Wifiology Test Plan

Robert Cope, Jason Nguyen, Baiyu Chen,
Jasper Niemeyer, Peng Jiang, Ryan Campbell
Group: 404 Group Not Found

April 14, 2019

1 Test Plan Overview

1.1 Application Description

The Wifiology application is a wireless profiling and monitoring tool, which utilizes physically colocated nodes actively listening to 802.11 (WiFi) traffic, in conjunction with a central service to aggregate the collected data. Our service allows users to deduce current wireless congestion, watch for rogue access points, gauge the number of connected users at any physical location, and in large deployments, help users find locations that aren't overcrowded and have reasonable wireless networking performance. Our application is a combination of Python on the physical nodes themselves, with a centralized NodeJS server aggregating data.

1.2 Target Users

For this application, we plan to have two sets of target users. The first set of users are owners and administrators of wireless networks, the second set of users being the actual consumers of the wireless networks. Owners and administrators will purchase this service to help better understand their wireless network user experience and to augment the process of administering and policing spectrum; once installed the second set of users can use the service to plan visits to locations with 802.11 hotspots and check for poor connectivity.

To help guide our user acceptance testing, we plan to find a group of early adopter network admins, for whom we will offer the service at a discount or for free; in return we will gather their feedback and guidance in reviewing our user acceptance tests and providing feedback on the deployed application.

1.3 Test Execution Team

For the execution of our manual UAT tests, the group has elected Robert Cope and Jasper Niemeyer to fulfill the role of manual tester in executing our tests.

The automated tests, as described below will be executed as part of our continuous integration/continuous deployment pipeline. The role of test script developer for end-to-end automated user acceptance tests so far has been conducted by Robert Cope, but has been agreed to be conducted by Jason and Ryan as well.

2 User Acceptance Testing

2.1 Tools

For the implementation of our automated UTA test cases, which will interact with the both the API and our web user interface, we use the Python "unittest" framework in conjunction with Selenium webdriver to automate a user interacting with our web service. We current record results by hand and store our test plans this document, but plan make use of the qTest tool in the near future, for automated recording of both manual and automated test cases and their results.

2.2 Feature: Export Data To CSV

2.2.1 Description

In order to facilitate external analysis and data recording, our tools allows administrators to download the collected and aggregated data from their nodes in CSV format. This is done by logging into the application, selecting the node the user is interested in, and clicking a button to trigger the creation of a CSV from data stored in our database.

Our BDD statement for this test is as follows: "As an admin consuming the Wifiology service, I want to be able to download the raw data my nodes have been collecting, so that I can perform additional analyses and import my data into other tools to facilitate my network administration job."

2.2.2 Entry Criteria

To begin user acceptance testing, a build of the central server code base must be available and be free of syntax errors. This build must also have passed all of our unit and integration tests, as described below. Data will be populated from a sample set, described below, and the application will be started locally for the user acceptance tester. Additionally the final entry criterion for this is that an additional document is produced explicitly detailing how to conduct the manual test for this feature. Once this document is completed, we will examine how much of this can be converted to Selenium based automated tests; needless to say, this additionally gates the automated testing portion of our test plan.

2.2.3 Exit Criteria

The exit criteria for this test is that the tester is successfully able to log in, generate a valid CSV file from the data stored in the database and download the data locally, and that the CSV data matches the expected values given knowledge of the prepopulated data.

2.2.4 Test Preparation

To facilitate consistency in testing of this feature, the application developers and test scripter(s) will generate a prerecorded set of sample data, and create a script to update a target database with this information. Prior to the test being executed, this script will be run to prepopulate the database with this information. The server will then be started for the user to begin testing.

2.2.5 Manual Testing

In order to conduct the manual testing of this feature, the tester will do the following:

- The manual tester point their browser to the supplied instance of the application.
- The tester will login using the provided credentials, and verify that they are able to successfully log in.
- The tester will use the application menu to browse to the listener node of interest.
- The tester will click the button for CSV download.
- The tester will verify that the downloaded CSV is a valid format, and may be opened using a common spreadsheet tool like Excel.
- The tester will verify that the information stored in the CSV matches the expected values.

Additionally the tester should score the following subjective questions as satisfactory or unsatisfactory, and provide feedback if unsatisfactory:

- Did I find the process of finding where to download the CSV data I needed intuitive?
- Did the CSV data include all of the fields of data I was interested in?

If the tester is able to complete all of these, the test will be considering passing; any failure in this process will mark this test as failed. If the test fails, a bug will be opened against the product for the developers to analyze the failure and rectify the code base.

2.2.6 Automated Testing

Automated testing for this feature will be identical to the manual test plan above, with the exception being that a script is generated to perform this test using the Selenium webdriver toolkit. The test will still be performed manually as a gate to release, but this automated UAT test will provide a means to save tester time.

2.3 Feature: Show Aggregate 802.11 Traffic on an Individual Node

2.3.1 Description

This feature allows both wireless network administrators to gauge the amount of traffic (and hence congestion and potentially oversubscription) for a given physical proximity around a listener node. Administrators will be able to see the number of devices (and their MAC addresses) that have been seen recently, and with what access points have they been interacting with.

Our BDD statement for this test is as follows: "As a network admin consuming the Wifiology service, I want to be able to see the amount of 802.11 traffic and the devices creating this traffic, so that I can gauge the current and historic utilization of my wireless spectrum."

