Real-Time Oracle 11g Log File Analysis

How To Configure Splunk to Monitor and Troubleshoot the Database

By Peter Magee
Lead Database Architect
RCF Information Systems

February 2013

Abstract

The core of virtually every application that manipulates data is the database. It is vitally important, in both production and development environments, to understand at all times what the database is doing and why. There are a variety of commercial database monitoring solutions available in the market today; most rely on SQL monitoring and monitoring of the Oracle Alert Log. Few provide customizable search capabilities or allow easy correlation of events recorded in different files, within the database, from script output, from TCP feeds, or allow a DBA to correlate database events with events from other applications in the technology stack. In this paper I will describe a robust monitoring architecture that allows in-depth, real-time log file analysis and reporting of an individual Oracle database for virtually no cost.

Introduction

The core piece of virtually every application that manipulates data is the database. When the database is performing well, so usually is the application. When the database is not performing as expected, the application is almost certainly experiencing problems. It is vitally important, in both production and development environments, for the database administrator to understand at all times what the database is doing and why. Event monitoring and diagnostic troubleshooting are therefore two of the most important functions of any DBA. Unfortunately, the information required to build an accurate picture of events is often spread across a variety of sources and is difficult to correlate or compile. The difficulty is compounded in any environment where lost time represents lost productivity or revenue. Problems must be identified quickly and accurately so that impact to end users and customers can be minimized.

There are a variety of commercial database monitoring solutions available in the market today; most rely exclusively on SQL monitoring for performance analysis or simple monitoring of the Oracle Alert Log. While this data is certainly critical, there is a great deal more information available that these solutions generally overlook. Few of these applications provide customizable search capabilities or allow easy correlation of events recorded in different files, within the database, from script output, from TCP feeds, or allow a DBA to correlate database events with events from other applications in the technology stack. In this paper I will describe a robust monitoring architecture – called the Oracle 11g Intelligence App – that allows in-depth, real-time log file analysis and reporting of an individual Oracle database for virtually no cost, and that can be expanded to monitor large, heterogeneous computing environments at relatively low cost.

Architecture

Server

For the purposes of writing this paper and creating the Oracle 11g Intelligence App, I used a virtual server environment based on Oracle VM VirtualBox. My initial server image was the “Database App Development VM”, downloaded from Oracle.com. The server image included Oracle Linux 5, Oracle Database 11g R2 Enterprise Edition, and Oracle Application Express 4.1. By default, Oracle has set all operating system and database account passwords (oracle, root, sys, etc.) for this system image to "oracle".

Oracle Data Sources

Oracle provides a number of potential sources for log and metric data. The possibilities are almost limitless. Within the scope of this project, I included a sampling of major data sources as described in the following table.

Data Type	Data Location
Database Alert Log	/home/oracle/app/oracle/diag/rdbms/orcl/orcl/trace/alert_orcl.log
Database Audit Trail	/home/oracle/app/oracle/diag/syslog/orcl/audit_orcl.log
Scheduler Log	SYS.DBA_SCHEDULER_JOB_RUN_DETAILS

Listener Log	/home/oracle/app/oracle/diag/tnslsnr/localhost/trace/listener.log
APEX Web Activity Logs	APEX_040100.WWV_FLOW_ACTIVITY_LOG1$ APEX_040100.WWV_FLOW_ACTIVITY_LOG2$
APEX Web Access Logs	APEX_040100.WWV_FLOW_USER_ACCESS_LOG1$ APEX_040100.WWV_FLOW_USER_ACCESS_LOG2$

Splunk

Splunk is a commercial software tool that collects, indexes, and harnesses machine data generated by IT systems and infrastructure (Splunk, Inc., 2012). It tracks log files, the output of shell scripts, database contents, and even direct TCP feeds, then indexes the results and allows administrators to query and report across the entire collection. Custom dashboards, summary reports, charts, or detailed line-by-line output can be created in seconds. The enterprise version of the software also allows the creation of monitors and alerts.

With the free version of Splunk I can index up to 500MB of log entries per day, which is generally plenty when looking at a single, modestly-sized database. For comparison, a two-node Enterprise 11gR2 RAC configuration, supporting multiple financial 3-tier web applications for a 12,000 member user community only generates about 200MB of log data per day including alert logs, audit trails, listener logs, and operating system metrics. The free version of Splunk lacks some of the nicer bells and whistles of the enterprise version, but still offers a lot of functionality that if carefully implemented can be really useful to any database administrator. Universal real-time indexing, real-time and historical searches, reporting, knowledge mapping and dashboards are just a few of the options available. I have personally found these capabilities invaluable when troubleshooting a variety of database and networking issues.

Splunk DB Connect

Splunk DB Connect is a Splunk "app": a free add-on that allows Splunk to directly query database data in addition to machine data found in log files (Splunk, Inc., 2012). Using DB Connect, also called DBX, I can expand my monitoring to include information generated or maintained internally by the database, such as tablespace growth, application audit trails stored as database tables, initialization parameters, performance metrics, scheduled job results, and the like.

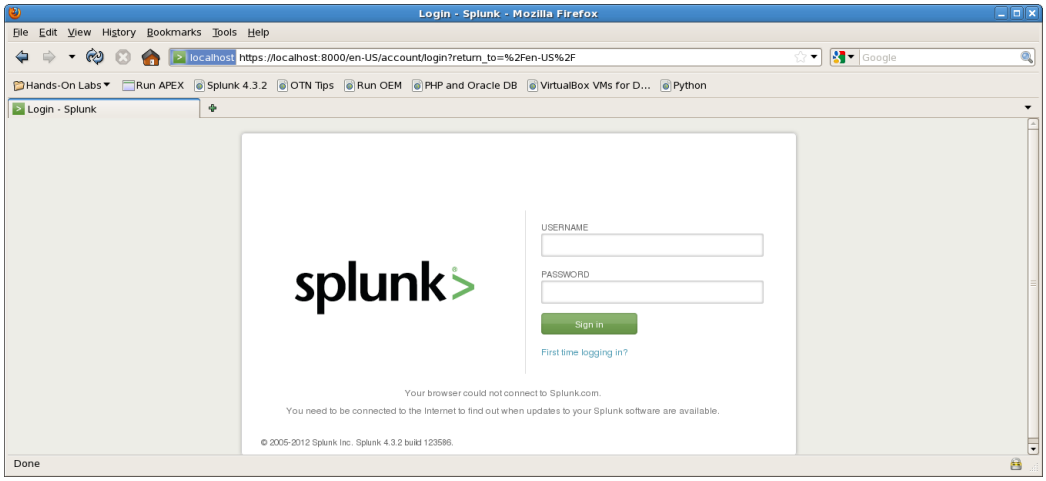
Splunk Installation

Oracle software is pre-installed on the system image I downloaded from Oracle, but Splunk is not. The Splunk installation can be downloaded in RPM format from splunk.com. For the Oracle 11g Intelligence project I used both Splunk versions 4.3.2 and 5.0.1. After copying the installation file onto my virtual system, I can install Splunk in the default /opt/splunk directory using the following commands as the root user:

```
# rpm -i splunk-5.0.1-143156.i386.rpm
# export SPLUNK_HOME=/opt/splunk
# export PATH=$PATH:$SPLUNK_HOME/bin
# splunk start --accept-license
```

```
# splunk enable boot-start
```

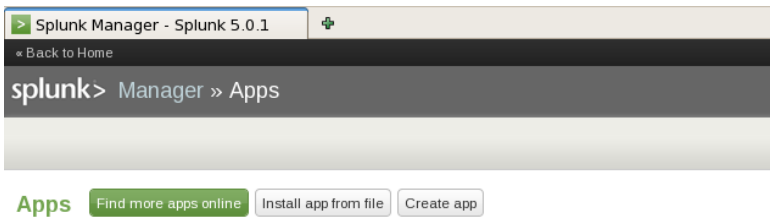
Once installed, Splunk web services are available on localhost, port 8000. The default username and password is "admin" and "changeme".



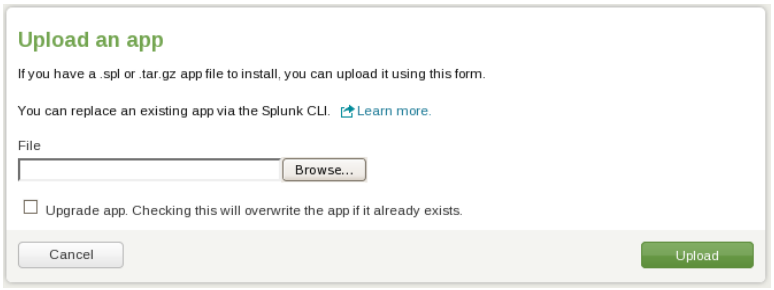
Splunk DB Connect Installation

Splunk DBX can be downloaded from the splunk-base.com web site. For this project I used DBX versions 1.0.6. and 1.0.8. DBX also requires the use of the Oracle JDBC driver for Java 1.6 – ojdbc6.jar – available from the Oracle TechNetwork web site. After downloading both the DB Connect distribution file and the JDBC file to my virtual system, I can install the app with the following steps.

