Cloud Application Logging for Forensics

Raffael Marty
Loggly, Inc.
78 First Street
San Francisco, CA 94105
rmarty@loggly.com

ABSTRACT

Logs are one of the most important pieces of analytical data in a cloud-based service infrastructure. At any point in time, service owners and operators need to understand the status of each infrastructure component for fault monitoring, to assess feature usage, and to monitor business processes. Application developers, as well as security personnel, need access to historic information for debugging and forensic investigations.

This paper discusses a logging framework and guidelines that provide a proactive approach to logging to ensure that the data needed for forensic investigations has been generated and collected. The standardized framework eliminates the need for logging stakeholders to reinvent their own standards. These guidelines make sure that critical information associated with cloud infrastructure and software as a service (SaaS) use-cases are collected as part of a defense in depth strategy. In addition, they ensure that log consumers can effectively and easily analyze, process, and correlate the emitted log records. The theoretical foundations are emphasized in the second part of the paper that covers the implementation of the framework in an example SaaS offering running on a public cloud service.

While the framework is targeted towards and requires the buy-in from application developers, the data collected is critical to enable comprehensive forensic investigations. In addition, it helps IT architects and technical evaluators of logging architectures build a business oriented logging framework.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Log Forensics

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

Keywords

cloud, logging, computer forensics, software as a service, logging guidelines

1. INTRODUCTION

The "cloud"[5, 11] is increasingly used to deploy and run end-user services, also known as software as a service (SaaS)[28]. Running any application requires insight into each infrastructure layer for various technical, security, and business reasons. This section outlines some of these problems and use-cases that can benefit from log analysis and management. If we look at the software development life cycle, the use-cases surface in the following order:

- Debugging and Forensics
- Fault monitoring
- Troubleshooting
- Feature usage
- Performance monitoring
- Security / incident detection
- Regulatory and standards compliance

Each of these use-cases can leverage log analysis to either completely solve or at least help drastically speed up and simplify the solution to the use-case.

The rest of this paper is organized as follows: In Section 2 we discuss the challenges associated with logging in cloud-based application infrastructures. Section 3 shows how these challenges can be addressed by a logging architecture. In section 4 we will see that a logging architecture alone is not enough. It needs to be accompanied by use-case driven logging guidelines. The second part of this paper (Section 5) covers a references setup of a cloud-based application that shows how logging was architected and implemented throughout the infrastructure layers.

2. LOG ANALYSIS CHALLENGES

If log analysis is the solution to many of our needs in cloud application development and delivery, we need to have a closer look at the challenges that are associated with it.

The following is a list of challenges associated with cloud-based log analysis and forensics:

- Decentralization of logs
- Volatility of logs

- Multiple tiers and layers
- Archival and retention
- Accessibility of logs
- Non existence of logs
- Absence of critical information in logs
- Non compatible / random log formats

A cloud-based application stores logs on multiple servers and in multiple log files. The volatile nature of these resources[1] causes log files to be available only for a certain period of time. Each layer in the cloud application stack generates logs, the network, the operating system, the applications, databases, network services, etc. Once logs are collected, they need to be kept around for a specific time either for regulatory reasons or to support forensic investigations. We need to make the logs available to a number of constituencies; application developers, system administrators, security analysts, etc. They all need access, but only to a subset and not always all of the logs. Platform as a service (PaaS) providers often do not make logs available to their platform users at all. This can be a significant problem when trying to analyze application problems. For example, Amazon[5] does not make the load balancer logs available to their users. And finally, critical components cannot or are not instrumented correctly to generate the logs necessary to answer specific questions. Even if logs are available, they come in all kinds of different formats that are often hard to process and analyze.

The first five challenges can be solved through log management. The remaining three are more intrinsic problems and have to be addressed through defining logging guidelines and standards[2].