2.3.2 Entry Criteria

To begin user acceptance testing, a build of the central server code base must be available and be free of syntax errors. This build must also have passed all of our unit and integration tests, as described below. Data again will be populated from a sample set, described below, and the application will be started locally for the user acceptance tester. As before, the final entry criterion for this is that an additional document is produced explicitly detailing how to conduct the manual test for this feature, once completed.

2.3.3 Exit Criteria

The exit criteria for this test is that the tester is successfully able to log in, see a detailing listing of the current traffic, as well as browse historic traffic data for the historic window of their choosing.

2.3.4 Test Preparation

Again, to facilitate consistency in testing of this feature, the application developers and test scripter(s) will generate a prerecorded set of sample data, and create a script to update a target database with this information. Prior to the test being executed, this script will be run to prepopulate the database with this information. The server will then be started for the user to begin testing.

2.3.5 Manual Testing

In order to conduct the manual testing of this feature, the tester will do the following:

- The manual tester point their browser to the supplied instance of the application.
- The tester will login using the provided credentials, and verify that they are able to successfully log in.
- The tester will use the application menu to browse to the listener node of interest.
- The tester will examine the page to see the "latest" traffic report, and verify that the data is intelligible.
- The tester utilize an input on the page to select a given historic range at random.
- The tester will make the service execute the historic lookup for the range specified and again verify the data is intelligible.

Additionally the tester should score the following subjective questions as satisfactory or unsatisfactory, and provide feedback if unsatisfactory:

- Did I find the process of finding where the node data page intuitive or possible?
- Was I able to input the historic range of interest with relative ease?
- Did data read out include all of the information I was interested in?

If the tester is able to complete all of these, the test will be considering passing; any failure in this process will mark this test as failed. If the test fails, a bug will be opened against the product for the developers to analyze the failure and rectify the code base.

2.3.6 Automated Testing

Some portions of this, once the we demonstrate the ability to execute the test manually, will be converted to automated Selenium testing (though this will not replace the manual tests). We expect our Selenium tests to also perform the same checklist above, logging in, checking for predefined data, and verify that the correct data fields are present on the page.

2.4 Feature: Show Discovered 802.11 Access Points/Service Sets

2.4.1 Description

The final feature for this UAT test plan to the ability for our admin users to see all "service sets" (colloquially knowns as "wireless networks") that have been seen broadcasting their existence within listening distance of a deployed node. This feature enable network admins to track rogue access points and clamp down on forms of unauthorized access.

Our BDD statement for this test is as follows: "As a network administrator, I want a listing of access points or service sets that have been discovered near my listener nodes, so that I can clamp down on rogue access points, and ensure secure network access."

2.4.2 Entry Criteria

To begin user acceptance testing, a build of the central server code base and the listener node code base must be available and be free of syntax errors. Both builds must also have passed all of our unit and integration tests, as described below. Because data will not be prepopulated for this test, the tester or any supporting staff must stand up both the listener and server, configure the listener with the service, and deploy a new "rogue" access point in the vicinity of the listener node. As before, the final entry criterion for this is that an additional document is produced more explicitly detailing how to conduct the manual test for this feature, once completed; once this document is completed.

2.4.3 Exit Criteria

The exit criteria for this test is that the tester is successfully able to log in, and see the "rogue access" point in the listing of access points/service sets discovered by the given listener node.

2.4.4 Test Preparation

In order to allow the testing to occur the tester will have both a on the shelf wireless access point (like those commonly available at electronics stores), as well as a physical listener node with a working build of the listener node code base. The listener node will be turned on, and configured to talk to an instance of the central server, which has been deployed for this test. The tester will place the access point and listener node within relative proximity of each other (no close than 3 feet, but no further than 100 feet), and let data collection occur for at least 30 minutes.

2.4.5 Manual Testing

In order to conduct the manual testing of this feature, the tester will do the following:

- The manual tester point their browser to the supplied instance of the application.
- The tester will login using the provided credentials, and verify that they are able to successfully log in.
- The tester will use the application menu to browse to the listener node of interest.
- The tester will examine the page for the latest service sets seen report.
- The will verify that the deployed "rogue access" point is listed in the report.

Additionally the tester should score the following subjective questions as satisfactory or unsatisfactory, and provide feedback if unsatisfactory:

- Did I find the process of finding where the node data page intuitive or possible?
- Did I find all of the data I needed about the service sets on this page?

If the tester is able to complete all of these, the test will be considering passing; any failure in this process will mark this test as failed. If the test fails, a bug will be opened against the product for the developers to analyze the failure and rectify the code base.

2.4.6 Automated Testing

We do not foresee any automated testing being utilized for user acceptance testing on this feature.

3 Unit and Integration Testing

3.1 Listener Node Code Base

The Python code base for the remote nodes which listen for 802.11 traffic will continue to utilize unit and integration tests written in the Python "unittest" framework. We aim to have 80-90% code coverage for this code base. We require new tests be written for all new feature work introduced to this code bases.

3.2 Central Server Code Base

Our NodeJS code base, which is the implementation of the central server and web service for Wifiology, also has unit and integration tests. We utilize the "Mocha" and "Chai" test framework tools to provide both unit and integration tests on our NodeJS code base. We also aim to have 80-90% code coverage for our tests.