MITRE Baseline Configuration System

Testing Plan

FINAL REVISION, October 27, 2008
Purdue University, CS 307, Fall 2008

Team MITRE:
Catherine Brown
Michael Dunn
Mark Nowicki
David Tittle

Table of Contents

1 Introduction

1.1 Document Scope

This document contains information regarding the testing plan developed for the MITRE XML Baseline Configuration Tool. This includes test designs and tools, as well as definitions for quality goals and measurements

1.2 Intended audience

This document is intended for the MITRE team, class supervisors, and the corporate partners.

1.3 Project Identification

This document pertains to the MITRE XML Baseline Configuration Tool, code-named Sentinel. More information about this project can be found in the High Level Design Document.

2 Software Quality Goals

2.1 Overview

The quality requirements addressed here include maintainability, readability, and extensibility as a primary quality goal for the customer. With these goals, it is also necessary for high general quality, high commenting quality and error handling routines to cover any conceivable error.

2.2 Metrics

To measure the quality of the product in terms of maintainability, readability, and extensibility, coding standards will be implemented. These standards are to be followed throughout the code, with the exception of extreme circumstance. These standards include:

- Methods contain no more than 25 lines
- Method names will include a verb and a noun that accurately describe the use of that method
- non-trivial variables* should be named accurately as to what they represent
- method names and non-trivial variable names must contain a minimum of 5 characters
- method names and non-trivial variable names cannot contain abbreviations

* Trivial variables are method specific variables that are not passed in or out of the function. Examples including: counters, iterators, and other temporary variables.

Commenting quality will be measured in two ways. The first is that any file or class must contain a header that mentions the title of the file/class, the original author and date, a brief description of its purpose, and a change log that lists all changes made to the file. Methods, very similarly, require headers that include the title, a brief description, and a note of what parameters need to be passed in, and what variables are returned.
The second measure of quality will be to have a minimum of 50% of the code in any method commented, with an additional minimum of including all the header comments.

Errors of any kind will be encapsulated by an exception handler and be dealt with accordingly.

As a measure of overall quality, production code (test code) will only be in the source control (working code). The Cyclomatic complexity per method should not exceed 6 paths. Test code coverage should be at a minimum of 80%.

These measures will ensure that the code is easily readable (with standardized naming conventions, above) and understandable (using commenting standards, above). The testing measurements will ensure that methods are not overly complex and that all situations are considered.

2.3 Tools

Find and describe tools relevant to your project that can gather data, calculate the metric, and analyze the results.

The tool that will be used primarily for code coverage and cyclomatic complexity will be PYUnit, the Python language equivalent of Java's JUnit.

Naming and commenting conventions will be measured during production and checked during the code review process.

3 Testing Processes

3.1 Overview

Testing will be done using PYUnit. This can be used by any person during the code production, but must be completed prior to code review.

3.2 Version Control

The version control used for this project is an SVN that is hosted externally. As described in part 2.1 of the Implementation Plan, the project will be hosted on Assembla.

3.3 Code Reviews

Code reviews will be performed following the submission of completed code. In addition to the code, the authors will also submit a copy of test unit results. The teams, as described in the Implementation Plan, section 3.5, will meet for the review process. The authoring team will explain their code and defend their coding choices. The other team will review the code with the authors.
If the code measures up to the quality goals set forth and is accepted by all Sentinel team members, it is considered finalized and the team will move on to the next piece of code.

3.4 Other Test Processes

PyUnit will calculate the metrics every time there is a team code review. A copy of the code will be provided to the reviewing team for analysis.

4 Test Cases

4.1 Unit Test Cases

Module or Class: **DBManager**

Test Case 001
1) Testing: connect() method
Method:
5 Instantiate DBManager with incorrect information for user, password, domain and database name
6 Call connect()
Expected Result: “False” is returned and the message “DBManager Error: connect(): Unable to connect to database: <database_error>” is printed to stderr, where <database_error> is the error returned from the database.
Cleanup: Delete DBManager object

Test Case 002
2) Testing: disconnect() method
Method:
i. Instantiate DBManager with information for user, password, domain and database name
ii. Call disconnect()
Expected Result: “False” is returned and the message “DBManager Error: disconnect(): No database connection found.” is printed to stderr.
Cleanup: Delete DBManager object

Test Case 003
prepare() → Takes in a string and makes sure characters don't interfere with the query itself, return true
3) Testing: prepare(number) method
Method:

- Instantiate DBManager with information for user, password, domain and database name
- Call prepare(number) method, where "number" is not a valid string

Expected Result: Returns "False" and the message "DBManager Error: prepare(): parameter is not a valid string." is printed to stderr.
Cleanup: Delete DBManager object