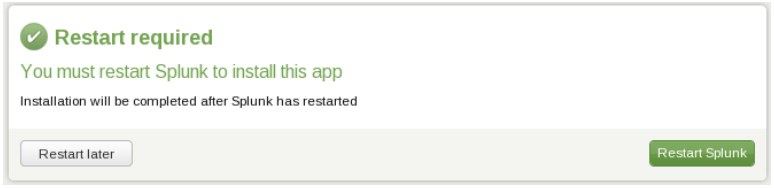
1. Once logged in to Splunk, I can install the DBX App by clicking on the “App” link in the upper right hand corner of the page and selecting “Manage Apps” from the menu.



From the Manager window, I click the “Install App from File” button. Following the on-screen directions and selecting the DBX distribution file (e.g. splunk_app_dbx-1.0.6-144521.tar.gz) for upload from my local file system, the initial installation is soon complete.



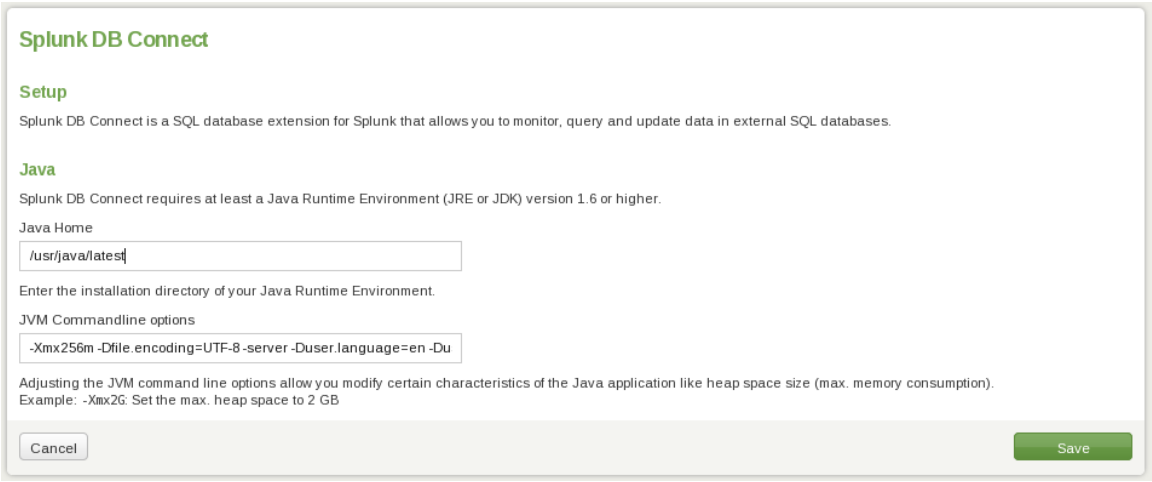
2. When prompted, I click the "Restart Splunk" button.



3. While Splunk is restarting, I need to place the Oracle JDBC library where DBX will expect to find it. As the root user, I must copy my downloaded ojdbc6.jar file to the /opt/splunk/etc/apps/dbx/bin/lib directory.

```
# cd /home/oracle/Desktop
# cp ojdbc6.jar /opt/splunk/etc/apps/dbx/bin/lib
```

4. After Splunk is finished restarting and the JDBC library is in place, I can log back in to Splunk and finish configuring the app. As soon as I log in I need to click the "Set up now" button on the "Installation Successful" screen. I will be prompted to supply additional information, including the JAVA_HOME path. The Java location on my virtual system is /usr/java/latest.



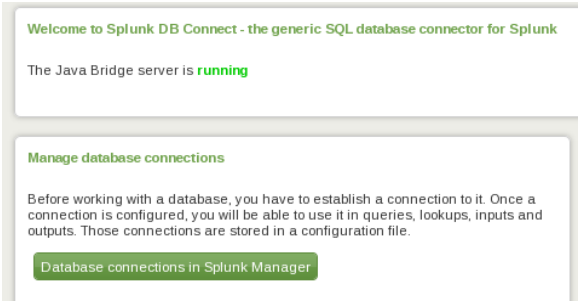
5. After the DBX configuration has been saved, I must restart Splunk one more time. The simplest way to restart is from the command line.

```
# /opt/splunk/bin/splunk restart
```

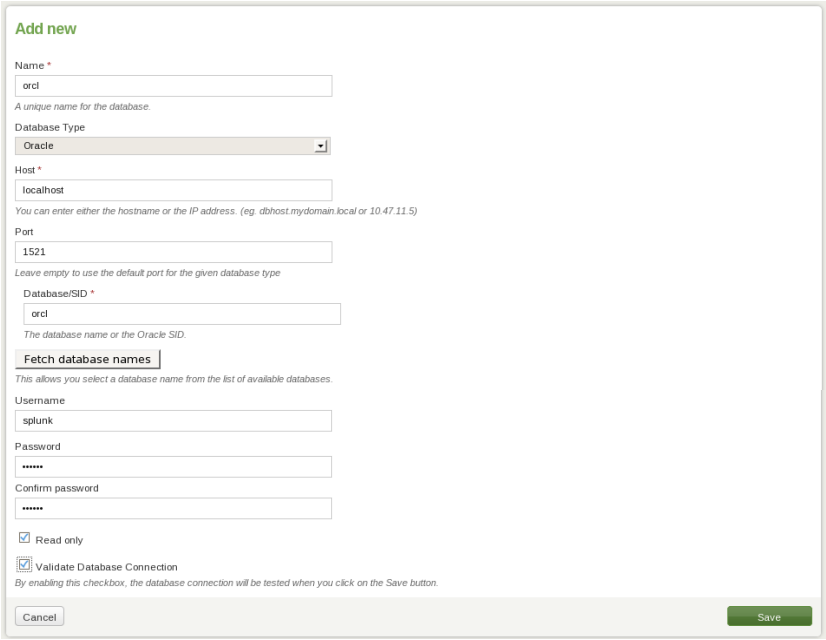
6. Before connecting Splunk directly to my database, I need to create a database account for the app to use. While Splunk is restarting, I can finish the database configuration. During this project I granted very broad privileges to my monitor account for demonstration purposes; in an actual production system, a more limited set of privileges would be more appropriate.

```
# sqlplus /nolog
SQL> conn / as sysdba
SQL> create user splunk identified by oracle default tablespace users
temporary tablespace temp;
SQL> grant create session, select any table, select any dictionary to splunk;
SQL> exit
```

7. When my database user is created and Splunk has restarted for the final time, I am ready to configure DBX. To create the database connection, I log in to Splunk and select the "Splunk DB Connect" app from the Apps menu or from the Home screen. I confirm that the Java Bridge Server is running, then click the "Database connections in Splunk Manager" button.



8. On the database management screen, I click "New," fill out the information for the local database connection, and click "Save". It is important to set the "Read Only" and "Validate Database Connection" options; there is no need for a monitoring app to make changes to the database, and it is always good to find out as soon as possible if there is some kind of error with my configuration.



Splunk Security

One feature that is a little more difficult to live without in the free version of Splunk is user and role-based security. The free version of Splunk has one user account, admin, which has full control of the application. It is highly advisable to either limit the interfaces on which the user interface is available to localhost – forcing a user to have an account on the Splunk server and run a local browser – or to use a proxy server of some sort to perform user authentication, and limit access to those people who should have administrator privileges. The enterprise version of Splunk offers a complete range of role-based access controls that are considerably more robust – allowing users to access only specific data or data types – but that comes with a price tag.

To limit access to the Splunk console to the localhost, I need to set the following entry in the /opt/splunk/etc/system/local/web.conf file. If the web.conf file does not exist in the "local" directory yet, I can copy it from the /opt/splunk/etc/system/default directory to the local directory and update it. Once the socket_host parameter has been set, I must restart Splunk to make it take effect.

```
server.socket_host = localhost
```

It is also a good idea to change the admin password. Splunk may prompt me to change it the first time I log in to the user interface. I can also set the new password from the command line, like this:

```
# splunk edit user admin -password [new password] -auth admin:changeme
```

Creating a Splunk App for Oracle Monitoring

While all of the steps I am about to describe can be accomplished using the web user interface, I find that it is often just as fast to continue working from the command line. Once Splunk has been installed and is running, I can quickly create my application and begin collecting data from there.

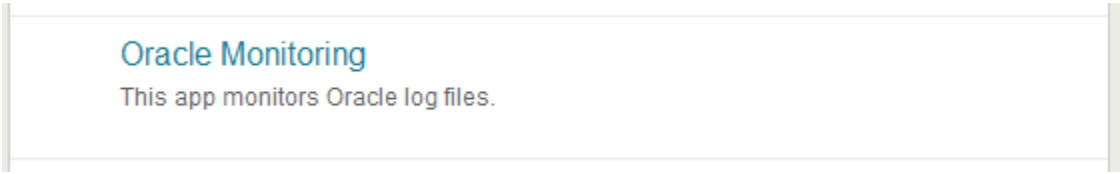
I always make sure to configure the shell environment to include the Splunk "bin" directory in the PATH.

```
# export SPLUNK_HOME=/opt/splunk
# export PATH=$PATH:$SPLUNK_HOME/bin
```

A custom Splunk application, or "app" can then be created from the command line, as I've shown here. I use the Splunk admin username and password when prompted for login credentials.

```
# splunk create app ora_11g_demo -label "Oracle Monitoring" -author "Pete Magee" -
template barebones -visible 1 -description "This app monitors Oracle log files"
Your session is invalid.  Please login.
Splunk username: admin
Password:
Login successful, running command...
App 'ora_11g_demo' is created.
```

Now my application will be listed on the Home page when I log in to Splunk. But what does it do? The answer is not much, until I configure it to properly receive and interpret the data I want to search.



