CS Senior Design Specification Report

Port Scanner

Matthew Enea, William Deming

Dr. Xiang Fu

Spring 2018

5/18/2018

**Abstract**

Due to the coming of the digital age when all of our important information is on the internet the importance of cybersecurity has risen sharply. Ports act as a kind of doorway to your network and the outside network which means poor security of ports are a glaring vulnerability that hackers can take advantage of. Port scanners combat this vulnerability but each one out has their own problems. To address these problems a new port scanner was created with the goals to better protect these doorways such as building the scanner with the C language utilizing libraries that give control over the base level packets for more flexibility and adding a monitoring component to constantly scan over ports looking for any unauthorized access. The flexibility of this port scanner allows the user to change settings at the port level meaning the user chooses the state that each port should be at and sends alerts to the user if the states are wrong. Flexibility also includes ease of use with a streamlined GUI. The monitoring component will keep a constant watch over the ports so the user will not have to manually scan to find out if any ports states are open when they are suppose to be closed. This port scanner adds stronger protection in the battle against hackers and keeping important information safe.

***Table of Contents***

Page 5 – 1. Project Scope and Vision

Page 6 – 2. Project Goals

Page 6 – 3. Requirement Specification

Page 7 – 3.1 System Perspective

Page 8 – 3.2 Use Cases

Page 18 – 3.3 External Interfaces

Page 19 – 3.4 Functional Requirements

Page 20 – 3.5 Non-functional Requirements

Page 22 – 4. Engineering Standards and Constraints

Page 22 - 4.2 Engineering Standards

Page 22 – 4.3 Realistic Constraints

Page 23 – 5. Project Design

Page 23 – 5.1 Overview of System Components

Page 24 – 5.2 Structure and Relationship

Page 27 – 5.3 User Interface

Page 31 – 5.4 Detailed Component

Page 38 – 5.5 Reuse and Relationships with Other Products

Page 38 – 5.6 Design Decisions and Tradeoffs

Page 38 – 5.7 Resource List

Page 39 – 5.8 Resource Course List

Page 39 – 6. Project Plan

Page 39 – 6.1 List of Tasks

Page 43 – 6.2 Project Schedule

Page 43 – 6.3 Risk Plan

Page 44 – 6.4 Estimated Financial Budget

Page 44 – 6.5 Teamwork Plan

Page 44 - 7. Test Plan

Page 44 - 7.1 Test Plan

Page 45 - 7.2 Test Cases

Page 48 - Test Schedule

Page 48 - 8. Project Implementation & Results

Page 48 - 8.1 Implementation Strategy

Page 49 - 8.2 Sprints

Page 51 - 8.3 Results

Page 56 - 8.4 Testing Results

Page 57 - 9. Problems & Changes

Page 57 - 9.1 Design Changes

Page 58 - 9.2 Sprint Changes

Page 59 - 9.3 Final Results

Page 60 - 10. Risks

Page 60 - 11. Financial Budget

Page 60 - A1. Project Societal Concerns

Page 60 - A2. References

1. **Project Scope and Vision**

Hacking in the 21st century has been refined to where it can almost be considered an art form. Attackers use a variety of tactics and tools to plan digital heists on their victims, where one simple security slip-up can mean the end for a company. The solution therefore, is to think like the hacker, and ensure all holes are plugged. One of the biggest security holes are open network ports, entry points into internal networks that can be accessed from the Internet side. Hackers occasionally find these open ports to be running outdated or unpatched software, which they then exploit by using a tool and/or method known for breaching that version of the software, thus gaining illegal access to the network.

Port scanners or vulnerability scanners are the common solution to fixing problems with open ports. They check each port on every node in the network, and provide the user with an assessment of what isn’t secure. Nmap is by far the best and most well-known port scanner, with others including Angry IP Scanner, Superscan, NetScanTools, and Unicornscan. However, none of these are quite perfect: Nmap is the most robust scanner but lacks any additional functionality; Angry IP Scanner is outdated and has trouble running on newer Windows versions; Superscan is closed-source and not well-maintained; NetScanTools port scanner must be purchased in a bundle with other tools; Unicornscan is known for being very difficult to use.

Our project aims to create a port scanner that will ensure open ports on the system are functioning as they should and inform the user when they are malfunctioning and/or being used when they should not be. The scanner will use socket programming as Nmap does to determine if open ports are being used to access the system without authorization[1].

The scanner will be created from scratch without use of existing tools except for C language libraries that will be used to communicate with the system at a base level, making it possible to receive responses from the different sockets and formulate a response for the user of the system. We aim to make our scanner different by offering additional functionality such as adding in a monitor process that keeps information in a database and sends alerts if any ports are newly opened or changed.

1. **Project Goals**

The goals for our project include:

1. Develop a port scanner that checks an internal network for open ports
2. Create a network of virtual machines to act as a testing environment
3. Create a graphical user interface for the port scanner
4. Set-up the monitoring process and database
5. **Requirements Specification**

The main features of the port scanner include:

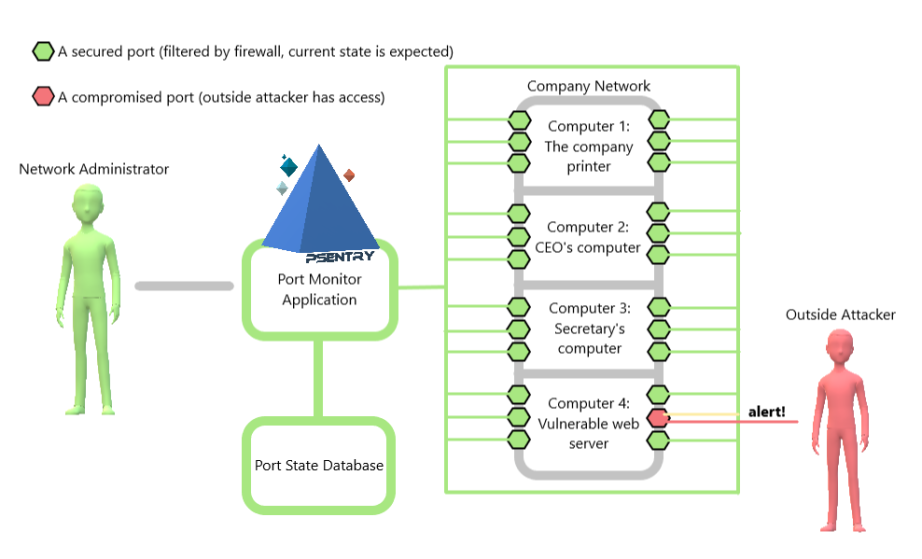
* Running a port scan on an internal network
  + Identifying the state of a port
  + Reacting to the state of a port
  + Logging the state of a port
* Compiling information from a scan into a report
* Monitoring ports
  + A database will hold information on current ports
  + Alerts on new ports/changes
  + Settings can be changed to only alert specific changes/ports

The port scanner will operate by attempting to initiate connections with ports on an internal network. It will identify the state of these ports by how they respond: if the connection gets dropped, accepted, and/or if a response is given. Depending on the state of the port, different actions will be taken: the scanner will check the response (if one was received) and probe further if possible, or it will move on to the next port.