Module or Class: **SystemController**

Test Case 004
1) Testing: executeTransfer() method
Method:

i. Instantiate SystemController class
ii. Call executeTransfer()

Expected Result: Returns "True" upon completion
Cleanup: Delete SystemController object

Test Case 005
2) Testing: executeRule() method
Method:

i. Instantiate SystemController object
ii. Call executeRule()

Expected Result: Returns "True" upon completion
Cleanup: Delete SystemController object

Test Case 006
3) Testing: executeAll() method
Method:

i. Instantiate SystemController object
ii. Call executeAll()

Expected Result: Returns "True" upon completion
Cleanup: Delete SystemController object

Test Case 007
4) Testing: heartbeat() method
Method:

i. Instantiate SystemController object
ii. Call heartbeat() method

Expected Result: Returns "True"
Cleanup: Delete SystemController object

Test Case 008
5) Testing: scheduler() method
Method:
i. Instantiate SystemController object
ii. Call scheduler() method
Expected Result: Returns “True”
Cleanup: Delete SystemController object

Module or Class: **UI/Shell**

Test Case 009
1) Testing: getCommand() method
Method:
i. Instantiate UIShell object
ii. call getCommand() method
Expected Result: String is returned (possibly empty)
Cleanup: Delete UIShell object

Test Case 010
2) Testing: parseCommand() method
Method:
i. Instantiate UIShell object
ii. Call parseCommand(null) where “null” is a null reference.
Expected Result: Returns “false” and “UIShell Error: parseCommand(): Command string passed is a null reference.” is printed to stderr.
Cleanup: Delete UIShell object

Test Case 011
3) Testing: executeCommand() method
Method:
i. Instantiate UIShell object
ii. Call executeCommand() method
Expected Result: Returns “false” and “UIShell Error: executeCommand(): Command reference is null.” is printed to stderr.
Cleanup: Delete UIShell object

Module or Class: **AlertManager**

Test Case 012
1) Testing: createAlertList() method
Method:
i. Instantiate AlertManager object
ii. Call createAlertList()
Expected Result: Returns “false” if no database connection has been established. “AlertManager Error: createAlertList(): Could not connect to database.” is printed to stderr.
Cleanup: Delete AlertManager object

Test Case 013
2) Testing: formatMessage() method
Method:
i. Instantiate AlertManager object
ii. Call formatMessage() method
Expected Result: Returns “false” when no message list can be found. “AlertManager Error: formatMessage(): Message list is null.” is printed to stderr.
Cleanup: Delete AlertManager object

Test Case 014
3) Testing: sendMessage() method
Method:
i. Instantiate AlertManager object
ii. call sendMessage() method
Expected Result: Returns “false” when no message list can be found. “AlertManager Error: formatMessage(): Message list is null.” is printed to stderr.
Cleanup: Delete AlertManager object

Module or Class: **RuleManager**

Test Case 015
1) Testing: treeInit(filepath) method
Method:
i. Instantiate RuleManager object
ii. Call treeInit(filepath) where “filepath” is an invalid string (empty or numeric)
Expected Result: returns “false” and “RuleManager Error: treeInit(): Path name is invalid. Unable to open file.” is printed to stderr.
Cleanup: Delete RuleManager object

Test Case 016
2) Testing: checkDatabase(client, filepath) method
Method:

i. Instantiate RuleManager object
ii. call checkDatabase(client, filepath) where “client” is an invalid reference to a client id, and “filepath” is an invalid string (empty or numeric).

Expected Result: returns “false” and “RuleManager Error: checkDatabase(): Path name is invalid or client id is invalid.” is printed to stderr.
Cleanup: Delete RuleManager object

Test Case 017
3) Testing: checkFile(filepath) method
Method:

i. Instantiate RuleManager object
ii. call checkFile(filepath) where “filepath” is an invalid string (empty or numeric).

Expected Result: returns “false” and “RuleManager Error: checkFile(): Path name is invalid. Unable to open file.” is printed to stderr.
Cleanup: Delete RuleManager object

Test Case 018
4) Testing: checkClient(client) method
Method:

i. Instantiate RuleManager object
ii. call checkClient(client) where “client” is an invalid id or non-numeric

Expected Result: returns “false” and “RuleManager Error: checkClient(): Client id is invalid.” is printed to stderr.
Cleanup: Delete RuleManager object

Test Case 019
5) Testing: checkSchema(schemapath) method
Method:

i. Instantiate RuleManager object
ii. call checkSchema(schemapath) where “schemapath” is an invalid string (empty or numeric)

Expected Result: Returns “false” and “RuleManager Error: checkSchema(): Path name is invalid. Unable to open file.” is printed to stderr
Cleanup: Delete RuleManager object