Splunk Indexing

Before I can collect data, I must have somewhere to put it. From the command line, I can create indexes rapidly, associating them with my new application, as shown:

```
# splunk add index oracle_audit -app ora_11g_demo
Index oracle_audit added.
```

I must repeat this process to create indexes for each major log or input type. Having separate indexes allows me to improve performance by limiting the amount of data I have to search for any particular query or report. It also allows me to restrict access to certain types of data through User Roles if using the enterprise edition of Splunk.

In my example app, I create the following indexes:

Index	Description
oracle_apex	APEX activity logs
oracle_audit	Audit trail entries
oracle_backup	Backup logs
oracle_dbx	DB Connect data
oracle_net	Network listener logs

oracle_rdbms	Database alert logs
oracle_test	An index for testing new inputs before storing data in a real index

Depending on the target environment, the following indexes may also be useful:

Index	Description
oracle_asm	ASM alert logs
oracle_rac	Oracle RAC or cluster log files

Splunk Source Types

While it is a good idea to create multiple indexes to support major categories of data input, it is not practical for me to create a new index for every individual data source I might want to monitor. Splunk source types allow me to categorize different types of data input within a single index – sub categories for my major input types. For instance, a source type definition would allow me to tell the difference between an RMAN backup log and a DataPump log in the oracle_backup index, or a local listener log and a Connection Manager log in the oracle_net index. Source types are attributes of individual data inputs or monitors, which I will describe in following sections of this paper. Below are some of the various source types that I commonly use:

Index	Source Type	Description
oracle_apex	apex_logs	APEX log records stored in database tables.
oracle_audit	os_audit	Database audit trail records stored in an operating system file.
	db_audit	Database audit trail records stored in a database table.
oracle_backup	Datapump	Datapump output log files.
	Rman	Oracle RMAN logs.
oracle_dbx	job_log	Oracle Scheduler job logs.
oracle_net	Listener	Local listener log files.
	Cman	Oracle Connection Manager log files.
oracle_rdbms	Alert	Oracle alert log files.

Oracle Data Collection

Data on Oracle database activity can be collected from a number of different sources, including traditional log files, shell scripts, SQL queries, and direct TCP input.

File Monitors

A file monitor allows me to watch for new additions to log files, or for new log files to be added to a particular directory. File monitors can be created from the command line, using the "splunk add monitor" command as shown in the following sections. Common log files to watch include the Oracle alert logs, listener logs, backup logs and audit trail.

Alert Logs

The Oracle alert log may contain error messages, startup and shutdown messages, and a variety of other useful information. As the SYSDBA user, I can make sure that redo log checkpoints and switches are

recorded in the alert log as well; this allows me to track the high-level transaction activity of my database.

```
# sqlplus / as sysdba

SQL> alter system set log_checkpoints_to_alert=TRUE scope=spfile;

System altered.
```

Once my alert logging is configured the way I want it, I can add a file monitor that will capture the contents of any database alert logs on my server. Most Oracle diagnostic information, including alert logs, is now captured in the Automatic Diagnostic Repository, or ADR. The location of the ADR can be determined from the DIAGNOSTIC_DEST initialization parameter, and is often the same as the ORACLE_BASE directory. In the Oracle VM system image that I downloaded, the ADR path is /home/oracle/app/oracle/diag. Once the ADR path is determined, the monitor can be created.

```
# splunk add monitor /home/oracle/app/oracle/diag/rdbms/*/*/trace/alert*.log -
sourcetype alert -index oracle_rdbms -app ora_11g_demo
Monitor added.
```

I must specify the source type, index, and app name for each monitor that I create. Also note the use of wildcards in the path; this allows me to monitor all alert logs on the system with a single monitor, with no need to reconfigure Splunk as databases are added to or removed from the system.

Listener Logs

Oracle's listener and connection manager logs contain a wealth of information on database client programs, including OS user names, program names, and IP address and port information. This data is invaluable both for security monitoring and network connectivity troubleshooting. Listener and connection manager logs are also located in the ADR directory tree. I can create monitors on these files in the same way that I did for the alert logs.

```
# splunk add monitor /home/oracle/app/oracle/diag/tnslsnr/*/*/trace/*.log -
sourcetype listener -index oracle_net -app ora_11g_demo
Monitor added.
```

```
# splunk add monitor /home/oracle/app/oracle/diag/netcman/*/*/trace/*.log -
sourcetype cman -index oracle_net -app ora_11g_demo
Monitor added.
```

Backup Logs

Whether I am using Oracle RMAN or DataPump to create backups of my database, I will generate log data that will contain important information. Not only can I monitor easily for error messages, but I can also look for trends in my backups over time, like the number or size of backup files, or the number of rows exported for a particular table.

```
# splunk add monitor /home/oracle/app/oracle/diag/rman/*/*.log -sourcetype rman -
index oracle_backup -app ora_11g_demo
Monitor added.
```

Audit Trail

When configuring the Oracle audit trail, I often make use of the Linux syslog utility to store records in a file, external to the database. The reason for this is three-fold. First, it reduces processing overhead on the database. In a busy system with many audits enabled, this overhead can be significant; I have seen systems where auditing accounted for almost 50% of all transaction processing. Since I am now sending records to the operating system instead of a database table, there is no need to record inserts in the online redo logs, or consume data file or database backup space with static records. It also makes feeding the records into Splunk for analysis much simpler; file monitoring (as I have already shown) is quite simple to configure. Second, it allows me to save disk capacity and automate my retention policy. Using the Linux log rotate utility, I can rotate out, compress, or delete old syslog files automatically, while still maintaining a complete audit history in Splunk. Lastly, it is more secure. By storing audit trail records outside the database in multiple locations, possibly even on other systems, it becomes impossible for an intruder to delete or alter records, even with elevated database privileges.

The steps I use to configure Oracle and Linux for this type of auditing are as follows:

1. As the root user, I create a directory in the ADR to hold my audit trail file.

```
# mkdir /home/oracle/app/oracle/diag/syslog
# mkdir /home/oracle/app/oracle/diag/syslog/orcl
```

2. Next I configure a syslog service in the /etc/syslog.conf file to direct messages to an audit trail file in the directory I created.

```
# User defined messages
local3.debug        /home/oracle/app/oracle/diag/syslog/orcl/audit_orcl.log
```

3. Then I restart the syslog service.

```
# service syslog restart
Shutting down kernel logger:                               [  OK  ]
Shutting down system logger:                               [  OK  ]
Starting system logger:                                    [  OK  ]
Starting kernel logger:                                    [  OK  ]
```

   On a system with multiple databases, I will need to configure each one to use a different syslog input (i.e. local4.debug, local5.debug, etc.) with a different log file destination to avoid confusing which records belong to each database.

4. As a database user with DBA privileges, I reset the audit_syslog_level, audit_sys_operations, and audit_trail initialization parameters and restart the database. The audit_syslog_level parameter should match the input message type that syslog is expecting for my log file (local3.debug). In a secure environment, the audit_syslog_operations parameter should be set to "true" to make sure I capture even the highest-level operations. The audit_trail parameter must be set to "OS" to ensure that I am not using internal database storage for my audit data.

```
# sqlplus / as sysdba

SQL> alter system set audit_syslog_level='local3.debug' scope=spfile;

System altered.

SQL> alter system set audit_sys_operations=TRUE scope=spfile;

System altered.

SQL> alter system set audit_trail='OS' scope=spfile;

System altered.
```

5. As a database user with DBA privileges, I can now configure my specific database audits. It is generally a good idea to audit sessions and the use of the "ANY" privileges by access. It is also a good idea to audit create, alter, and drop statements in the database. The table below contains a list of the audits I use most frequently.

