ABLS

An Attribute-Based Logging System for the Cloud

User Manual

Christopher A. Wood
caw4567@rit.edu

January 26, 2013

Abstract

User-based non-repudiation is an increasingly important property of cloud-based applications. It provides irrefutable evidence that ties system behavior to specific users, which in turn enables the strict enforcement of organizational security policies. System logs, which can be used to construct audit trails, are typically used as the basis for this property. Thus, the effectiveness of system audits based on log files reduces to the problem of maintaining the integrity and confidentiality of log files. This manual covers the usage of ABLS, an attribute-based logging system for the cloud. It covers system bootstrapping, configuration, and third-party library installation. For more technical information about this project, please see the ABLS research paper.

Contents

Chapter 1

Getting Started

1.1 Bootstrapping

ABLS comes packaged with a set of configuration scripts and SQL files that initialize the database to a clean state. These files are included in the DatabaseModule directory that comes packaged with ABLS, as shown below:

```
ABLS
├── Main.py - main executable
├── Bootstrap.py - bootstrap file for the database
├── LoggerModule
├── PolicyEngineModule
├── AuditModule
├── VerifyModule
├── CryptoModule
├── Common
├── TestModule
└── DatabaseModule
    ├── bootstrap - bash script
    └── bootstrap SQL files
```

In order to bootstrap an ABLS instance for development or debugging purposes, one can simply run the following commands.

```
$> ./DatabaseModule/bootstrap
$> python Boostrap.py
$> python Main.py -l
```

The first bootstrap script will wipe the database files and configure them for use with an ABLS instance. This script should be modified if the user wants to change the physical location of each database server. The second command will tell the Bootstrap program to insert a set of fake data into the log, user, and audit_user databases. This will enable the developer to test the new ABLS instance using some predefined data. Finally, the third command runs the `Main.py` and starts the logging service ("-l") so that new log messages may be intercepted from a client.
If the user wants to start the verification or audit services as well they can simple pass the "-v" or "-a" flags to the `Main.py` program, respectively. Parameters for these services (i.e. the number of verification threads) can be configured by changing the source code in the respective modules (`VerificationModule` and `AuditModule`).

1.2 Configuration

The network and database connectivity options for an ABLS instance are defined in the file `abls.conf`, which is located in the root directory of an ABLS system. Users can modify this file to change the network settings

(i.e. host name, log proxy port, audit proxy port, etc) and the database connections. A snippet of a configuration file is shown below.

```
# Network configuration paramters
abls_host = localhost
abls_logger_port = 9998
abls_audit_port = 9999

# Database configuration string
location.db.log = ~/DatabaseModule/log.db
location.db.key = ~/DatabaseModule/key.db
location.db.users = ~/DatabaseModule/users.db
location.db.audit_users = ~/DatabaseModule/audit_users.db
location.db.policy = ~/DatabaseModule/policy.db
```

Since ABLS is in the prototype phase and does not need to be deployed to a production environment, it only supports local SQLite databases. Thus, the database location strings simply correspond to the names of local database files that are used to persist all log information used at runtime. Future versions of ABLS will provide the user with a more comprehensive set of database configuration options.

1.3 Installation

ABLS utilizes many third-party packages and libraries to run. For brevity, these are listed below along with the online locations where they can be downloaded. The user is left with the task of installing them on their own machines in order to deploy an ABLS instance.

1. Charm Crypto - `http://charm-crypto.com/Main.html`
2. Pykka (Python Akka Library) - `http://www.pykka.org/en/latest/`
3. SQLite - `http://www.sqlite.org/`

Chapter 2

Usage

2.1 Log Proxy Test Driver

Assuming an ABLS instance has been started with the logging serice enabled, one can interface and send test data to the log server as follows:

```
$> python LogProxyDriver.py localhost 9998
```

Once loaded, the log proxy will display something similar to the following:

```
('127.0.0.1', 9998)
('AES256-SHA', 'TLSv1/SSLv3', 256)
{'notAfter': 'Dec 25 19:21:03 2013 GMT',
 'subject': ((('countryName', u'US'),),
             (('stateOrProvinceName', u'NY'),),
             (('localityName', u'Rochester'),),
             (('organizationName', u'RIT'),),
             (('organizationalUnitName', u'CS'),),
             (('commonName', u'Chris'),),
             (('emailAddress', u'NOT_TELLING_YOU@gmail.com'),))}
-----------------------------------------------
                Log Proxy Driver
Type 'help' or '?' for available commands
-----------------------------------------------
>>
```

At this point, the user may type "help" to see what commands are available, or simply start sending fake log data by typing "test", which will result in the following:

```
{"userId":1,"sessionId":0,"payload":"TEST PAYLOAD"}
{"userId":1,"sessionId":0,"payload":"TEST PAYLOAD"}
{"userId":1,"sessionId":0,"payload":"TEST PAYLOAD"}
{"userId":1,"sessionId":0,"payload":"TEST PAYLOAD"}
{"userId":1,"sessionId":0,"payload":"TEST PAYLOAD"}
...
...
```

By examinig the `ablS.log` file and the `log.db` databse, the user can verify that the contents of these log messages were properly stored.

2.2 Audit Proxy Test Driver

Assuming an ABLS instance has been started with the audit service enabled, one can interface and request log data as follows:

```
$> python AuditProxyDriver.py localhost 9999
```

Once loaded, the audit proxy will display something similar to the following:

```
--------------------------------------------------
                Audit Proxy Driver
Type 'help' or '?' for available commands
--------------------------------------------------
>>
```

At this point, the user must first login before they can request data. Assuming the `Bootstrap.py` file was run prior to loading the ABLS instance, the user may log in as follows:

```
>> login bob bobPassword
{"result":True,"message":"Login successful."}
```

Now that the user is verified, they may request log data by specifying user IDs or user and session IDs, as shown below.

```
>> selectByUser 1
{u'message': u'["ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668",
"ded2cde2817cd22bc204ef1265dce668"]',
u'result': True}
```