As the port scanner connects to ports, it will be logging which ports drop connections, which ports react and how they react. This information will be compiled into a report which can then be saved or sent to others via email.

The port scanner will also have a monitoring process which will keep information of current ports in a database. Any changes will be alerted to the administrator and will then be logged back into the database. Setting will be available to filter through common/meaningless changes/ports to avoid clutter.

* 1. **System Perspective**



*Figure 1: System Overview*

Figure 1 is a diagram showing how the port scanner would be used by a network administrator working for a small company. The network administrator’s job is to ensure all nodes are properly connected to the internal network, as well as protect the nodes and internal network from attackers.

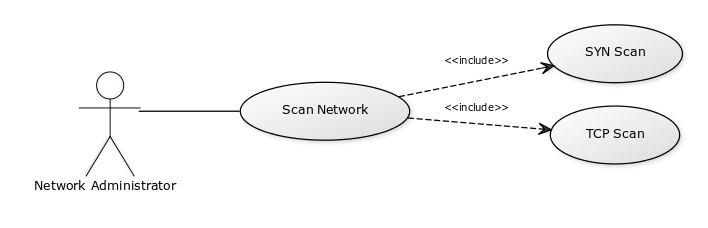
A node is any device requiring internet or communication with other devices, e.g. printers, personal computers, servers, and network devices such as switches and routers. The internal network is a term used to describe the union of the various connections between all nodes in the company’s building, including wired and wireless connections. An attacker is defined as any person or group looking to gain unauthorized access to the internal network or nodes.

In Figure 1, the network administrator uses the port scanner application on the administrator’s computer. The port scanner is used to determine which nodes on the network have open ports, a common vulnerability that attackers exploit to gain unauthorized access to the network. After a scan, the network administrator can then read the report and take the necessary steps to fix any discovered vulnerabilities. The port state database is connected to the monitoring process of the port scanner and keeps track of any changes over the entire network.

**3.2 Use Cases**

Use cases for the port scanner are grouped into two categories: Scanning use cases, which relate to running scans and using output from scans, and Utility use cases, which relate to the more miscellaneous features of the application.

*3.2.1 Scanning Use Cases*



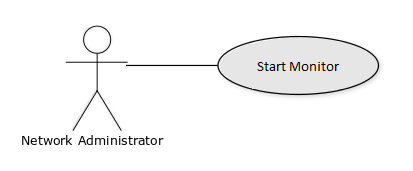
*US-1: Scan Network*

|  |  |
| --- | --- |
| **Name** | US-1: Start Scan |
| **Description** | The user selects which types of scans they wish to run, and then clicks “Scan Network” to execute. |
| **Rationale** | Start a network scan. |
| **Users** | Network Administrator |
| **Preconditions** | The user has selected at least one type of scan to run. |
| **Basic course of events** | 1. The user will open the user interface and be prompted by the home screen to select a scan type. 2. The user selects which scans they wish to run, each scan will have a short description informing the user that more information on the scans can be found on the help page. 3. The user then clicks “Scan Network” button located at the bottom of the user interface. 4. Upon the scan completing, the user will receive a notification from the application. |
| **Alternative paths** | If the scan fails, the application will display an error message to the user. |
| **Postconditions** | The scan is completed and a log of information is displayed. |



*US-2: Specify Ports*

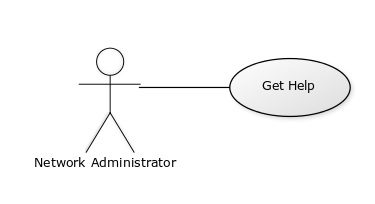
|  |  |
| --- | --- |
| **Name** | US-2: Specify Ports |
| **Description** | The user adds ports to the XML file |
| **Rationale** | Specify which ports to scan. |
| **Users** | Network Administrator |
| **Preconditions** | None. |
| **Basic course of events** | 1. The user edits the XML file to add ports. 2. The application saves the entered port numbers and only scans the entered ports. |
| **Alternative paths** | None. |
| **Postconditions** | The port numbers are saved. |



*US-3: Start Monitor*

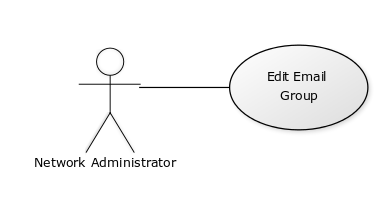
|  |  |
| --- | --- |
| **Name** | US-3: Start Monitor |
| **Description** | The user starts the port monitor |
| **Rationale** | Start the monitoring |
| **Users** | Network Administrator |
| **Preconditions** | None. |
| **Basic course of events** | 1. The user opens the user interface and clicks on the monitor tab. 2. The user can select to turn on or off timestamps and logging of ports. 3. Once options are as desired the user can start the monitor. 4. The user can stop the monitor at any time. |
| **Alternative paths** | None. |
| **Postconditions** | Ports are no longer updated. |

*3.2.2 Utility Use Cases*



*US-4: Get Help*

|  |  |
| --- | --- |
| **Name** | US-4: Get Help |
| **Description** | The user clicks the “Help” button. |
| **Rationale** | Open the Help document. |
| **Users** | Network Administrator |
| **Preconditions** | The help document has not been moved from the program directory. |
| **Basic course of events** | 1. The user clicks the Help button. 2. The linked document is opened from the programs directory. 3. The user can navigate the help document by sections, subsections, and scrolling. |
| **Alternative paths** | If the help document cannot be found, the application will display an error message to the user. |
| **Postconditions** | The help document is opened. |



*US-5: Edit Email Group*

|  |  |
| --- | --- |
| **Name** | US-5: Edit Email Group |
| **Description** | The user clicks the “Edit Email Group” button. |
| **Rationale** | Edit emails, which reports will be sent to. |
| **Users** | Network Administrator |
| **Preconditions** | None. |
| **Basic course of events** | 1. The user clicks the Edit Email Group button. 2. The application opens a textbox and prompts the user to enter email addresses. 3. The user enters email addresses into the textbox and clicks “Save.” 4. The application saves the currently entered email addresses to a file. |
| **Alternative paths** | None. |
| **Postconditions** | The email addresses are saved. |



