Risk Management Report

# **Create GUI**

**Likelihood:** Since it is likely that most of the team knows how to build a GUI using java and is has some knowledge of the MVC pattern this task should not be too difficult. However, this is not something we do all the time so we may be rusty and need to refresh our brains as to the exact steps.

**Severity:** Since the client is requesting a ‘useful UI’ it is mandatory that this task is completed.

**Consequences:** The GUI provides an easy way for the user to connect to the router and also view the data. Therefore, the GUI needs to function correctly or there will be a huge negative impact on the user experience.

**Workarounds:** As an alternative we could provide a command line interface so that the client could view data.

**Difficulty:** It should be fairly easy to provide a program that runs in a command line console.

**Impact:** The overall program will seem less polished but overall it will have the same usability as the GUI version.

**Pros:** Slightly easier to implement than GUI version

**Cons:** The customer is likely to be very unhappy with a command line program.

# **Program Router Related Functionality**

**Likelihood:** Connecting to a router and capturing data is something no one on our team has programed before. This task may be difficult unless we find a library or framework which will facilitate such a task.

**Severity:** This project is hinged on this feature being operational. It is mandatory for the success of the project.

**Consequences:** Without this feature there is no point in completing other features in this project such as a data parser because there will be no data to parse.

**Workarounds:** One possible work-around is to use existing software such as NMAP or Wireshark to connect to the router, capture data, and save to a temporary buffer file.

**Difficulty:** It may be moderately difficult to implement pipes or OS system calls to make this work, especially on two different operating systems. It’s difficult to gauge how much extra work this will cause if any since this task is something we have never done before. It’s more important how quickly our team recognizes that we need to switch to this work-around and begin implementing it. If we recognize early that the work-around is necessary, then the delay this causes should only be approximately a week or less.

**Impact:** There will be no impact on the user unless the user has similar software to compare the performance of our software with. This is because the work-around may make our application run slower.

**Pros:** Using the work-around makes it possible for our team to complete the project.

**Cons:** Using the work-around will require installing the software that our application depends on to work. It will have to be incorporated with application installer which will need to identify the OS and install the appropriate version. This is likely going to prove to be difficult since none of us has ever programmed an installer before.

# **Create Data Parse and Filter Functionality**

**Likelihood:** Since most of the team has likely programmed some sort of parser before this task may not be extremely difficult, but it will not be easy either. Part of what will determine how difficult this task is, is how ‘dirty’ the data is. Well-formed data containing a limited and known character set should make this task easier.

**Severity:** This task is very important since the user will likely not be happy viewing the raw data captured from the router. However, viewing raw data is better than viewing no data at all.

**Consequences:** If this task is not completed the user will have to view the raw data captured from the router. Capturing the raw data from the router is one of the subtasks of the previous main task.

**Workarounds:** One possible partial work around is to use existing software that makes filtering data easier. Also, we could write scripts which use grep (or something similar) and pipe the output to our program.

**Difficulty:** If we are going to use pre-existing software in tandem with our program, it should be installed along with our software, hidden and only available to the process our application runs in. This will minimize the weakening effect this software dependency has on our application.

**Impact:** Doing the work around should not affect the user experience unless the user has other software to compare our program to, as it is likely that ours will run slower given the additional software dependency layer.

**Pros:** If the data is extremely ‘dirty’ and not well formed the work-around may make accomplishing this task possible.

**Cons:** Introducing software dependencies will likely complicate installing the software on the user’s machine and will cause our program to rely on system calls to invoke the dependent software when necessary.

# **Compute statistical data and display to user**

**Likelihood:** No one in our group has any experience with Shannon’s Entropy algorithm but I’m sure this is something we could research and learn. Displaying the statistical data in text boxes or a table should be fairly easy. However, if the user requires fancy graphical visuals of the data in the form of charts and graphs, then leveraging existing graphic libraries or frameworks should help make this task easier.

**Severity:** Without this feature the software is useless.

**Consequences:** The user will be unhappy and will likely never speak to us again ; )

**Workarounds:** If the user wants fancy graphics, as mentioned above, we can try to find software libraries that make this easy.

**Difficulty:** The learning curve of the software libraries should not be too difficult.

**Impact:** The overall program will function as if there was no work-around.

**Pros:** Slightly easier to implement using a library designed for displaying graphics.

**Cons:** We may have to switch programing languages if we cannot find a graphics library for the existing programming languages.