Recommended Audits	
AUDIT ALL BY ACCESS;	AUDIT DROP JAVA CLASS BY ACCESS;
AUDIT ALL PRIVILEGES BY ACCESS;	AUDIT GRANT EDITION BY ACCESS;
AUDIT SYSDBA BY ACCESS;	AUDIT ALTER JAVA SOURCE BY ACCESS;
AUDIT SYSOPER BY ACCESS;	AUDIT GRANT MINING MODEL BY ACCESS;
AUDIT ALTER ANY OPERATOR BY ACCESS;	AUDIT ALTER MINING MODEL BY ACCESS;
AUDIT ANALYZE ANY DICTIONARY BY ACCESS;	AUDIT CREATE JAVA SOURCE BY ACCESS;
AUDIT ALTER SEQUENCE BY ACCESS;	AUDIT EXEMPT IDENTITY POLICY BY ACCESS;
AUDIT ALTER TABLE BY ACCESS;	AUDIT EXECUTE ANY OPERATOR BY ACCESS;
AUDIT COMMENT TABLE BY ACCESS;	AUDIT ALTER DATABASE LINK BY ACCESS;
AUDIT DEBUG PROCEDURE BY ACCESS;	AUDIT COMMENT MINING MODEL BY ACCESS;
AUDIT EXEMPT ACCESS POLICY BY ACCESS;	AUDIT CREATE JAVA CLASS BY ACCESS;
AUDIT GRANT DIRECTORY BY ACCESS;	AUDIT COMMENT EDITION BY ACCESS;
AUDIT GRANT PROCEDURE BY ACCESS;	AUDIT CREATE JAVA RESOURCE BY ACCESS;
AUDIT GRANT SEQUENCE BY ACCESS;	AUDIT DROP JAVA SOURCE BY ACCESS;
AUDIT GRANT TABLE BY ACCESS;	AUDIT ALTER PUBLIC DATABASE LINK BY ACCESS;
AUDIT GRANT TYPE BY ACCESS;	AUDIT ALL STATEMENTS BY ACCESS;
AUDIT RENAME ON DEFAULT BY ACCESS;	AUDIT OUTLINE BY ACCESS;
AUDIT DROP JAVA RESOURCE BY ACCESS;	AUDIT ALTER JAVA RESOURCE BY ACCESS;
AUDIT ALTER JAVA CLASS BY ACCESS;	AUDIT DIRECT PATH UNLOAD BY ACCESS;
AUDIT DIRECT_PATH LOAD BY ACCESS;	
NOAUDIT EXECUTE ANY LIBRARY;	NOAUDIT EXECUTE ANY PROCEDURE;
NOAUDIT EXECUTE ANY TYPE;	NOAUDIT EXECUTE LIBRARY;
NOAUDIT LOCK ANY TABLE;	NOAUDIT SELECT ANY SEQUENCE;
NOAUDIT SELECT ANY TABLE;	NOAUDIT UPDATE ANY TABLE;
NOAUDIT DELETE ANY TABLE;	NOAUDIT EXECUTE ANY INDEXTYPE;
NOAUDIT EXECUTE ANY OPERATOR;	NOAUDIT INSERT ANY TABLE;
NOAUDIT DELETE TABLE;	NOAUDIT INSERT TABLE;
NOAUDIT UPDATE TABLE;	NOAUDIT EXECUTE PROCEDURE;

```
NOAUDIT SELECT TABLE;                    NOAUDIT SELECT SEQUENCE;
```

```
SQL> audit session by access;

Audit succeeded.

...
```

Once auditing is configured, entries will start to appear in the audit trail file. Then I can configure a Splunk file monitor input to record the data.

```
# splunk add monitor /home/oracle/app/oracle/diag/syslog/*/*.log -sourcetype
os_audit -index oracle_audit -app ora_11g_demo
Monitor added.
```

TCP Inputs

Another way in which I can load Oracle data into Splunk is through a direct TCP connection. Using stored procedures, triggers, or advanced queues instead of external files, I can transmit application data, audit data, or other report data to Splunk in real-time from within the database. This method is particularly useful if the data I want to capture is already stored in the database by an application for its own purposes, and I want to avoid tampering directly with proprietary code or database objects. In most cases it also imposes very little processing overhead on the database. A drawback to this approach is that if Splunk is down for any reason, any data transmitted as a TCP input will be lost.

For the purpose of my sample application, I will be loading Oracle Application Express (APEX) activity logs. Oracle switches APEX log tables every 14 days, purging the contents of the active log before inserting new records. This makes it impossible to maintain much of an online history, and difficult to search as we must first determine which log table is active or may contain the records we want.

Splunk TCP Reception

To configure Splunk to receive TCP inputs on a given network port, I can execute the following command. This configures Splunk to listen on TCP port 8099 for any incoming data, to assign the "apex_logs" sourcetype to that data, store it in the "oracle_apex" index, and resolve the source hostname using DNS.

```
# splunk add tcp 8099 -sourcetype apex_logs -index oracle_apex -resolvehost dns
Listening for data on TCP port 8099.
```

Oracle TCP Transmission

Configuring Oracle to transmit data to Splunk takes a little more effort. The following steps allow me to begin my TCP data feed.

1. I need to create an Oracle schema account to contain all my Splunk-related code and objects. This allows me to keep my custom code separate from any internal Oracle or application code,

and should also be separate from the account used by DB Connect. My schema account is called "splunk_api", and it will have permission to select from various log tables in the apex_040100 schema.

```
CREATE USER SPLUNK_API IDENTIFIED BY oracle
DEFAULT TABLESPACE "USERS"
TEMPORARY TABLESPACE "TEMP";

GRANT "RESOURCE" TO SPLUNK_API;
GRANT "CONNECT" TO SPLUNK_API;
ALTER USER SPLUNK_API DEFAULT ROLE ALL;

GRANT EXECUTE ON UTL_TCP TO SPLUNK_API;
GRANT SELECT ON APEX_040100.WWV_FLOW_ACTIVITY_LOG1$ TO SPLUNK_API;
GRANT SELECT ON APEX_040100.WWV_FLOW_ACTIVITY_LOG2$ TO SPLUNK_API;
GRANT SELECT ON APEX_040100.WWV_FLOW_USER_ACCESS_LOG1$ TO SPLUNK_API;
GRANT SELECT ON APEX_040100.WWV_FLOW_USER_ACCESS_LOG2$ TO SPLUNK_API;
```

2. As a SYSDBA user, I must configure Oracle's internal access control lists (ACLs) to allow me to access the network (Oracle Corporation, 2012). The following PL/SQL script creates an ACL for Splunk that allows it to transmit to port 8099 on the localhost.

```
Begin
dbms_network_acl_admin.create_acl(
acl => 'splunk_acl.xml',
description => 'Splunk ACL',
principal => 'SPLUNK_API',
is_grant => TRUE,
privilege => 'connect');
commit;
end;
/

begin
dbms_network_acl_admin.assign_acl(
acl => 'splunk_acl.xml',
host => '127.0.0.1',
lower_port => 8099,
upper_port => 8099);
commit;
end;
/
```

3. As my new SPLUNK_API user, I can now configure triggers using the util_tcp package to capture and transmit the data I want to index in Splunk. In this case I want to transmit new records in the APEX_040100.WWV_FLOW_ACTIVITY_LOG1$ and WWV_FLOW_ACTIVITY_LOG2$ tables of our test database. Once triggers like the ones below are in place on both tables, Splunk will gather the log data from whichever table happens to be active into a single stream that we can maintain and search independently of Oracle.

```
CREATE OR REPLACE
TRIGGER SPLUNK_API.APEX_ACCESS_LOG1_TRG
```

```
AFTER INSERT ON APEX_040100.WWV_FLOW_ACTIVITY_LOG1$
REFERENCING OLD AS old NEW AS new
FOR EACH ROW
DECLARE
   c utl_tcp.connection;
   v_splunk_entry clob;
   ret_Val pls_integer;
BEGIN
   v_splunk_entry := 'APEX_LOG:[ACTIVITY'||
      '] TIME_STAMP:['||to_char(:new.time_stamp,'MM/DD/YYYY HH24:MI:SS')||
      '] COMPONENT_TYPE:['||:new.component_type||
      '] COMPONENT_NAME:['||:new.component_name||
      '] COMPONENT_ATTRIBUTE:['||:new.component_attribute||
      '] INFORMATION:['||:new.information||
      '] ELAP:['||to_char(:new.elap)||
      '] NUM_ROWS:['||to_char(:new.num_rows)||
      '] USERID:['||:new.userid||
      '] IP_ADDRESS:['||:new.ip_address||
      '] USER_AGENT:['||:new.user_agent||
      '] FLOW_ID:['||to_char(:new.flow_id)||
      '] STEP_ID:['||to_char(:new.step_id)||
      '] SESSION_ID:['||to_char(:new.session_id)||
      '] SECURITY_GROUP_ID:['||to_char(:new.security_group_id)||
      '] SQLERRM:['||:new.sqlerrm||
      '] SQLERRM_COMPONENT_TYPE:['||:new.sqlerrm_component_type||
      '] SQLERRM_COMPONENT_NAME:['||:new.sqlerrm_component_name||
      '] PAGE_MODE:['||:new.page_mode||
      '] CACHED_REGIONS:['||to_char(:new.cached_regions)||
      '] CONTENT_LENGTH:['||to_char(:new.content_length)||
      '] APPLICATION_INFO:['||:new.application_info||
      '] WORKSHEET_ID:['||to_char(:new.worksheet_id)||
      '] IR_SEARCH:['||to_char(:new.ir_search)||
      '] IR_REPORT_ID:['||to_char(:new.ir_report_id)||
      '] WEBSHEET_ID:['||to_char(:new.websheet_id)||
      '] WEBPAGE_ID:['||to_char(:new.webpage_id)||
      '] DATAGRID_ID:['||to_char(:new.datagrid_id)||'] ';

   c := utl_tcp.open_connection(remote_host=>'127.0.0.1',
                                            remote_port=> 8099,
                                            charset => 'AL32UTF8');
   ret_val := UTL_TCP.WRITE_LINE(c,v_splunk_entry);
   utl_tcp.close_connection(c);
EXCEPTION
   WHEN OTHERS THEN RAISE;
END;
/

CREATE OR REPLACE
TRIGGER SPLUNK_API.APEX_ACCESS_LOG2_TRG
AFTER INSERT ON APEX_040100.WWV_FLOW_ACTIVITY_LOG2$
REFERENCING OLD AS old NEW AS new
FOR EACH ROW
DECLARE
   c utl_tcp.connection;
   v_splunk_entry clob;
   ret_Val pls_integer;
BEGIN
   v_splunk_entry := 'APEX_LOG:[ACTIVITY'||
      '] TIME_STAMP:['||to_char(:new.time_stamp,'MM/DD/YYYY HH24:MI:SS')||
      '] COMPONENT_TYPE:['||:new.component_type||
      '] COMPONENT_NAME:['||:new.component_name||
      '] COMPONENT_ATTRIBUTE:['||:new.component_attribute||
```

```
      '] INFORMATION:['||:new.information||
      '] ELAP:['||to_char(:new.elap)||
      '] NUM_ROWS:['||to_char(:new.num_rows)||
      '] USERID:['||:new.userid||
      '] IP_ADDRESS:['||:new.ip_address||
      '] USER_AGENT:['||:new.user_agent||
      '] FLOW_ID:['||to_char(:new.flow_id)||
      '] STEP_ID:['||to_char(:new.step_id)||
      '] SESSION_ID:['||to_char(:new.session_id)||
      '] SECURITY_GROUP_ID:['||to_char(:new.security_group_id)||
      '] SQLERRM:['||:new.sqlerrm||
      '] SQLERRM_COMPONENT_TYPE:['||:new.sqlerrm_component_type||
      '] SQLERRM_COMPONENT_NAME:['||:new.sqlerrm_component_name||
      '] PAGE_MODE:['||:new.page_mode||
      '] CACHED_REGIONS:['||to_char(:new.cached_regions)||
      '] CONTENT_LENGTH:['||to_char(:new.content_length)||
      '] APPLICATION_INFO:['||:new.application_info||
      '] WORKSHEET_ID:['||to_char(:new.worksheet_id)||
      '] IR_SEARCH:['||to_char(:new.ir_search)||
      '] IR_REPORT_ID:['||to_char(:new.ir_report_id)||
      '] WEBSHEET_ID:['||to_char(:new.websheet_id)||
      '] WEBPAGE_ID:['||to_char(:new.webpage_id)||
      '] DATAGRID_ID:['||to_char(:new.datagrid_id)||']';

   c := utl_tcp.open_connection(remote_host=>'127.0.0.1',
                                            remote_port=> 8099,
                                            charset => 'AL32UTF8');
   ret_val := UTL_TCP.WRITE_LINE(c,v_splunk_entry);
   utl_tcp.close_connection(c);
EXCEPTION
   WHEN OTHERS THEN RAISE;
END;
/
```