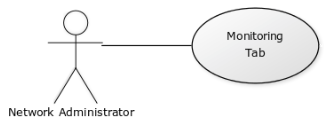
*US-6: Send Report*

|  |  |
| --- | --- |
| **Name** | US-6: Send Report |
| **Description** | The user sends a report of the scan to an email group. |
| **Rationale** | Email reports to an email group. |
| **Users** | Network Administrator |
| **Preconditions** | The user has at least one email address set in the email group; the user has run a scan. |
| **Basic course of events** | 1. The administrator clicks the “Send Report” button. 2. The application creates an email, attaches the report to it, and sends it to the specified emails. |
| **Alternative paths** | If there are no emails set in the email group, a report will not be sent.  If a scan has not been run, a report will not be sent. |
| **Postconditions** | The report is sent. |



*US-7: Save Report*

|  |  |
| --- | --- |
| **Name** | US-7: Save Report |
| **Description** | The user names and saves the report. |
| **Rationale** | Save reports for later reference. |
| **Users** | Network Administrator |
| **Preconditions** | None. |
| **Basic course of events** | 1. The administrator clicks the “Save Report” button. 2. The application opens a textbox and prompts the user to enter a name. 3. The user clicks “Save.” 4. The application saves the report to the program directory. |
| **Alternative paths** | If a scan has not been run, a report will not be saved. |
| **Postconditions** | The report is saved. |



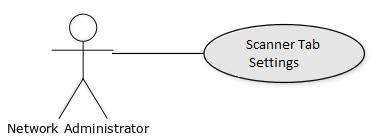
*US-8: Monitoring Tab*

|  |  |
| --- | --- |
| **Name** | US-8: Monitoring Tab |
| **Description** | Check on the monitoring state. |
| **Rationale** | Check for any issues/alerts; To see the overall status of the network. |
| **Users** | Network Administrator |
| **Preconditions** | The monitoring process is set up and turned on. |
| **Basic course of events** | 1. The user clicks on the “Monitoring Tab” 2. The status tab is displayed |
| **Alternative paths** | None |
| **Postconditions** | None |

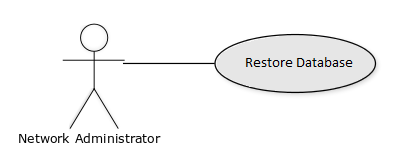


*US-9: Monitoring Tab Settings*

|  |  |
| --- | --- |
| **Name** | US-9: Monitoring Tab Settings |
| **Description** | Modify the Monitoring Tab |
| **Rationale** | Allows user to customize the tab to preferences |
| **Users** | Network Administrator |
| **Preconditions** | The monitoring process is set up and turned on. |
| **Basic course of events** | 1. In the Monitoring Tab click on Log changes 2. In the Monitoring Tab click on Include timestamps |
| **Alternative paths** | None |
| **Postconditions** | The monitoring tab is changed accordingly. |



|  |  |
| --- | --- |
| **Name** | US-10: Scanner Tab Settings |
| **Description** | Modify the Scanner |
| **Rationale** | Allows user to choose the scan |
| **Users** | Network Administrator |
| **Preconditions** | No scan is currently running |
| **Basic course of events** | 1. In the scanner tab click SYN Scan 2. In the scanner tab click ACK Scan |
| **Alternative paths** | None |
| **Postconditions** | The monitoring tab is changed accordingly. |



*US-10: Monitoring Tab Settings*

|  |  |
| --- | --- |
| **Name** | US-11: Restore Database |
| **Description** | Restore the mySQL database to empty tables |
| **Rationale** | Allows user to clear out stored computers and ports |
| **Users** | Network Administrator |
| **Preconditions** | mySQL has portscan database present |
| **Basic course of events** | 1. The user clicks the Database tab on the menu bar. 2. The user clicks Restore Database. 3. The application restores the database, clearing all data and creating new tables in portscan. |
| **Alternative paths** | None |
| **Postconditions** | The database tables are empty. |

**3.3 External Interfaces**

|  |  |
| --- | --- |
| **Name of input/output** | Input 1: XML file |
| **Source of input or destination of output** | Scanner inputs |
| **Purpose** | Receives data from user for database |
| **Processing (if needed)** | None |
| **Valid range, accuracy, and/or tolerance** | 99.90% accuracy |
| **Units of measure** | Correct computer name, network, and ports. |
| **Timing** | None |
| **Relationships to other inputs/outputs** | Accessed by port scanner |
| **Data formats (if applicable)** | XML |

**3.4 Functional Requirements**

The following are functional requirements

|  |  |
| --- | --- |
| Name | FR-1 Check for database updates |
| Summary | The system checks for changes in the database |
| Rationale | Security systems must always be up-to-date |
| System Behavior | The system checks to see if the database has had any changes made. If there are changes, the program updates to the current version |
| Reference | N/A |

|  |  |
| --- | --- |
| Name | FR-2 Connection to network |
| Summary | Checks for a network to perform the scan on |
| Rationale | Before a scan can be started, a connection with an internal network must be established |
| System Behavior | The system checks available networks, and connects to the indicated one. |
| Reference | US-1, US-2, US-3 |

|  |  |
| --- | --- |
| Name | FR-3 Scan Network |
| Summary | Able to scan a given network for vulnerabilities |
| Rationale | The software is made to scan networks |
| System Behavior | The system goes through all ports found in the internal network it is ran |
| Reference | US-1, US-3 |

|  |  |
| --- | --- |
| Name | FR-4 Find Ports |
| Summary | Uses user input to find ports in a network to check for vulnerabilities |
| Rationale | The system needs a target for a port scan |
| System Behavior | The system checks each computer on the network for currently used ports |
| Reference | US-2 |

|  |  |
| --- | --- |
| Name | FR-5 Database storage system |
| Summary | Inputs data into the database |
| Rationale | The scan needs to read info about the base computer and target ports and their states. |
| System Behavior | The system writes to the database to save the information for future reference |
| Reference | US-1, US-2, US-3, US-5 |

|  |  |
| --- | --- |
| Name | FR-6 Send Email |
| Summary | Send an email to specific addresses with the scan reports |
| Rationale | For network or security teams working in large companies, being able to quickly send reports of issues found by a scan is helpful |
| System Behavior | The system sends out emails to specific addresses |
| Reference | US-5, US-6 |

|  |  |
| --- | --- |
| Name | FR-7 Alert |
| Summary | Alerts user of events from monitoring system |
| Rationale | Allows the user to know of important events if the user isn’t in the monitoring tab |
| System Behavior | The system sends an alert regarding information collected from monitoring |
| Reference | US-3, US-8, US-9 |

**3.5 Non-functional Requirements**

**3.5.1: Performance Requirements**

The program will only be accessed by one user at any given time. The response time for major features like loading information for the scan should take no less than 5 seconds, depending on the strength of the internet connection to the database which is assumed to be strong. The system will need to be able to handle one scan at a time as well as a constant monitoring state on the ports of the computers on the network.