2.1 Log Management

Solving the cloud logging problems outlined in the last section requires a log management solution or architecture to support the following list of features:

- Centralization of all logs
- Scalable log storage
- Fast data access and retrieval
- Support for any log format
- Running data analysis jobs (e.g., map reduce[16])
- Retention of log records
- Archival of old logs and restoring on demand
- Segregated data access through access control
- Preservation of log integrity
- Audit trail for access to logs

These requirements match up with the challenges defined in the last section. However, they do not address the last three challenges of missing and non standardized log records.

[1]For example, if machines are pegging at a very high load, new machines can be booted up or machines are terminated if they are not needed anymore without prior warning.

[2]Note that in some cases, it is not possible to change anything about the logging behavior as we cannot control the code of third-party applications.

2.2 Log Records

What happens if there are no common guidelines or standards defined for logging? In a lot of cases, application developers do not log much. Sometimes, when they do log, the log records are incomplete, as the following example illustrates:

```
Mar 16 08:09:58 kernel: [    0.000000]
Normal    1048576 ->  1048576
```

There is not much information in this log to determine what actually happened and what is `Normal`?

A general rule of thumb states that a log record should be both understandable by a human and easily machine processable. This also means that every log entry should, if possible, log *what* happened, *when* it happened, *who* triggered the event, and *why* it happened. We will later discuss in more detail what these rules mean and how good logging guidelines cover these requirements (see Sections 4.2 and 4.3).

In the next section we will discuss how we need to instrument our infrastructure to collect all the logs. After that we will see how logging guidelines help address issues related to missing and incomplete log records.

3. LOGGING ARCHITECTURE

A log management system is the basis for enabling log analysis and solving the goals introduced in the previous sections. Setting up a logging framework involves the following steps:

- Enable logging in each infrastructure and application component
- Setup and configure log transport
- Tune logging configurations

3.1 Enable Logging

As a first step, we need to enable logging on all infrastructure components that we need to collect logs from. Note that this might sound straight forward, but it is not always easy to do so. Operating systems are mostly simple to configure. In the case of UNIX, syslog[31] is generally already setup and logs can be found in `/var/log`. The hard part with OS logs is tuning. For example, how do you configure the logging of password changes on a UNIX system[22]? Logging in databases is a level harder than logging in operating systems. Configuration can be very tricky and complicated. For example, Oracle[24] has at least three different logging mechanisms. Each of them has their own sets of features, advantages, and disadvantages. It gets worse though; logging from within your applications is most likely non-existent. Or if it exists, it is likely not configured the way your use-cases demand; log records are likely missing and the existing log records are missing critical information.

3.2 Log Transport

Setting up log transport covers issues related to how logs are transfered from the sources to a central log collector. Here are issues to consider when setting up the infrastructure:

- Synchronized clocks across components

- Reliable transport protocol
- Compressed communication to preserve bandwidth
- Encrypted transport to preserve confidentiality and integrity

3.3 Log Tuning

Log data is now centralized and we have to tune log sources to make sure we get the right *type* of logs and the right *details* collected. Each logging component needs to be visited and tuned based on the use-cases. Some things to think about are where to collect individual logs, what logs to store in the same place, and whether to collect certain log records at all. For example, if you are running an Apache Web server, do you collect all the log records in the same file; all the media file access, the errors, and regular accesses? Or are you going to disregard some log records?

Depending on the use-cases you might need to log additional details in the log records. For example, in Apache it is possible to log the processing time for each request. That way, it is possible to identify performance degradations by monitoring how long Apache takes to process a request.

4. LOGGING GUIDELINES

To address the challenges associated with the information in log records, we need to establish a set of guidelines and we need to have our applications instrumented to follow these guidelines. These guidelines were developed based on existing logging standards and research conducted at a number of log management companies[4, 15, 20, 30, 33].

4.1 When to Log

When do applications generate log records? Making the decision when to write log records needs to be driven by use-cases. These use-cases in cloud applications surface in four areas:

- Business relevant logging
- Operations based logging
- Security (forensics) related logging
- Regulatory and standards mandates

As a rule of thumb, at every return call in an application, the status should be logged, whether success or failure. That way errors are logged and activity throughout the application can be tracked.