Test Case 020
6) Testing: getRuleList(schemapath) method
Method:
i. Instantiate RuleManager object
ii. call getRuleList(schemapath) where “schemapath” is an invalid string (empty or numeric)
Expected Result: Returns “false” and “RuleManager Error: getRuleList(): Path name is invalid. Unable to open file.” is printed to stderr
Cleanup: Delete RuleManager object

Test Case 021
7) Testing: createAlert(rule, client) method
Method:
i. Instantiate RuleManager object
ii. call createAlert(rule, client) where “rule” is an invalid pointer to a rule object, or client is an invalid client id (non-numeric or invalid index)
Expected Result: Returns “false” and “RuleManager Error: createAlert(): Null rule reference, or invalid client id.” is printed to stderr
Cleanup: Delete RuleManager object

Module or Class: **Rule**

Test Case 022
1) Testing: getClientInfo(client) method
Method:
i. Instantiate Rule object
ii. call getClientInfo(client) where “client” is an invalid client id (non-numeric or invalid index)
Expected Result: Returns “false” and “Rule Error: getClientInfo(): Invalid client id.” is printed to stderr
Cleanup: Delete Rule object

Module or Class: **XMLTree**
-libparse() → takes in a reference to an xml file and returns a dom tree

Test Case 023
1) Testing: libparse(filepath) method
Method:
i. Instantiate XMLTree object
ii. call libparse(filepath) where “filepath” is an invalid string (empty or numeric)
Expected Result: Returns “false” and “XMLTree Error: libparse(): File path is invalid. Unable to open file.” is printed to stderr
Cleanup: Delete XMLTree object

Module or Class: **TransferManager**

Test Case 024
1) Testing: getFiles() method
Method:
i. Instantiate TransferManager object
ii. call getFiles()
Expected Result: Returns “false” when one or more files cannot be retrieved. Prints “TransferManager Error: getFiles(): Could not retrieve files for every client.” is printed to stderr
Cleanup: Delete TransferManager object

Test Case 025
2) Testing: saveFiles() method
Method:
i. Instantiate TransferManager object
ii. call saveFiles()
Expected Result: Returns “false” when one or more files cannot be saved to their appropriate paths. Prints “TransferManager: saveFiles(): Unable to save all files to appropriate paths.” is printed to stderr.
Cleanup: Delete TransferManager object

Module or Class: **FileRetriever**

Test Case 026
1) Testing: connect(client) method
Method:
i. Instantiate FileRetriever object
ii. call connect(client) where “client” is an invalid client id (non-numeric or invalid index)
Expected Result: returns “false” and “FileRetriever Error: connect(): Invalid client id.” is printed to stderr.
Cleanup: Delete FileRetriever object

Test Case 027
2) Testing: close(client) method
Method:
i. Instantiate FileRetriever object
ii. call close(client) where “client” is an invalid client id (non-numeric or invalid index)
Expected Result: returns “false” and “FileRetriever Error: close(): Invalid client id.” is printed to stderr.
Cleanup: Delete FileRetriever object

Test Case 028
3) Testing: fetch() method
Method:

i. Instantiate FileRetriever object
ii. call fetch()

Expected Result: Returns “false” when a transfer is does not complete successfully. Prints “FileRetriever Error: fetch(): File transfer was interrupted and did not complete.”
Cleanup: Delete FileRetriever object

Test Case 029
4) Testing: fetchFileInfo() method
Method:

i. Instantiate FileRetriever object
ii. call fetchFileInfo()

Expected Result: Returns “false” when information for a file cannot be fetched. Prints “FileRetriever Error: fetchFileInfo(): Cannot retrieve file information.” is printed to stderr.
Cleanup: Delete FileRetriever object

4.2 Integration Test Cases

Integration Testing can easily be executed in the XML Baseline Configuration System, since it has a System Controller Component which can monitor and execute any other system component individually. With this being said, the following Integration Test Cases will be either separate or combined test executions of the System Controllers built-in functionality.

Test Case 1001
System: XML Baseline Configuration System Phase: 1
Start XML Baseline Configuration System
Severity: 3
Instructions:

1. At the console, enter: python *sentinel*

Expected Result:

1. The System should report a successful startup of the XML Baseline Configuration System with the following message: *Sentinel Is Up and Running... Scheduled Time for Collection and Comparison: <Time>*. Where the Time is determined by the user when the automation process should begin.