**3.5.2: Logical Database Requirements**

The database stores each computer being scanned and gives them a unique id. Each entry has the unique id, computer ip, and the network the computer is on. Each port is stored along with the unique computer id to associate them together. The port table also includes the current status and the expected status of the port. A history entry is added to the database for each port to keep track of the changes. Then there is a table for email groups.

**3.5.3: Engineering Requirements**

Each client will need a space of 1024MB on the database as it will log scan information, reports, and information on the current state of each computer and their ports.

**3.5.4: Security Requirements**

This program accesses important information on a network but does not directly tamper with anything. As a result it can be ran by anyone but it will alert any IPS running on the network so the providers will need to be notified of any checks being done.

**3.5.5: Reliability Requirements**

The system should work without failure if the user does not enter bad data. The monitoring function should not come into conflict with any anti-virus programs and interfere with accessing the internet.

**3.5.6: Robustness**

The scan should come back accurate every time, with no failures unless the user enters bad data. The monitoring function should run without problem in the background without causing problems for the network.

**4.0 Engineering Standards and Constraints**

**4.1 Engineering Standards [2]**

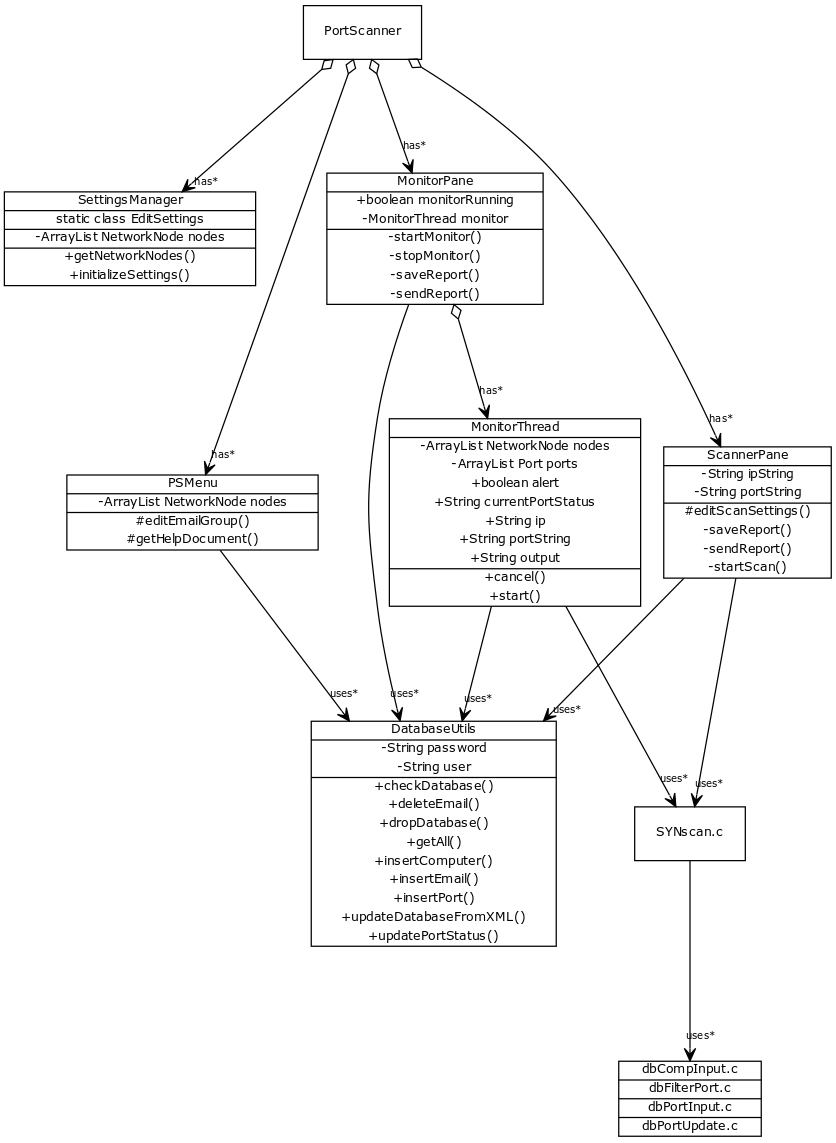
The port scanner will make use of the following network communication protocols:

* Ethernet
* Internet Protocol (IP)
* Transmission Control Protocol (TCP)

**4.2 Realistic Constraints**

Although the intended use of this tool is for assessing networks for vulnerabilities, it could also be used by attackers for discovering and exploiting vulnerabilities. Another constraint is that port scanning is illegal in some jurisdictions.  A specific real world issue is the alarms that go off due to an internal scan. The IPS company monitoring the network of the customer would have to be told in advance that a port scan is occurring and the IP of the machine being used. If the company is not alerted the machine being used is at risk of being blocked out of the network. However, since none of these concerns are completely overarching, they don’t hinder the development or design of the port scanner.

**5. Project Design**

**5.1 Overview of System Components**

*Figure 2: Class Diagram*

The PortScanner is the main class of the system. It handles preliminary tasks and starts the SettingsManager, PSMenu, ScannerPane, and MonitorPane classes. PortScanner uses SettingsManager class to ensure needed files are initialized. The PSMenu class creates the MenuBar for the application and handles user functions for updating and restoring the database, as well as editing users in the emailgroup. The ScannerPane class handles functions related to launching scans for individual ip addresses, editing settings for these scans and creating reports from output. The MonitorPane class handles functions related to the port monitor, which cycles through known computers on the network and ensures ports are not vulnerable to attack.

**5.2 Structure and Relationship**

**5.2.1 PortScanner**

The highest overarching class in the structure, the PortScanner class is responsible for launching the SettingsManager, PSMenu, ScannerPane, and MonitorPane classes, which create the application interface and handle backend functions. The PortScanner class ensures that SettingsManager runs the initializeSettings script to ensure that needed files are present, and if not creates them(FR-1, FR-2).