4.1.1 *Business*

Business relevant logging covers features used and business metrics being tracked. Tracking features in a cloud application is extremely crucial for product management. It helps not only determine what features are currently used, it can also be used to make informed decisions about the future direction of the product. Other business metrics that you want to log in a cloud application are outlined in [8].

Monitoring service level agreements (SLAs) fall under the topic of business relevant logging as well. Although some of the metrics are more of operational origin, such as application latencies.

4.1.2 *Operational*

Operational logging should be implemented for the following instances:

- Errors are problems that impact a single application user and not the entire platform.
- Critical conditions are situations that impacts all users of the application. They demand immediate attention[3].
- System and application start, stop, and restart. Each of these events could indicate a possible problem. There is always a reason why a machine stopped or was restarted.
- Changes to objects track problems and attribute changes to an activity. Objects are entities in the application, such as users, invoices, or goods. Other examples of changes that should be logged are:
  - Installation of a new application (generally logged on the operating system level).
  - Configuration change[4].
  - Logging program code updates enable attribution of changes to developers.
  - Backup runs need to be logged to audit successful or failed backups.
  - Audit of log access (especially change attempts).

4.1.3 *Security*

Security logging in cloud application is concerned with authentication and authorization, as well as forensics support[5]. In addition to these three cases, security tools (e.g., intrusion detection or prevention system or anti virus tools) will log all kinds of other security-related issues, such as attempted attacks or the detection of a virus on a system. Cloud-applications should focus on the following use-cases:

- Login / logout (local and remote)
- Password changes / authorization changes
- Failed resource access (denied authorization)
- All activity executed by a privileged account

Privileged accounts, admins, or root users are the ones that have control of a system or application. They have privileges to change most of the parameters in the application. It therefore is crucial for security purposes to monitor very closely what these accounts are doing[6].

4.1.4 *Compliance*

Compliance and regulatory demands are one more group of use-cases that demand logging. The difference the other use-cases is that it is often required by law or by business partners to comply with these regulations. For example, the payment card industry's data security standard (PCI DSS[25]) demands a set of actions with regards to logging (see Section 10 of PCI DSS). The interesting part about the PCI DSS is that it demands that someone reviews the

[3]Exceptions should be logged automatically through the application framework.

[4]For example a reconfiguration of the logging setup is important to determine why specific log entries are not logged anymore.

[5]Note that any type of log can be important for forensics, not just security logs.

[6]Note also that this has an interesting effect on what user should be used on a daily basis. Normal activity should not be executed with a privileged account!

logs and not just that they are generated. Note that most of the regulations and standards will cover use-cases that we discussed earlier in this section. For example logging privileged activity is a central piece of any regulatory logging effort.

4.2 What to Log

We are now leaving the high-level use-cases and infrastructure setup to dive into the individual log records. How does an individual record have to look?

At a minimum, the following fields need to be present in every log record: Timestamp, Application, User, Session ID, Severity, Reason, Categorization. These fields help answer the questions: when, what, who, and why. Furthermore, they are responsible for providing all the information demanded by our use-cases.

A *timestamp* is necessary to identify when a log record or the recorded event happened. Timestamps are logged in a standard format[18]. The *application* field identifies the producer of the log entry. A *user* field is necessary to identify which user has triggered an activity. Use unique user names or IDs to distinguish users from each other. A *session ID* helps track a single request across different applications and tiers. The challenge is to share the same ID across all components. A *severity* is logged to filter logs based on their importance. A severity system needs to be established. For example: debug, info, warn, error, and crit. The same schema should be used across all applications and tiers. A *reason* is often necessary to identify why something has happened. For example, access was denied due to insufficient privileges or a wrong password. The reason identifies why. As a last set of mandatory field, *category* or taxonomy fields should be logged.

Categorization is a method commonly used to augment information in log records to allow addressing similar events in a common way. This is highly useful in, for example, reporting. Think about a report that shows all failed logins. One could try to build a really complicated search pattern that finds failed login events across all kinds of different applications[7] or one could use a common category field to address all those records.