Cleanup:
None

Test Case 1002
System: XML Baseline Configuration System Phase: 1
Transfer XML File from Client
Severity: 1
Instructions:

1. At the console, enter: transfer *n*
2. With *n* being the number of xml files tested

Expected Result:

1. The System should report a successful file transfer: *XML File(s) successfully transferred*

Cleanup:

1. None

Test Case 1003
System: XML Baseline Configuration System Phase: 1
Parse and Compare Given XML File with Baseline
Severity: 3
Instructions:

1. At the console, enter: compare *xmlfilepath*

Expected Result:

1. The System should report a successful file comparison, by either
    a. Displaying any Alert Messages if crucial difference was found: *Difference was found: <Alert Message>*
    b. If no crucial differences were found: *No Differences Found*

Cleanup:
None

Test Case 1004
System: XML Baseline Configuration System Phase: 1
Format Alert Messages and Send E-Mails
Severity: 2
Instructions:

1. At the console, enter: alert *xmlfilepath*

Expected Result:

1. If successful, the System should report with either of the messages:
    a. If crucial differences are found: *E-Mail Message sent to intended recipient*. From there the administrator can double check to make sure e-mail was received.
    b. If crucial differences were not found: *E-Mail Message not sent. Error: No crucial differences found!*

Cleanup:
None

Test Case 1005
System: XML Baseline Configuration System Phase: 1
Execute Entire System
Severity: 3
Instructions:

1. At the console, enter: runall *n*
2. With *n* being the number of xml files tested

Expected Result:

1. The System should report a successful system execution with the following message: *System Executed Successfully*
2. If there was an error in a specific subsystem, the System will report that in the following error message: *System Error: <Specific Component> Failed*

Cleanup:
None

Test Case 1006
System: XML Baseline Configuration System Phase: 1
Database Connection Test
Severity: 2
Instructions:

1. At the console, enter: dbtest

Expected Result:

1. The System should report a successful database connection and display the following message: *Database Connection Successful – Retrieved This Value: <value>*
    a. The value will be an arbitrary value stored in the database.

Cleanup:
None

Test Case 1007
System: XML Baseline Configuration System Phase: 1
XML Baseline Configuration System Shutdown
Severity: 2
Instructions:

1. At the console, enter: shutdown

Expected Result:

1. If the XML Baseline Configuration System shutdown successfully, the System should report with the following message: *Sentinel Shutdown Sequence Completed – System Now Offline*

Cleanup:
None

4.3 System or User Interface Test Cases

Test cases for entire XML Baseline Tool system:

#	Description	Pass	Fail
Preconditions: System is up and running on server.			
1	Time for scheduled run of XML Baseline Tool arrives. Multiple XML files have a recorded change to system configuration that they represent which requires an e-mail alert. Result: Alerts are sent out to the appropriate e-mails.		
2	Add a new XML file is added for a new client that has no previous XML baseline record. Run XML Baseline Tool. Result: E-mail Alert is sent out to the appropriate e-mail regarding this new client.		
3	Delete an XML file that the application is supposed to harvest and parse for system configuration changes. Run XML Baseline Tool. Result: E-mail Alert is sent out informing the appropriate e-mail address that an expected file is missing.		
4	Change an XML file so that it has an invalid format. Run XML Baseline Tool. Result: E-mail Alert is sent out informing the appropriate e-mail address that a file is corrupt or is of an invalid/unrecognized format.		
Preconditions: User is logged in to shell access point and has proper credentials.			
5	Enter Baseline command and choose server 1 workstation 1. Set baseline date equal to the date of the last running of the application. Make sure, and if not then change the file so that, XML file for server 1 workstation 1 is identical to the new baseline, but has changed since the old baseline. Result: No Alert is sent out.		
6	Enter Heartbeat command Set date for heartbeat to today's date. Run XML Baseline Tool. Result: "Heartbeat e-mail" sent out to appropriate e-mail address with information that the system is still functioning properly.		
7	Enter Add/Remove client command. Add a new client to the list of expected clients. Add a new XML file for the new client (the one added in previous action). Run XML Baseline Tool. Make alert-requiring changes to the XML file. Run XML Baseline Tool. Result: Baseline for new client should be set to first XML file upon the first running of the application in this test. Alerts should be sent out to the appropriate e-mail address regarding the changes in second running		

8	Enter Add/Remove client command. Remove a client from the list of expected clients. Make sure that XML file for the removed client is still produced (and has alert-requiring changes) in its location where it would normally be harvested. Result: No Alert is sent out because that XML file is no longer harvested.		
9	Run tests 1, 2, 3, 4, 7 above. Enter "Log File" command. Result: Log file is displayed with all alerts sent out.		
10	Enter "Run Module" command. Choose Transfer Manager. Result: XML files transferred to main server.		
11	Run test 10. Enter "Run Module" command. Choose Rule Manager. Result: XML files are parsed and Alerts that are to be sent out are stored in the database.		
12	Run tests 10, 11. Enter "Run Module" command. Choose Alert Manager. Result: Alerts are sent out to the appropriate e-mail addresses.		