z. Anwar. Dofur: Ddos forensics using mapreduce. *In Frontiers of Information Technology (FIT), 2011*, pages 117 – 120, dec. 2011.

[13] M. Kumar, M. Hanumanthappa, and T.V.S. Kumar. Intrusion Detection System for Grid Computing Using SNORT. *In Computing, Communication and Applications (ICCCA), 2012 International Conference on*, pages 1 – 6, feb. 2012.

[14] Yan Liu, Wei Pan, Ning Cao, and Guangwei Qiao. System Anomaly Detection in Distributed Systems through MapReduce-Based Log Analysis. In *Advanced Computer Theory and Engineering (ICACTE)*, 2010 3rd International Conference on, volume 6, pages V6–410 – V6–413, aug. 2010.

[15] Raffael Marty. *Applied Security Visualization.* Addison Wesley Professional, 2008.

[16] Michael S. Mimoso. Gartner declares IDS obsolete by 2005. http://searchsecurity.techtarget.com/news/905961/Gartner-declares-IDS-obsolete-by-2005, 2003. Accessed: 5/6/2013.

[17] MongoDB. Pre-Aggregated Reports. http://docs.mongodb.org/manual/use-cases/pre-aggregated-reports/, 2012. Accessed: 5/6/2013.

[18] Michael Rash. *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort*. No Starch Press, 2007.

[19] Jianwen Wei, Yusu Zhao, Kaida Jiang, Rui Xie, and Yaohui Jin. Analysis Farm: A Cloud-Based Scalable Aggregation and Query Platform for Network Log Analysis. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 354–359, dec. 2011.

[20] Robert Winding, Timothy Wright, and Michael Chapple. System anomaly detection: Mining firewall logs. In *Securecomm and Workshops, 2006*, pages 1–5, sept. 2006.

[21] Nimbula Director. http://nimbula.com/product/. Accessed: 5/6/2013

[22] Splunk. http://www.splunk.com/. Accessed: 5/6/2013

[23] OpenStack. http://www.openstack.org/. Accessed: 5/6/2013

[24] Rabkin, A. and Randy, K. Chukwa: A system for reliable large-scale log collection. *USENIX Conference on Large Installation System Administration,* pages 1–15, 2010.

[25] Oliner, A., Ganapathi A., and Wei X. Advances and Challenges in Log Analysis. In *Communications of the ACM*, volume 55, pages 55–61, feb. 2012.

Rijnard van Tonder received his Honours degree in Computer Science in 2012 from the University of Stellenbosch and is presently studying towards his Master of Engineering degree at the same institution. His research interests include Information Security, including mobile, web application, and embedded device security.

Willem Visser holds a PhD in Computer Science from the University of Manchester (1998) and is currently a Full Professor at Stellenbosch University. He is on the Editorial Board of ACM Transactions on Software Engineering and Methodology and on the steering committee of the Automated Software Engineering conference.