4.3 How to log

What fields to log in a log record is the first piece in the logging puzzle. The next piece we need is a *syntax*. The basis for a syntax is rooted in *normalization*, which is the process of taking a log entry and identifying what each of the pieces represent. For example, to report on the top users accessing a system we need to know which part of each log record represents the user name. The problems that can be solved with normalized data are also called *structured data analysis*. Sometimes additional processes are classified under normalization. For example, normalizing numerical values to fall in a predefined range can be seen as normalization. There are a number of problems with normalization[21]. Especially if the log entries are not self-descriptive. More on that in Section 4.3.

The following is a syntax recommendation that is based on standards work and the study of many existing logging standards (e.g., [9, 17, 29, 35]). Common event expression (CEE)[10] is a new standard that is going to be based on the following syntax:

```
time=2010-05-13 13:03:47.123231PDT,
session_id=08BaswoAAQgAADVDG3IAAAAD,
severity=ERROR,user=pixlcloud_zrlram,
object=customer,action=delete,status=failure,
reason=does not exist
```

There are a couple of important properties to note about the example log record:

First, each field is represented as a *key-value* pair. This is the most important property that every records follows. It makes the log files easy to parse by a consumer and helps with interpreting the log records. Also note that all of the field names are lower case and do not contain any punctuations. You could use underscores for legibility.

Second, the log record uses three fields to establish a categorization schema or taxonomy: object, action, and status. Each log record is assigned exactly one value for each of these fields. The category entries should be established upfront for a given environment. Standards like CEE are trying to standardize an overarching taxonomy that can be adopted by all log producers. Establishing a taxonomy needs to be based on the use-cases and allow for classifying log records in an easy way. For example, by using the object value of 'customer', one can filter by customer related events. Or by querying for status=failure, it is possible to search for all failure indicating log records across all of the infrastructure logs.

These are recommendations for how log entries should be structured. To define a complete syntax, issues like encoding have to be addressed. For the scope of this paper, those issues are left to a standard like CEE[10].

4.4 Don't forget the infrastructure

We talked a lot about application logging needs. However, do not forget the infrastructure. Infrastructure logs can provide very useful context for application logs. Some examples are firewalls, intrusion detection and prevention systems, or web content filter logs which can help identify why an application request was not completed. Either of these devices might have blocked or altered the request. Other examples are load balancers that rewrite Web requests or modify them. Additionally, infrastructure issues, such as high latencies might be related to overloaded machines, materializing in logs kept on the operating systems of the Web servers.

There are many more infrastructure components that can be used to correlate against application behavior.

5. REFERENCE SETUP

In the first part of this paper, we have covered the theoretical and architectural basis for cloud application log management. The second part of this paper discusses how we have implemented a application logging infrastructure at a software as a service (SaaS) company. This section presents a number of tips and practical considerations that help overall logging use-cases but most importantly support and enable forensic analysis in case of an incident.

Part of the SaaS architecture that we are using here is a traditional three-tiered setup that is deployed on top of the Amazon AWS cloud. The following is an overview of the application components and a list of activities that are logged in each component:

[7]Each application logs failed logins in completely different formats[34].

- *Django[13]*: Object changes, authentication and authorization, exceptions, errors, feature usage
- *JavaScript*: AJAX requests, feature usage
- *Apache*: User requests (successes and attempts)
- *MySQL*: Database activity
- *Operating system*: System status, operational failures
- *Java Backend*: Exceptions, errors, performance

In the following, we are going to briefly touch upon each of the components and talk in more detail about what it means to log in these components and what some of the pitfalls were that we ran into.