Shell Script Inputs

Where file monitors are good for collecting data in real time as it is appended to a log file, shell script inputs are good for collecting data at regular intervals. This is useful for things like performance or capacity monitoring. Output from standard operating system commands like `top`, `last`, `vmstat`, or `iostat` can be collected and correlated with specific database events.

According to the "Developing Dashboards, Views, and Apps for Splunk Web" manual, the following are common use cases for scripted inputs (Splunk, Inc., 2012):

- Whenever data is not available as an ordinary file, or the data cannot be sent using TCP or UDP.
- Stream data from command-line tools, such as vmstat and iostat.
- Poll a database, web service, or API for specific data, and process the results with Splunk.
- Reformat complex data so Splunk can more easily parse the data into events and fields.
- Maintain data sources with slow or resource-intensive startup procedures.
- Provide special or complex handling for transient or unstable inputs.
- Scripts that manage passwords and credentials.
- Wrapper scripts for command line inputs that contains special characters

An Oracle-specific application for a shell script input is the tnsping utility. This utility can be used to monitor the network latency between the local database and a remote database.

```
# tnsping remote_db

TNS Ping Utility for Linux: Version 11.2.0.2.0 - Production on 21-NOV-2012 15:10:23
Copyright (c) 1997, 2010, Oracle.  All rights reserved.

Used TNSNAMES adapter to resolve the alias

Attempting to contact (DESCRIPTION = (ADDRESS_LIST = (SOURCE_ROUTE = on) (ADDRESS =
(PROTOCOL = TCP)(HOST = 172.16.0.152)(PORT = 1521))) (CONNECT_DATA = (SERVICE_NAME
= remote_db)))
OK (10 msec)
```

To turn this into a shell script monitor, I will create a shell script to monitor the specific remote database I am interested in. My script will be named "tnsping.sh" and should be located in the /opt/splunk/etc/apps/oracle_monitor/bin directory.

```
#!/bin/ksh
export ORACLE_HOME=/home/oracle/app/oracle/product/11.2.0/dbhome_2
$ORACLE_HOME/bin/tnsping remote_db
```

Once the script is created and made executable (with `chmod 750 tnsping.sh`), I must create an "executable" Splunk monitor to run it and collect the output.

```
# splunk add exec /opt/splunk/etc/apps/ora_11g_demo/bin/tnsping.sh -sourcetype
tnsping -index oracle_net -interval 300 -app ora_11g_demo
```

For monitoring the operating system, rather than re-invent the wheel I recommend using the Splunk's free "*NIX" app, which contains a full suite of performance and security related shell script inputs. Data from these inputs can also be incorporated into Oracle monitoring dashboards. It is generally wise, unless you have a specific need for them, to disable the `lsof` input and possibly even the `netstat` and `ps` inputs, as they tend to generate more data volume than most users would have a license for.

DB Connect Inputs

As I mentioned earlier, Splunk DB Connect allows me to collect data directly from the database; however it is important to consider exactly which data is worth collecting in this manner. Additional SQL processing imposed by too much monitoring will affect database performance, so I want to keep my footprint as small as possible. It does not make a lot of sense to monitor data with SQL commands when the same data can be obtained through a log file monitor, or through a TCP port monitor and trigger arrangement, which would impose far less processing overhead. It makes more sense to use DBX's abilities to monitor data which cannot be obtained in any other way. Tables or views owned by the SYS user are a perfect example: they contain a mountain of useful information, and cannot be accessed via log files or triggers. If I keep my monitoring interval reasonable, selection of key data from these tables will not impose an undue burden on my database.

The DBA_SCHEDULER_JOB_RUN_DETAILS view contains a history of Oracle Scheduler Job results, including error messages and performance metrics. By placing the following code in the

/opt/splunk/etc/apps/ora_11g_demo/local/inputs.conf file and restarting Splunk, I can automatically import log entries from the view, using the database connection I configured earlier.

```
[dbmon-tail://orcl/scheduler_job_run_details]
host = localhost
index = oracle_dbx
output.format = kv
output.timestamp = 0
output.timestamp.format = yyyy-MM-dd HH:mm:ss
output.timestamp.parse.format = yyyy-MM-dd HH:mm:ss
query = select to_char(log_date,'YYYY-MM-DD HH24:MI:SS') log_date, log_id, owner,
job_name,  status, error# return_code, to_char(req_start_date,'YYYY-MM-DD
HH24:MI:SS') req_start_date, to_char(actual_start_date,'YYYY-MM-DD HH24:MI:SS')
actual_start_date, to_char(run_duration) run_duration, instance_id, session_id,
to_char(cpu_used) cpu_used, additional_info from dba_scheduler_job_run_details
{{WHERE $rising_column$ > ?}}
sourcetype = job_run_details
tail.rising.column = LOG_ID
interval = auto
table = scheduler_job_run_details
```

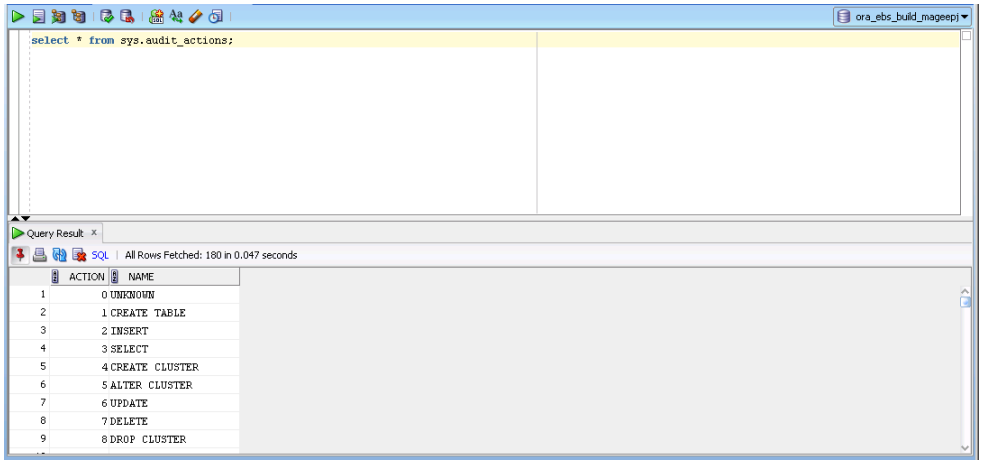
While this form of monitoring may place more load on the database server, it is important to note that because DB Connect is can track a specific log id for each entry, no data will be lost in the event that either Splunk or Oracle is down. Also, if using the “interval = auto” setting for the monitor, Splunk will adjust the polling interval automatically based on how often it observes changes to the data, thus minimizing the impact of the monitor as much as possible.

Lookup Tables

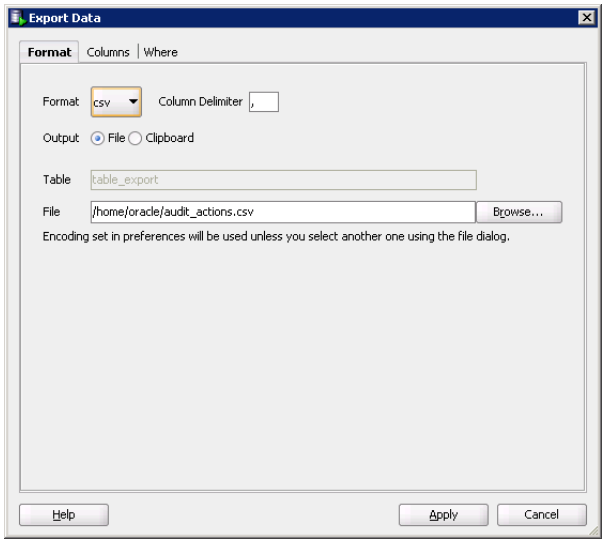
Splunk lookup tables allow me to embed useful code values into my application, to make my logs easier to interpret. For instance, Oracle uses a variety of numeric codes in auditing to identify the action being audited and the privilege used to perform the action. Oracle stores these codes and their definitions in database tables where they can be referenced by various SQL-based applications. By referencing these tables as part of the audit trail, it is possible to understand that an audit action of “1” relates to a “CREATE TABLE” statement. It is a relatively easy process to import this lookup data into Splunk, where I can use it in exactly the same way.