**5.2.2 SettingsManager**

The SettingsManager class is part of the PortScanner application class, and is responsible for ensuring needed classes are present as well as retrieving elements from the settings.xml file. The settings.xml file contains user-inputted information including computers (“network nodes”), their ports, number and expected status of the ports. It transmits this information back to PortScanner, which in turn gives it to the PSMenu class for handling the UpdateDatabaseFromXML function(US-2, FR-4).

**5.2.3 PSMenu**

The PSMenu class is part of the PortScanner application class, and handles creation of the menubar for the application and functions for all items within it. These include database update and restore options, editing users in the email group, editing settings and opening the help document(US-4, US-5, US-11, FR-1).

**5.2.4 SYNscan.c**

The SYNscan.c program is used to run scans of network computers, and is used by MonitorThread and ScannerPane classes. The program code uses the C libraries libnet and libpcap, for constructing network packets and receiving network packets respectively. It uses functions from the libraries to check if the ip address is reachable, construct TCP and IP packet headers, send the packet and retrieve the response, and then determines from the response whether the port is open (connections accepted), filtered (connections accepted, but contents filtered by firewall), or closed (connections are denied/dropped). It relays this output back to the MonitorThread and ScannerPane classes. (US-1, US-3, FR-2, FR-3, FR-4, FR-5)

**5.2.5 ScannerPane**

The ScannerPane class is part of the PortScanner application class, and is responsible for displaying the graphical user interface of the scanner tab as well as functions pertaining to running probing scans for individual computers. These include editing scan settings, running scans, and creating and sending reports of its output. The ScannerPane displays network info of the host computer, and displays all output from a port scan. ScannerPane uses the static class Edit Settings from SettingsManager to keep track of the currently entered ip address and ports.(US-1, US-2, US-6, US-7, US-10, FR-3)

**5.2.6 MonitorPane**

The MonitorPane class is part of the PortScanner application class, and is responsible for displaying the graphical user interface of the monitor tab as well as functions pertaining to the port monitor. These include starting and stopping the monitor, which runs through the class MonitorThread, as well as creating and sending reports of its output. The MonitorPane in particular will display alerts from the monitor (ports that are open or whose expected status does not equal the returned status from MonitorThread), as well as computers present in the database (computers being monitored)(US-2, US-3, FR-2, FR-3, FR-4, FR-5, FR-7).

**5.2.7 MonitorThread**

The MonitorThread class is part of the MonitorPane class, and is responsible for running the monitor loop. The monitor loop consists of several steps, detailed as follows:

1. Checking database to ensure portscan database is present
2. Retrieving all computers and ports from database as an ArrayList of NetworkNode objects
3. While the thread hasn’t been interrupted, run the monitor loop
   1. For each NetworkNode, retrieve its ip address and concatenate its ports into a single string
   2. Run a ScanThread with ip address and port string as arguments, capture the output
   3. Parse the output into an ArrayList of output port objects
   4. For each port, compare the output port status to expected status of the port
      1. If the output status and expected status are not equal, generate an alert and update the status of the port in the database

(US-2, US-3, FR-3, FR-4, FR-5, FR-7)

**5.2.8 DatabaseUtils**

The DatabaseUtils class is responsible for all database requests coming from classes of the PortScanner application class (Java backend). These include the following:

* Checking if the portscan database and related tables exist
* Dropping the portscan database
* Deleting and inserting email addresses for users in the email notification group
* Inserting computers and ports
* Retrieving all computers and ports
* Updating the database with nodes returned from SettingsManager (which retrieves nodes from the settings.xml document)
* Updating status of individual ports

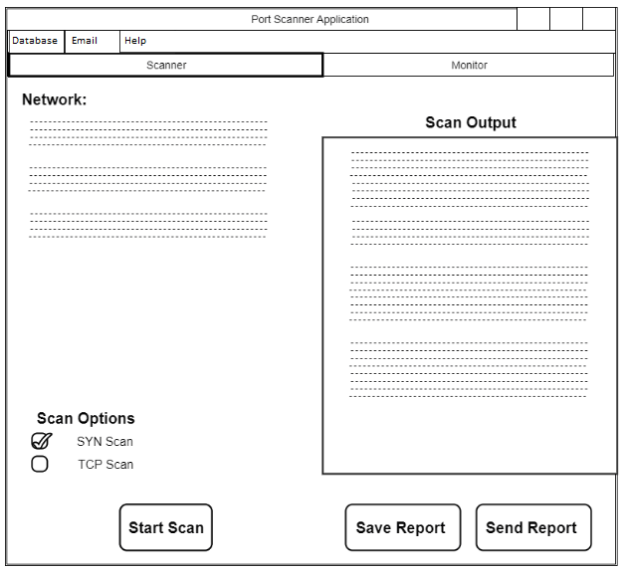
(FR-1, FR-5)

**5.2.9 db Files**

The C database files are used by the SYNscan.c program to perform low-level database updates while the scan is running. These include inserting computers and ports that are currently not present in the database, and updating status of ports. (FR-5)

**5.3 User Interface**

**5.3.1: Main – Scanner Tab**

****

*Figure 3: Scanner Tab*

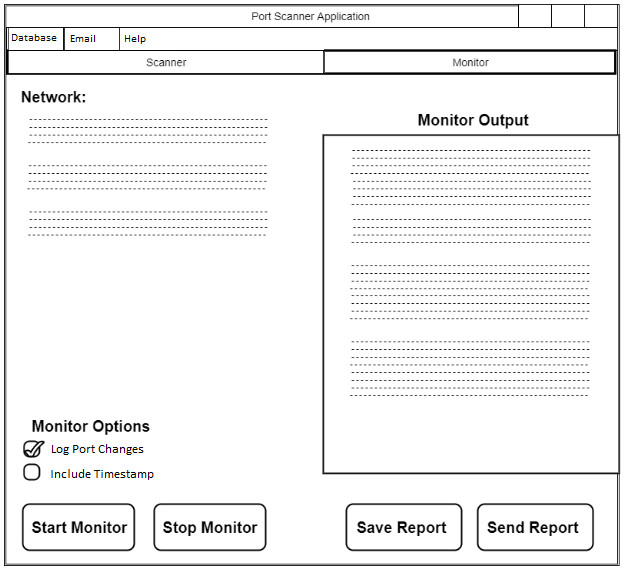
The Scanner tab of the application displays information about the network, a scan output log that details results from scans, a checklist of options for scanning, and buttons for starting scans (US-1), saving scan reports (US-7), and sending scan reports (US-6).