5.1 Django

Django assumes that developers are using the standard python logging libraries[26]. There is no additional support for logging built into Django. Due to this lack of intrinsic logging support we had to implement our own logging solution for Django and instrument Django, or more precisely, the Django authentication methods to write log entries[14]. In addition, we wrote a small logging library that can be included in any code. Once included, it exports logging calls for each severity level, such as `debug()`, `error()`, `info()`, `warn()`, `crit()`, and `feature()` The first five calls all work similar; they require a set of key-value pairs. For example:

```
error({'object':'customer','action':'delete',
'reason':'does not exists','id':'22'})
```

The corresponding log entry looks as follows:

```
2010 Jan 28 13:03:47 127.0.0.1 severity=ERROR,
user=pixlcloud_zrlram,object=customer,action=delete,
status=failure,reason=does not exist,id=22,
request_id=dxlsEwqgOxAAABrrhZgAAAAB
```

The extra key-values in the log record are automatically added to the log entries by our logging library without burdening the developer to explicitly include them. The unique_id is extracted from the HTTP request object. This ID is unique for each user request and is used on each application tier to allow correction of messages. The user is also extracted through the request object. The severity is included automatically based on the logging command. These different levels of logging calls (i.e., severities) are used to filter messages, and also to log debug messages only in development. In production, there is a configuration setting that turns debug logging off.

Note that we are trying to always log the category fields: an object, an action, a status, and if possible a reason. This enables us to do very powerful queries on the logs, like looking for specific objects or finding all failed calls.

In addition to the regular logging calls, we implemented a separate call to log feature usage. This goes back to the use-cases where we are interested in how much each of our features in the product is used.

5.2 JavaScript

Logging in JavaScript is not very developer friendly. The problem is that the logs end up on the client side. They cannot be collected in a server log to correlate them with other log records. Some of the service's features are triggered solely through Ajax[2] calls and we would like to log them on the server along with the other features. We therefore built a little logging library that can be included in the HTML code. Calling the library results in an Ajax call to an end point on the server that will log the message which is passed as a payload to the call. In order to not spawn too many HTTP requests, the library can be used in batch mode to bundle multiple log records into a single call.

5.3 Apache

Our Apache logging setup is based on Apache's defaults. However, we tuned a number of parameters in order to enable some of our use-cases. To satisfy our logging guidelines, we need to collect a timestamp, the originating server, the URL accessed, the session ID from the application, and the HTTP return code to identify what the server has done with the request.

The motivation for some of these fields you find in the first part of this paper, for example the session ID. Here is a sample log entry that shows how our Apache logs look. Note specifically the last two fields:

```
76.191.189.15 - - [29/Jan/2010:11:15:54 -0800]
  "GET / HTTP/1.1" 200 3874 "http://pixlcloud.service.ch/"
  "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_2; en-us)
  AppleWebKit/531.21.8 (KHTML, like Gecko) Version/4.0.4
  Safari/531.21.10" duvpqQqgOxAAABruAPYAAAAE
```

5.3.1 Apache Configuration

This section is going to outline how we configured our Apache instances. The first step is to configure the *LogFormat* to contain the extra information that we are interested in. The item we added is `{%UNIQUE_ID}e`. The former adds a unique ID into every request and the latter logs the latency for every request:

```
LogFormat "%h %l %u %t \"%r\" %>s %b
\"%{Referer}i\" \"%{User-Agent}i\"
%{UNIQUE_ID}e" service
```

Make sure you enable the `mod_unique_id`[3] module such that Apache includes a session ID in every log entry.

In a next instance, we turned off an annoying log record that you will see if you have Apache running with sub processes. When an Apache server manages its child processes, it needs a way to wake up those processes to handle them new connections. Each of these actions creates a log entry. We are not at all interested in those. Here is the Apache logging configuration we ended up with:

```
SetEnvIf Remote_Addr "127\.0\.0\.1" loopback=1
SetEnvIf loopback 1 accesslog

CustomLog /var/log/apache2/access.log service env=!accesslog
```

5.3.2 Load Balancing

In our infrastructure, we are using a load balancer, which turned out to significantly impact the Apache logs. Load balancers make requests look like they came from the load balancer's IP; the client IP is logged wrong, making it impossible to track requests back to the actual source and correlate other log recrods with it. An easy fix would be to change the LogFormat statement to use the `X-Forwarded_For` field instead of `%h`. However, this won't always work. There are two Apache modules that are meant to address this issue: mod_remoteip and mod_rpaf. Both of these modules allow us to keep the original LogFormat statement in the Apache

configuration. The modules replace the remote ip (%h) field with the value of the X-Forwarded-For header in the HTTP request if the request came from a load balancer. We ended up using mod_rpaf with the following configuration:

```
LoadModule rpaf_module mod_rpaf.so
RPAFenable On
RPAFsethostname On
RPAFproxy_ips 10.162.65.208 10.160.137.226
```

This worked really well, until we realized that on Amazon AWS, the load balancer constantly changes. After defining a very long chain of RPAFproxy_ips, we started looking into another solution, which was to patch mod_rpaf to work as desired[23].