Audit action codes are stored in the SYS.AUDIT_ACTIONS table. Using Oracle’s SQL Developer tool, I can dump the contents of this table into a .csv file, which can be used as a reference by Splunk. To create lookup tables for audit actions and system privileges in Splunk, I complete the following steps:

1. Once connected to the database with SQL Developer, I query the data I want to appear in the audit actions lookup table.



2. To export the results to the file, I right-click on the query results and select “Export Data” and “csv…”. I fill in the file name “audit_actions.csv” and click “Apply”.



When the file is created, the contents should look something like the following example. In this case I edited the column names to match my lookup field extractions (see below) within Splunk. Splunk is case-sensitive when matching lookup filters to column names, so accuracy is important.

```
action,action_name
0,UNKNOWN
1,CREATE TABLE
2,INSERT
3,SELECT
…
```

```
privilege,privilege_name
3,ALTER SYSTEM
4,AUDIT SYSTEM
```

```
5,CREATE SESSION
6,ALTER SESSION
…
```

3. I repeat the process for the SYS.SYSTEM_PRIVILEGE_MAP table and name the file "system_privileges.csv".
4. I place both .csv files in the $SPLUNK_HOME/etc/apps/ora_11g_demo/lookups directory.
5. To make the lookup tables visible to the app in Splunk, I add the following lines to the $SPLUNK_HOME/etc/apps/ora_11g_demo/local/transforms.conf file.

```
[audit_actions]
filename = audit_actions.csv

[system_privileges]
filename = system_privileges.csv
```

Lookup tables can also be defined through DBX, using data stored in the database. Unless the lookup table is extremely large, however, I prefer to use the .csv file as it allows me to access the lookups even when the database is down.

Field Extractions

Splunk field extractions and lookups allow me to search on custom fields within my log files like userid, action name, source IP address, or returncode. Splunk will automatically identify many fields within my data as it is indexed. Additional custom fields can be defined from within the Splunk user interface by selecting "Extract Fields" from the Event Options Menu next to each search result row, or by editing the oralcle_monitoring app configuration files.

Custom field extractions are defined using Perl regular expressions, as shown below. For this project I added these basic field extraction and lookup definitions – organized by sourcetype – to the props.conf file in the $SPLUNK_HOME/etc/apps/ora_11g_demo/local directory. I have found that these fields cover most of my general needs when searching Oracle-related sources. Note that the column names in the two LOOKUP entries match the column names in my lookup *.csv files, which I created in the previous section.

```
[cman]
EXTRACT-action_name = (?im)^(?:[^\*]*\*){1,3}?\s+(?P<action_name>\w+)\s+\*
EXTRACT-os_userid = (?i)\(USER=(?P<os_userid>[^\)]+)
EXTRACT-returncode = (?im)^(?:[^\*]*\*){2,5}\s+(?P<returncode>\d+)
EXTRACT-service_name = (?i) establish \* (?P<service_name>[^ ]+)
EXTRACT-sid = (?i) service_update \* (?P<sid>[^ ]+)
EXTRACT-src_ip = (?i)\(.*?=(?P<src_ip>\d+\.\d+\.\d+\.\d+)(?=\))
EXTRACT-src_port = (?i)\(PORT=(?P<src_port>[^\)]+)
EXTRACT-src_program = (?i)\(PROGRAM=(?P<src_program>[^\)]+)

[listener]
```

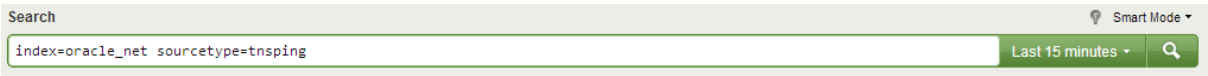
```
EXTRACT-action_name = (?im)^(?:[^\*]*\*){1,3}?\s+(?P<action_name>\w+)\s+\*
EXTRACT-returncode = (?im)^(?:[^\*]*\*){2,5}\s+(?P<returncode>\d+)
EXTRACT-service_name = (?i) establish \* (?P<service_name>[^ ]+)

[os_audit]
EXTRACT-action = (?:ACTION:\[\d+\])\s+"(?P<action>[^"]*)"
EXTRACT-comment_text = (?:COMMENT\$TEXT:\[\d+\])\s+"(?P<comment_text>[^"]*)"
EXTRACT-object_owner = (?:OBJ\$CREATOR:\[\d+\])\s+"(?P<object_owner>[^"]*)"
EXTRACT-os_userid = (?:OS\$USERID:\[\d+\])\s+"(?P<os_userid>[^"]*)"
EXTRACT-priv_used = (?:PRIV\$USED:\[\d+\])\s+"(?P<priv_used>[^"]*)"
EXTRACT-returncode = (?:RETURNCODE:\[\d+\])\s+"(?P<returncode>[^"]*)"
EXTRACT-userhost = (?:USERHOST:\[\d+\])\s+"(?P<userhost>[^"]*)"
EXTRACT-userid = (?:USERID:\[\d+\])\s+"(?P<userid>[^"]*)"
LOOKUP-audit_action_name = audit_actions action AS action OUTPUTNEW action_name AS
action_name
LOOKUP-priv_used_name = system_privileges privilege AS privilege OUTPUTNEW
privilege_name AS privilege_name
EXTRACT-session_id = (?i)^(?:[^"]*"){3}(?P<session_id>[^"]+)
EXTRACT-object_name = (?:OBJ\$NAME:\[\d+\])\s+"(?P<object_name>[^"]*)"
EXTRACT-os_userid_sys = (?i)^(?:[^\]]*\]){5}\s+'(?P<os_userid_sys>[^']+)
EXTRACT-priv_used_sys = (?i)^(?:[^\]]*\]){4}\s+'(?P<priv_used_sys>[^']+)
EXTRACT-userid_sys = (?i)^(?:[^\]]*\]){3}\s+'(?P<userid_sys>[^']+)
```

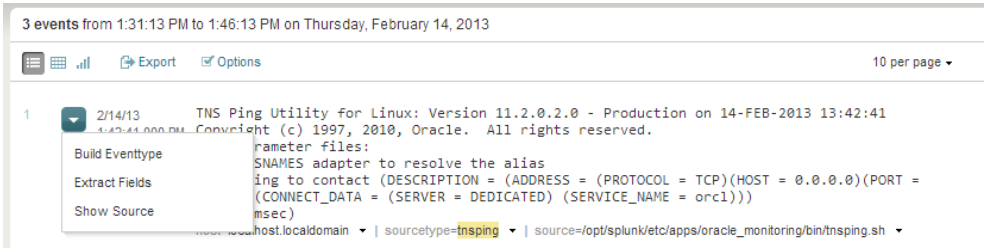
More Field Extractions

Additional fields can easily be defined through the Splunk user interface. For instance, to specify the response time output from the tnsping.sh script I created as a searchable field, I complete the following steps:

1. Splunk learns by example, so the first thing I need to do is search on the specific data that contains my new fields. In the search bar, I enter the following query:

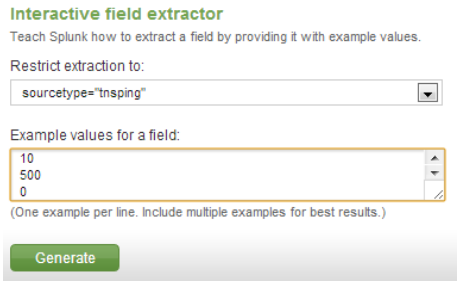


2. The search should produce at least three results, based on the schedule I set when the script monitor was created. Next to each result is an arrow for the Event Options Menu. When I click on one of the arrows next to a row (it doesn't matter which one), I select "Extract Fields".