The ports button opens a dialog that allows the user to edit port numbers/ranges to be scanned (US-2).

Above the tab bar are three additional tabs: database (US-11), email (US-5), and help (US-4), which each open their own windows.

Clicking the Monitor tab will switch the application to the Monitor display (US-8).

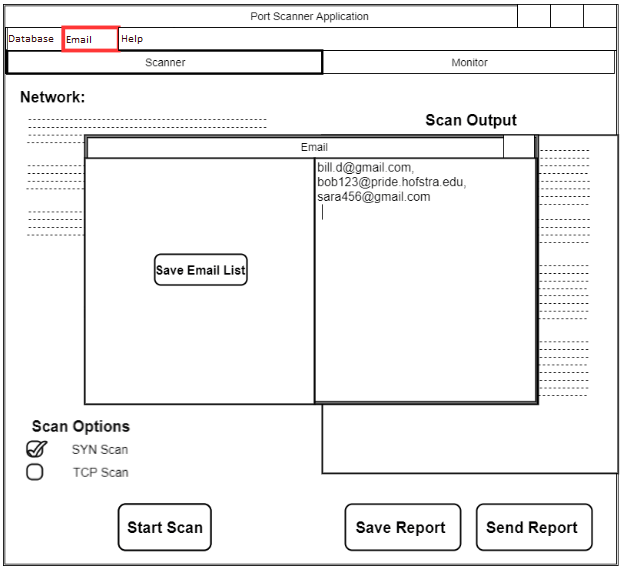
**5.3.2: Main – Monitor Tab**



*Figure 4: Monitor Tab*

The Monitor tab also displays network information, as well as options for the network monitor, buttons for starting and stopping the monitor, and buttons for saving and sending the monitor report (US-7, US-8).

**5.3.3: Email Window**

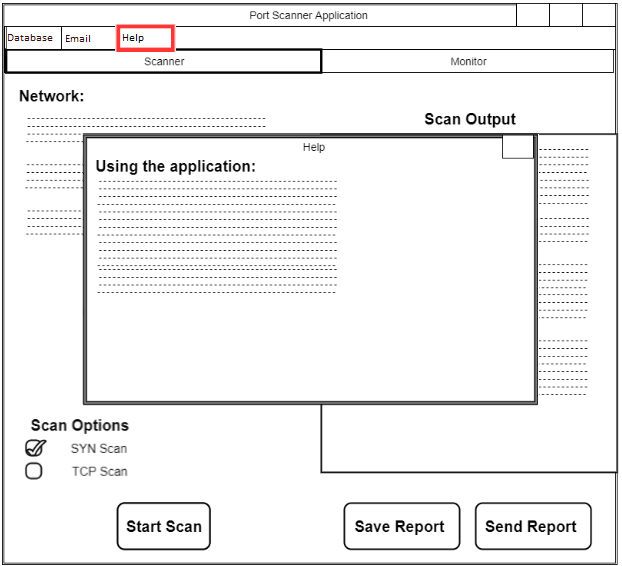


*Figure 5: Email Window*

The Email window displays all emails in the email list, a textbox that the user can edit and a button that will save all emails in the textbox as the current email list (US-5).

The emails in the list are the addresses that will receive reports when Send Report button is clicked (US-6).

**5.3.4: Help Window**

****

*Figure 6: Help Window*

The Help window will display a text document detailing how the application is used (US-4).

**5.4 Detailed Component Description**

**5.4.1 PortScanner**

|  |  |
| --- | --- |
| **Identification** | SC-1 PortScanner |
| **Type** | Main Component |
| **Function** | Initializes program. Controls all sub classes and starts up MonitorPane and ScannerPlane |
| **Dependencies** | FR-1, FR-2 |
| **Interfaces** | First there are three buttons on the top left: Database, Email, and Help. Then there are two main tabs Scanner and Monitor. The scanner tab leads to the ScannerPane. The Monitor tab leads to the MonitorPane. |
| **Resources** | Bandwidth  CPU  Memory  Ports |

**5.4.2 SettingsManager**

|  |  |
| --- | --- |
| **Identification** | SC-2 SettingsManager |
| **Type** | Sub Component |
| **Purpose** | Allow freedom and ease of access for user |
| **Dependencies** | US-2, FR-4 |
| **Interfaces** | There are two different setting menus, one for PortScannerApplication and one for NetworkMonitor. For the first, there will be fields to adjust the range of the scan such as what ports to hit and what computers to scan for said network selected. For second, there will be fields to set what situations warrant for an alert and/or email as well as any scheduled downtime or frequency. There will also be settings to determine which ports on which computers for said network will be recorded |
| **Resources** | Memory |
| **Data** | Kilobytes worth of data saved to the database. |

**5.4.3 PSMenu**

|  |  |
| --- | --- |
| **Identification** | SC-3 PSMenu |
| **Type** | Sub Component |
| **Purpose** | Initializes menu for use |
| **Dependencies** | US-4, US-5, US-11, FR-1 |
| **Interfaces** | On the top of the program there will be three tabs: Database, Email, and help. |

**5.4.4 SYNscan.c**

|  |  |
| --- | --- |
| **Identification** | SC-4 SYNscan.c |
| **Type** | Sub Component |
| **Function** | Scans specified ports |
| **Dependencies** | US-1, US-3, FR-2, FR-3, FR-4, FR-5 |
| **Resources** | Bandwidth  CPU  Memory  Ports |
| **Processing** | Using the libnet and the pcap libraries, packets will be built and sent to ports on each computer on the network and the responses will be used to decide the state of each port. The basic process of every check will be: establish a connection, build the packet, build the ipv4, build the ethernet, write the packet to the wire, and get response. |
| **Data** | The main form of data is packets. Packets are sent to a port and a response will be received as a packet. |

**5.4.5 ScannerPane**

|  |  |
| --- | --- |
| **Identification** | SC-5 ScannerPane |
| **Type** | Main Component |
| **Function** | Scans specified ports |
| **Dependencies** | US-1, US-2, US-6, US-7, US-10, FR-3 |
| **Interfaces** | There will be a start, pause, cancel button centered in the window under a progress bar There will be a help button in the top right of the window. There will be stats displayed such as current network connected, current computer focused on, and current port being scanned. |
| **Resources** | Bandwidth  CPU  Memory  Ports |
| **Processing** | Using the libnet and the pcap libraries, packets will be built and sent to ports on each computer on the network and the responses will be used to decide the state of each port. The basic process of every check will be: establish a connection, build the packet, build the ipv4, build the ethernet, write the packet to the wire, and get response. |
| **Data** | The main form of data is packets. Packets are sent to a port and a response will be received as a packet. Other data will be the logging of each port responding and the settings of the scan saved to the database. |