5.4 MySQL

Our MySQL setup is a such that we are using Amazon's Relational Database Service[7]. The problem with this approach being that we do not get MySQL logs. We are looking into configuring MySQL to send logs to a separate database table and then exporting the information from there. We haven't done so yet.

One of the challenges with setting up MySQL logging will be to include the common session ID in the log messages to enable correlation of the database logs with application logs and Web requests.

5.5 Operating system

Our infrastructure heavily relies on servers handling requests and processing customer data in the back end. We are using collectd[12] on each machine to monitor individual metrics. The data from all machines is centrally collected and a number of alerts and scripted actions are triggered based on predefined thresholds.

We are also utilizing a number of other log files on the operating systems for monitoring. For example, some of our servers have very complicated firewall rules deployed. By logging failed requests we can identify both mis-configurations and potential attacks. We are able to identify users that are trying to access their services on the platform, but are using the wrong ports to do so. We mine the logs and then alert the users of their mis-configurations. To assess the attacks, we are not just logging blocked connections, but also selected passed ones. We can for example monitor our servers for strange outbound requests that haven't been seen before. Correlating those with observed blocked connections gives us clues as to the origin of these new outbound connections. We can then use the combined information to determine whether the connections are benign or should be of concern.

5.6 Backend

We are operating a number of components in our SaaS's backend. The most important piece is a Java-based server component we instrumented with logging code to monitor a number of metrics and events. First and foremost we are looking for errors. We instrumented Log4J[19] to log through syslog. The severity levels defined earlier in this paper are used to classify the log entries.

In order to correlate the backend logs with any of the other logs, we are using the session ID from the original Web requests and pass them down in any query made to the back end as part of the request. That way, all the backend logs contain the same session ID as the original request that triggered the backend process.

Another measure we are tracking very closely is the number of exceptions in the logs. In a first instance, we are monitoring them to fix them. However, not all of them can be fixed or make sense to be fixed. What we are doing however, is monitoring the number of exceptions over time. An increase shows that our code quality is degrading and we can take action to prioritize work on fixing them. This has shown to be a good measure to balance feature vs. bug-related work and often proofs invaluable when investigating security issues.

6. FUTURE TOPICS

There are a number of issues and topics around cloud-based application logging that we haven't been able to discuss in this paper. However, we are interested in addressing them at a future point in time are: security visualization[1], forensic timeline analysis, log review, log correlation, and policy monitoring.

7. CONTRIBUTIONS

To date, there exists no simple and easy to implement framework for application logging. This paper presents a basis for application developers to guide the implementation of efficient and use-case oriented logging. The paper contributes guidelines for when to log, where to log, and exactly what to log in order to enable the three main log analysis uses: forensic investigation, reporting, and correlation. Without following these guidelines it is impossible to forensically recreate the precise actions that an actor undertook. Log collections and logging guidelines are an essential building block of any forensic process.

The logging guidelines in this paper are tailored to today's infrastructures that are often running in cloud environments, where asynchronous operations and a variety of different components are involved in single user interactions. They are designed for investigators, application developers, and operations teams to be more efficient and to better support their business processes.

8. BIOGRAPHY

Raffael Marty is an expert and author in the areas of data analysis and visualization. His interests span anything related to information security, big data analysis, and information visualization. Previously, he has held various positions in the SIEM and log management space at companies such as Splunk, ArcSight, IBM research, and PriceWaterhouse Coopers. Nowadays, he is frequently consulted as an industry expert in all aspects of log analysis and data visualization. As the founder of Loggly, a logging as a service company, Raffy spends a lot of time re-inventing the logging space and - when not surfing the California waves - he can be found teaching classes and giving lectures at security conferences around the world.