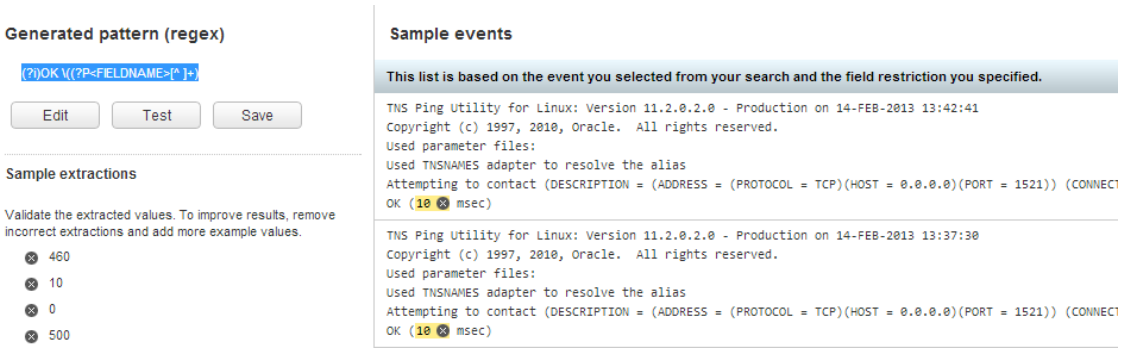


3. The Extract Fields screen will open in a new browser tab or window. It will contain a number of sample rows of data, not just the one I selected in my original results. The easiest way to define a new field is to enter representative values of the field (from the sample data shown) in the

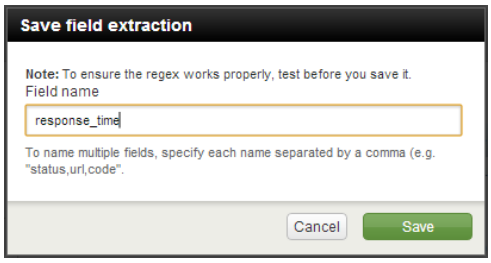
Interactive Field Extractor. When I have entered some sample values I can click the "Generate" button, and Splunk will attempt to create a Perl expression that filters out my desired field from the raw data.

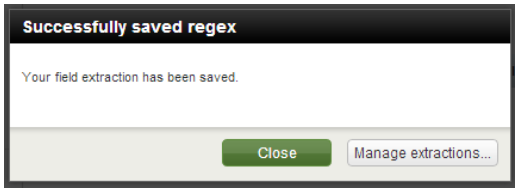


4. Splunk will show me the Perl expression, a list of additional sample values, and highlight those values within the sample data to make sure it has properly identified my field. In this case the Perl regular expression it generates is "(?i)OK \((?P<FIELDNAME>[^]+)".

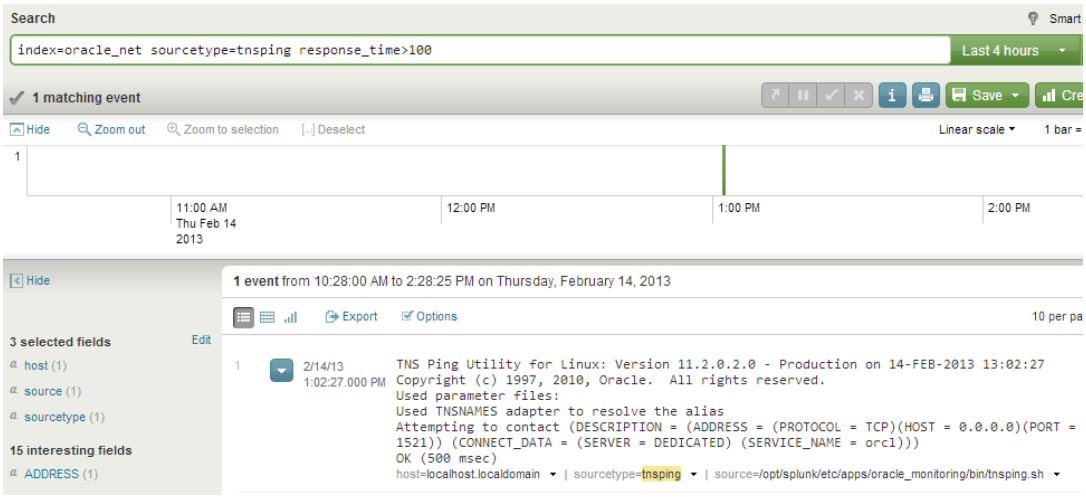


5. When I am satisfied that the pattern matching is accurate, I click the "Save" button, enter a name for my new field, and click "Save" again.





Once my new field is defined, I can use it in searches and reports. For instance, now I can search for tnsping response times that exceed a certain threshold, like 100ms. I can look for patterns in the response times and correlate them with other machine data, like CPU usage or network bandwidth. With the enterprise version of Splunk I could define an alert that would notify me automatically in the event that response time exceeded my threshold too many times.



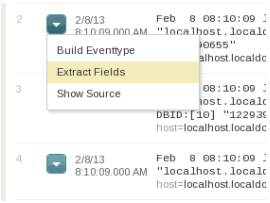
I can also turn my new field into a report that demonstrates graphically any trends or patterns in the behavior of my system, as I will show in the following section on reports.

Basic Oracle Searches and Reports

There is an entire book written on Splunk search commands and syntax, which I certainly cannot reproduce in this paper. Below I have included a series of basic searches that I have found useful over the years to serve as a starting point.

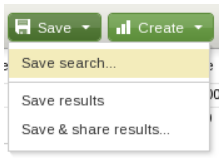
Search Name	Search Command
Audit Trail Events	(index=oracle_audit OR index=oracle_net) returncode>0
Audited Events by Host	(index=oracle_audit OR index=oracle_net) returncode>0 | timechart span=1h count by host
Database Alert Events	((index="oracle_rdbms" OR index="oracle_asm") AND "ORA-") OR (index="oracle_audit" AND (returncode=1017 OR returncode=28000 OR returncode=28001))
Database Backup Errors	index=oracle_backup ("ORA-" OR "RMAN-")
Database Restarts	index=oracle_rdbms "starting oracle instance"
DDL Events	index=oracle_audit (action_name="*create*" OR action_name="*drop*" OR action_name="*alter*")
Failed Login Attempts	index=oracle_audit sourcetype=os_audit action_name=LOGON returncode>0
Listener Connections by Host	index=oracle_net sourcetype=listener action_name="establish" | timechart count by host
Log Switch Activity by Host	index=oracle_rdbms "LGWR switch" | timechart span=15m count by host
Network Activity by Host	index=oracle_net sourcetype="listener" action_name="establish" | timechart count by host
Network Audit Events	index=oracle_net (returncode>0 OR TNS- OR ORA-)
Network Connections by Service	index=oracle_net action_name=establish | timechart usenull=false useother=false count by service_name
Network Connections by Source IP	index=oracle_net action_name=establish | timechart usenull=false useother=true count by src_ip
SYS Audit Events	index=oracle_audit sourcetype=os_audit priv_used_sys="SYSDBA"

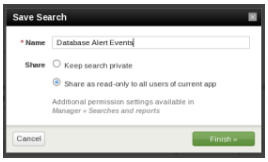
Any fields present in a particular set of search results will be listed on the left side of the display. The field list can be used to further narrow down search results by clicking on a field name, then a field value.

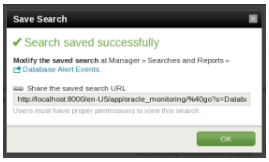


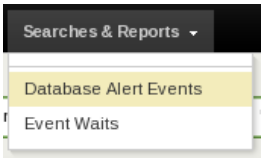
Any search can be saved as a part of the monitoring application so that I do not have to re-enter the entire search string every time I want to run it. To create a saved search, I first run the search using the "Search" window in the app.

Once the search has run, I click the "Save" button and select "Save search…" from the menu. I enter the name of the search, click "Finish" and then "Ok".



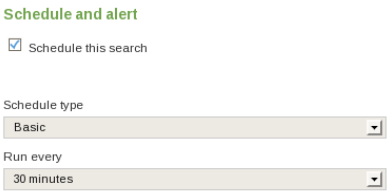






Now my search will be automatically listed in the "Searches & Reports" menu in the app, where I can select it easily.

One of the best features of Splunk is the ability to schedule a saved search to run automatically on a set schedule. Splunk caches the results of the search and serves them directly into dashboards or reports rather than run the same query multiple times. This lightens the workload on the Splunk index server considerably, and simplifies the creation of dashboards, as shown below. To schedule a search to run automatically, I select the "Manager" link at the top of the page, then "Searches and Reports". From the list of saved searches, I click the name of the search I wish to schedule, then I click the "Schedule this search" checkbox.



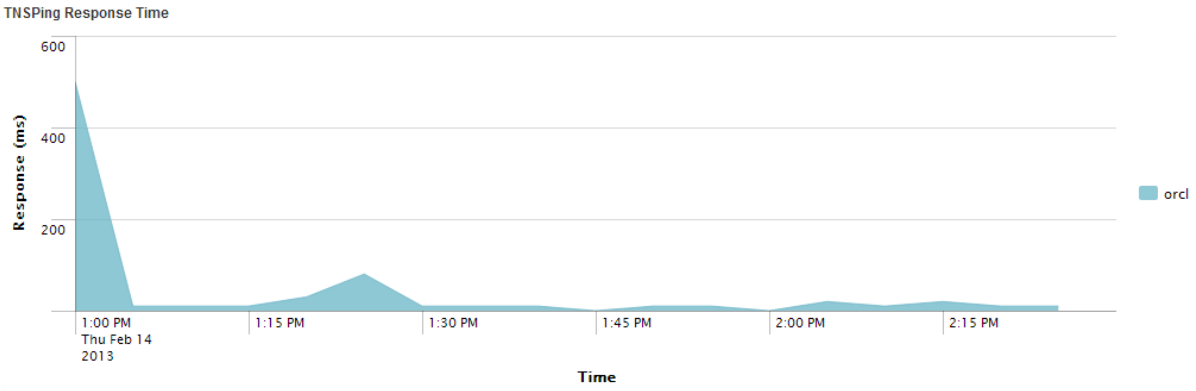
Schedules can be defined either using the "Basic" options, or using Unix "cron" style scheduling. For most searches I have found that the basic scheduling works fine. I set my schedule and click "Save". Now

when I execute this search from the app menu, Splunk will show me the cached results from the most recent scheduled run of the search.