**5.4.6 MonitorPane**

|  |  |
| --- | --- |
| **Identification** | SC-6 MonitorPane |
| **Type** | Main Component |
| **Function** | Connects the monitor to the settings |
| **Dependencies** | US-2, US-3, FR-2, FR-3, FR-4, FR-5, FR-7 |
| **Interfaces** | The top left will have a database button, an email button, and a help button. There will be stats on the current monitoring state, red for closed ports, green for open ports. The ports will be categorized under each computer in tabs. If a flagged event occurs an alert will be shown in a pop up, alerting a system user while also sending an email out. |
| **Resources** | Bandwidth  CPU  Memory  Ports |
| **Processing** | The scan takes input from the pane. Then the scan outputs results to be displayed on the pane. |
| **Data** | The data is transferred by command line arguments and strings |

**5.4.7 MonitorThread**

|  |  |
| --- | --- |
| **Identification** | SC-7 MonitorThread |
| **Type** | Main Component |
| **Function** | Monitors the network for changes |
| **Dependencies** | US-2, US-3, FR-3, FR-4, FR-5, FR-7 |
| **Resources** | Bandwidth  CPU  Memory  Ports |
| **Processing** | Puts the SYNscan.c component in a loop for constant running. |
| **Data** | The main form of data is packets. Packets are sent to a port and a response will be received as a packet. Other data will be the logging of any changes and flags set. |

**5.4.8 DatabaseUtils**

|  |  |
| --- | --- |
| **Identification** | SC-8 DatabaseUtils |
| **Type** | Sub Component |
| **Purpose** | SQL style database that stores settings, presets, and changes to ports |
| **Dependencies** | FR-1, FR-5 |
| **Resources** | Memory |
| **Data** | Bytes(kilo, mega, giga) |

**5.4.9 db Files**

|  |  |
| --- | --- |
| **Identification** | SC-9 db Files |
| **Type** | Sub Component |
| **Purpose** | Takes output from C file to database |
| **Dependencies** | FR-5 |
| **Resources** | Memory |
| **Data** | Bytes(kilo, mega, giga) |

**5.5 Reuse and Relationship With Other Products**

The sub components, SYNscan.c is coded in C with the help of two open source libraries, libnet and pcap. These two libraries deal with creating, sending, and receiving packets at a basic level which allows us full control over the components in the project. The open source MySQL database is used to store the information collected from the two main components.

**5.6 Design Decisions and Tradeoffs**

The decision to use C over Java, C++, C# and any other language was due to the libraries libnet and pcap. These two libraries allow a full control over the basic level of creating, sending, and receiving packets allowing a more robust project will more possibilities. The trade off is the ease of creating the components and the amount of basic knowledge needed to utilize the libraries to their full capabilities. The decision to use java to create a GUI instead of sticking with C is due to ease of use, available libraries, and experience on our part. The flow of the program would be better with a java GUI.

**5.7 Resource List**

**5.7.1 Libraries**

* Libnet - A C library made to build packets. Free on github
* Libpcap - A C library made to receive packets. Free on github

**5.7.2 Database**

* MySQL Database - An open-source database by oracle.

**5.7.3 Texts**

* TCP/IP Guide A Comprehensive, Illustrated Internet Protocols Reference by Charles M. Kozierok - Ebook $79.95 on nostarch.com

**5.7.4 People**

* Matthew Enea - Lead Programmer. 8 hours a week
* William Deming - Lead Programmer. 8 hours a week
* Dr. Fu - Project Advisor. Weekly meetings for 10-30 minutes.

**5.8 Resource Course List**

* Software Engineering
* Operating Systems
* Network Security
* Ethical hacking

**6.0 Project Plan**

**6.1 List of Tasks**

|  |  |
| --- | --- |
| Name | T1 - Revise final report from Fall semester |
| Description | Revise final report based on recommendations & missing material |
| Work Effort | 4 hours |

|  |  |
| --- | --- |
| Name | T2 - Research Connector C database driver |
| Description | Research how Connector C database driver is used |
| Work Effort | 1.5 hours |

|  |  |
| --- | --- |
| Name | T3 - Set up Connector C database driver |
| Description | Set up connector C library for running database operations in C |
| Work Effort | 3 hours |

|  |  |
| --- | --- |
| Name | T4 - Build C database prototype/schema |
| Description | Create a database with tables that will hold the port states of individual computers. |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T5 - Test C database prototype |
| Description | Create test code for searching and inserting records, ensuring Connector C is working properly |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T6 - Create header files & library for C database scripts |
| Description | Create library for database scripts for ease of access from scanner, monitor |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T7 - Research libnet and libpcap C libraries |
| Description | Research the libraries to determine their effectiveness in port scanning application |
| Work Effort | 3 hours |

|  |  |
| --- | --- |
| Name | T8 - Set up libnet and libpcap libraries on test system |
| Description | Install libnet and libpcap for use on test system |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T9 - SYN scan |
| Description | Set up SYN scan script for scanning ports for a single IP address |
| Work Effort | 4 hours |

|  |  |
| --- | --- |
| Name | T10 - Revise SYN scan: port ranges |
| Description | Revise and test SYN scan code so that it can take a specified range of ports to scan. |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T11 - Make skeleton for C monitor |
| Description | Create skeleton for C monitor |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T12 - Create README file for project |
| Description | Create README that describes project and libraries used in project |
| Work Effort | 0.5 hours |

|  |  |
| --- | --- |
| Name | T13 - Add Multithreading |
| Description | Modify SYN scan code to perform each scan in a thread. |
| Work Effort | 3 hours |

|  |  |
| --- | --- |
| Name | T14 - Research demonstration for final presentation |
| Description | Research how to perform hack that will open ports for final presentation |
| Work Effort | 8 hours |

|  |  |
| --- | --- |
| Name | T15 - Create final presentation |
| Description | Create final presentation slides to showcase how the application works |
| Work Effort | 4 hours |

|  |  |
| --- | --- |
| Name | T16 - Set up Virtual Machines |
| Description | Set up virtual machine network to test application on |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T17 - Set up GitHub for version control |
| Description | Set up a GitHub repository for managing version control of application |
| Work Effort | 1 hour |