References

[1] Raffael Marty, *Applied Security Visualization*, Addison Wesley, August 2008.

[2] *Asynchronous JavaScript Technology*, https://developer.mozilla.org/en/AJAX.

[3] *Apache Module mod_unique_id*, http://httpd.apache.org/docs/2.1/mod/mod_unique_id.html.

[4] *ArcSight - Common Event Format*, www.arcsight.com/solutions/solutions-cef.

[5] *Amazon Web Services*, http://aws.amazon.com.

[6] *AWS Elastic Load Balancing* http://aws.amazon.com/elasticloadbalancing.

[7] *Amazon Relational Database Service* http://aws.amazon.com/rds.

[8] *Bessemer cloud computing law Number 2: Get Instrument rated, and trust the 6C's of Cloud Finance*, www.bvp.com/About/Investment_Practice/Default.aspx?id=3988. Accessed June 6, 2010.

[9] Bridgewater, David, *Standardize messages with the Common Base Event model*, IBM DeveloperWorks, 21 Oct 2004.

[10] *Common Event Expression* http://cee.mitre.org.

[11] *Cloud Computing*. in Wikipedia. Retrieved June 2, 2010 from http://en.wikipedia.org/wiki/Cloud_computing.

[12] *collectd Â£ The system statistics collection daemon*, www.collectd.org.

[13] *Django*, Web framework www.djangoproject.com.

[14] *Django 1.2 logging patch* www.loggly.com/wp-content/uploads/2010/04/django_logging_1.2.patch

[15] M. Dacier et al. *Design of an intrusion-tolerant intrusion detection system*, Maftia Project, deliverable 10, 2005.

[16] Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[17] IDWG (Intrusion Detection Working Group), *Intrusion Detection Exchange Format*, www.ietf.org/html.charters/idwg-charter.html.

[18] *ISO 8601 - Data elements and interchange formats Â£ Information interchange Â£ Representation of dates and times*, International Organization for Standardization.

[19] *Apache log4j*, Java Logging, http://logging.apache.org/log4j.

[20] *Loggly Inc.*, www.loggly.com.

[21] *Event Processing Â£ Normalization*, http://raffy.ch/blog/2007/08/25/event-processing-normalization/, accessed June 4, 2010.

[22] *Linux/UNIX Audit Logs*, http://raffy.ch/blog/2006/07/24/linux-unix-audit-logs.

[23] *Fixing Client IPs in Apache Logs with Amazon Load Balancers*, www.loggly.com/2010/03/fixing-client-ips-in-apache-logs-with-amazon-load-balance Accessed June 11, 2010.

[24] Pete Finnigan, *Introduction to Simple Oracle Auditing*, www.symantec.com/connect/articles/introduction-simple-oracle-auditing.

[25] PCI security standards council, *Payment Card Industry (PCI), Data Security Standard*, Version 1.2.1, July 2009.

[26] *Python Logging*, www.python.org/dev/peps/pep-0282.

[27] *rsyslgo*, www.rsyslog.com.

[28] *Software as a Service* in Wikipedia. Retrieved June 2, 2010 from http://en.wikipedia.org/wiki/Software_as_a_service.

[29] , ICSA Labs. *The Security Device Event Exchange (SDEE)*, www.icsalabs.com/html/communities/ids/sdee/index.shtml

[30] *Splunk Wiki - Common Information Model*, www.splunk.com/wiki/Apps:Common_Information_Model.

[31] *Syslog(3) - Man page*, http://linux.die.net/man/3/syslog.

[32] *Syslog-ng logging system*, www.balabit.com/network-security/syslog-ng.

[33] *Thor: A tool to Test Intrusion Detection System by Variations of Attacks*, http://thor.cryptojail.net/thor.

[34] *Common Dictionary and Event Taxonomy*, http://cee.mitre.org/ceelanguage.html#event.

[35] *OpenXDAS*, http://openxdas.sourceforge.net/.