Reports

Reports are graphical representations of saved searches, often showing data trends over time. Splunk can create a variety of charts and displays for use as stand-alone reports, or for inclusion in dashboards. To create a report, I first run a search or a saved search. Using the custom field I created earlier, I can chart the tnsping response time, and then click on the "Create" button at the right side of the search results and select "Report" from the menu.

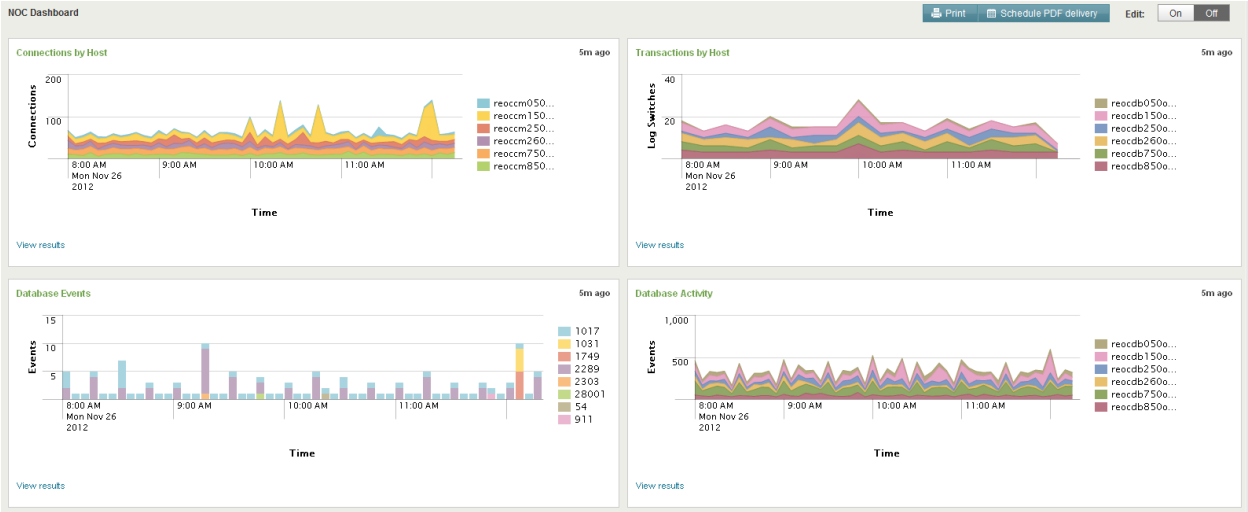
Following the instructions in the report wizard, I adjust the appearance and content of the reports, then save the finished report when I am satisfied. Saved reports can be scheduled just like saved searches, to reduce processing requirements on the Splunk indexer.



Dashboards

Dashboards are useful for customizing the display of my log data. I can use dashboards to highlight interesting and useful aspects of my logs, link to more detailed searches, and display high level reports. For example, I could build a security dashboard that highlights failed login attempts, current database connection details, client programs in use, and DDL activity. A different dashboard might highlight transaction log rollovers, CPU usage, and storage capacity metrics. Dashboards can contain forms to narrow or broaden search criteria, and can also be programmed to refresh automatically at set intervals.

A sample dashboard XML template is shown below. For a detailed description of how to build dashboards in Splunk, read the "Splunk Developing Dashboards, Views, and Apps for Splunk Web" manual.




```
<?xml version='1.0' encoding='utf-8'?>
<dashboard refresh="300">
  <label>NOC Dashboard</label>
  <row>
    <chart>
      <searchName>Net Connections by Host</searchName>
      <title>Connections by Host</title>
      <option name="drilldown">all</option>
    </chart>
    <chart>
      <searchName>Log Switch Activity by Host</searchName>
      <title>Transactions by Host</title>
    </chart>
  </row>
  <row>
    <chart>
      <searchName>Audit Trail Events by Returncode</searchName>
      <title>Database Events</title>
    </chart>
    <chart>
      <searchName>Database Audit Events by Host</searchName>
      <title>Database Activity</title>
      <option name="drilldown">all</option>
    </chart>
  </row>
</dashboard>
```

Distributed Deployment

One of the greatest benefits of Splunk is realized when it is deployed across multiple systems. Splunk includes a variety of specialized components, including search interfaces, indexers, deployment servers, and lightweight agents called “Universal Forwarders”. The indexer creates and manages indexes. In single-machine deployments consisting of just one Splunk instance (similar to the one I have demonstrated in this paper), the indexer also handles the data input and search management functions. In a distributed configuration, the data input function is often handled by a universal forwarder, installed on each monitored system, which collects data and sends it over the network to one or more

indexers (Splunk, Inc., 2013). Forwarder configurations can be updated and pushed from a centralized deployment server so that I do not spend all my time updating the same configuration files whenever I need to tweak a particular monitor. Similarly, the search function can be separated from the indexer and placed on a dedicated server, allowing a user to search data from multiple indexers in parallel from a single interface.

Whether I am monitoring a cluster of database servers or an entire application technology stack, once deployed in this manner Splunk allows me to centralize my monitoring and analysis for the entire environment. Consider the hours of painstaking analysis and line-by-line searching of multiple log files on multiple servers involved in troubleshooting user activity in a load-balanced, clustered environment with multiple application and database servers. With Splunk, I enter a single, simple web search and get complete, chronological results of user activity or events across the entire cluster within seconds. More details and best practice strategies for a distributed deployment can be found in the "Splunk Distributed Deployment Manual."

Conclusion

Whether supporting large production operations or a small development environment, it is vitally important to understand at all times what my databases are doing and why. Deploying Splunk as a core part of my monitoring infrastructure allows me to analyze security and system operations in real time by correlating data often ignored by other commercial monitoring tools. As I have shown, Splunk can be deployed quickly and easily for virtually no cost on individual servers. The simple installation and configuration lead to almost instantaneous return on investment. With its ability to search, report, and alert me to important events across multiple custom data sources, Splunk enables me to see patterns and trends in system behavior, or to find the proverbial needle in a log file haystack that might otherwise go undetected.

Deployed on a larger scale across entire computing environments, Splunk's value becomes even more apparent: database activity can be correlated directly with operating system and application activity to paint a full picture of computing operations across the entire technology stack. Consolidating this critical machine data helps to protect my databases from a variety of threats, both internal and external. When planning a well-managed, secure database network environment, Splunk is a key piece of the monitoring infrastructure.

Works Cited

Oracle Corporation. (2012, October). *Oracle Database Security Guide 11g Release 2.* Retrieved November 2012, from Oracle Documentation: http://docs.oracle.com/cd/E11882_01/network.112/e16543.pdf

Splunk, Inc. (2012, April). *Developing Dashboards, Views, and Apps for Splunk Web.* Retrieved April 2012, from http://docs.splunk.com/index.php?title=Documentation:Splunk:AdvancedDev:Whatsinthismanual:4.3&action=pdfbook

Splunk, Inc. (2012, April). *Getting Data In Manual.* Retrieved April 2012, from http://docs.splunk.com/index.php?title=Documentation:Splunk:Data:WhatSplunkcanmonitor:4.3&action=pdfbook

Splunk, Inc. (2012, December 14). *Splunk DB Connect 1.0.6: Deploy and Use Splunk DB Connect.* Retrieved December 2012, from Splunk Docs: http://docs.splunk.com/index.php?title=Documentation:DBX:DeployDBX:Abouttheconnector:1.0.5&action=pdfbook

Splunk, Inc. (2012). *Splunk Guide to Operational Intelligence.* Retrieved April 2012, from www.splunk.com: http://www.splunk.com/web_assets/pdfs/secure/Splunk_Guide_to_Operational_Intelligence.pdf

Splunk, Inc. (2013, February). *Distributed Deployment Manual.* Retrieved February 2013, from http://docs.splunk.com/index.php?title=Documentation:Splunk:Deploy:Distributedoverview:5.0beta&action=pdfbook

Software Installed

Title	Download URL
Oracle Virtual Box and System Images	http://www.oracle.com/technetwork/community/developer-vm/index.html?ssSourceSiteId=ocomen
Splunk	http://www.splunk.com/download?r=header
Splunk DB Connect	http://splunk-base.splunk.com/apps/50803/splunk-db-connect
Oracle JDBC 1.6	http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html

Biography

Peter Magee is a CompTIA Security+ certified Oracle database administrator with over twenty years of professional experience in both commercial and military computing environments. Over the years he has published and presented white papers and scripts on a variety of performance tuning and security subjects. Highlights of Mr. Magee's career include contributing content and scripts to the Defense Information Systems Agency's first Database Security Technical Implementation Guide, presentation of papers on real-time auditing and monitoring techniques using Oracle Enterprise Manager at the Independent Oracle Users Group (IOUG) Live '99 Conference, and participation as a network defender in the Air Force's "Black Demon" cyber defense exercise. His scripts and award-winning white papers on security and system monitoring have been published on the Oracle Magazine Interactive Web Site, the IOUG's "Library of Oracle Knowledge", the Sun Microsystems "Big Admin" portal, the Oracle Communities portal, and the CSC Leading Edge Forum. Mr. Magee is currently employed as a Lead Database Architect at RCF Information Systems, where he provides production operations support for a customer's 12,000 user business management system. Mr. Magee's blog, "Late Night Oracle Blog", is available at pmdba.wordpress.com.

License