|  |  |
| --- | --- |
| Name | T18 - Designate needed classes for GUI |
| Description | Create list of needed classes for implementing UI design from Section 5.3 (tabs, new windows, radio buttons, buttons). |
| Work Effort | 1 hours |

|  |  |
| --- | --- |
| Name | T19 - Create GUI skeleton |
| Description | Create skeleton code for GUI application (PortScanner, PSMenu, ScannerPane, MonitorPane). |
| Work Effort | 1 hour |

|  |  |
| --- | --- |
| Name | T20 - Set up GUI labels, buttons, textboxes |
| Description | Create code for GUI application (labels, buttons, textboxes, menus) |
| Work Effort | 4 hours |

|  |  |
| --- | --- |
| Name | T21 - Create icons for menu buttons |
| Description | Find icons that can be used for menu functions |
| Work Effort | 0.5 hours |

|  |  |
| --- | --- |
| Name | T22 - Create style sheet for application |
| Description | Create a style sheet for the application to use |
| Work Effort | 1 hour |

|  |  |
| --- | --- |
| Name | T23 - Create skeletons for backend classes |
| Description | Create skeleton classes for PortScanner, MonitorThread, SettingsManager, DatabaseUtils, and NetworkUtils. |
| Work Effort | 1 hour |

|  |  |
| --- | --- |
| Name | T24 - Create SettingsManager class |
| Description | Create a new class SettingsManager and XML file functions, for managing settings XML file. |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T25 - Create NetworkNode class |
| Description | Create class for storing information related to computers |
| Work Effort | 1.5 hours |

|  |  |
| --- | --- |
| Name | T26 - Create Port class |
| Description | Create class for storing information related to ports |
| Work Effort | 1.5 hours |

|  |  |
| --- | --- |
| Name | T27 - Create CommandThread class |
| Description | Create threading class for launching commands |
| Work Effort | 1.5 hours |

|  |  |
| --- | --- |
| Name | T28 - Create ScanThread class |
| Description | Create threading class for starting scanner threads |
| Work Effort | 1.5 hours |

|  |  |
| --- | --- |
| Name | T29 - Create bash script for initializing files |
| Description | Create initsettings bash script that will check if certain files exist or not, and if not, create them |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T30 - Wire up SYN scan to Scan Options in GUI |
| Description | Wire up the SYN scan to radio buttons and the Start Scan button on the Scanner tab of the application, format scanner output. |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T31 - MonitorThread: getters and setters (Computers, ports) |
| Description | Create MonitorThread class, and getter and setter functions. |
| Work Effort | 1 hour |

|  |  |
| --- | --- |
| Name | T32 - MonitorThread: start() and stop() |
| Description | Create start() and stop() functions for active monitoring of set networks. |
| Work Effort | 4 hours |

|  |  |
| --- | --- |
| Name | T33 - Wire up MonitorThread to GUI |
| Description | Wire up the monitor backend to the GUI buttons, format monitor output. |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T34 - Use Connector J and create DatabaseUtils class to manage database queries from application |
| Description | Set up Connector J driver and create queries in DatabaseUtils class |
| Work Effort | 4 hours |

|  |  |
| --- | --- |
| Name | T35 - Create email dialog |
| Description | Create email dialog for editing emailgroup (list of emails that get sent reports) |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T36 - Create functions for reports |
| Description | Create code for saving and sending reports to emailgroup |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T37 - Create edit scan dialog |
| Description | Create edit scan dialog for editing currently set IP address and ports to be scanned |
| Work Effort | 2 hours |

|  |  |
| --- | --- |
| Name | T38 - Create project poster |
| Description | Copy relevant information from final report into project poster, write other sections, provide screenshots |
| Work Effort | 3 hours |

**6.2 Project Schedule**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **(1/22 - 2/4)** | **(2/5 - 2/18)** | **(2/19 - 3/4)** | **(3/5 - 3/18)** |
| Matthew | T1, T2, T3 | T4, T5 | T6, T7 | T8, T9 |
| William | T16, T17, T18, T19 | T20, T21, T22, T23 | T24, T25, T26 | T27, T28, T29 |

|  |  |  |
| --- | --- | --- |
|  | **(3/19 - 4/1)** | **(4/2 - 4/29)** |
| Matthew | T10, T11, T12 | T13, T14, T15 |
| William | T30, T31, T32, T33 | T34, T35, T36, T37, T38 |

**6.3 Risk Plan**

|  |  |
| --- | --- |
| Name | R1 – Change database software |
| Description | If the current database is unable to support certain functions, replacement may be necessary. |
| Solution | 1. Research new database software to use  2. Re-add and reschedule completed tasks associated with the database  3. Workload for the newly added tasks will be split evenly |

|  |  |
| --- | --- |
| Name | R2 – Change GUI software |
| Description | If the current GUI is unable to support certain functions, replacement may be necessary. |
| Solution | 1. Research new GUI software to use  2. Re-add and reschedule completed tasks associated with the GUI  3. Workload for the newly added tasks will be split evenly |

|  |  |
| --- | --- |
| Name | R3 – Family/Medical Emergencies |
| Description | A team member is unable to complete their assigned tasks due to a family or medical emergency |
| Solution | 1. Reschedule uncompleted tasks  2. Workload for rescheduled tasks will be split evenly |

**6.4 Estimated Financial Budget**

All libraries and programs to be used are either open-sourced or available free through the university making the project cost free.

**6.5 Teamwork Plan**

The version control used is Github.com. For communication we used discord and text messaging. For files unrelated to programming we used google drive to store them and share between each other. We created a .txt file that listed various tasks currently ongoing in the drive as well to keep track as it suited our needs. With weekly meetings and frequent communication we were always on track and needed no other 3rd party to help keep track.

**7.0 Testing**

**7.1 Test Plan**

The test plan for the project will use the following procedure:

1. Complete task for current sprint
2. Determine which test cases are related to completed task by comparing use case requirements between test case and task
3. For each test case:
   1. Ensure preconditions are met
   2. Follow basic course of events
   3. Determine whether results are expected or not:
      1. If results were unexpected, review code and steps taken to implement task, debug, rerun test plan procedure
      2. If results were expected, the test case has been passed
   4. If the test case has been passed, check next test case if necessary
4. If all test cases have been passed, mark the task as complete and move onto next task in sprint

This test plan will ensure that no task gets marked as completed unless its implementation passes all related test cases (a common use case or requirement between the two). No task will move beyond a sprint unless it has been run against these test cases successfully.

**7.2 Test Cases**

|  |  |
| --- | --- |
| **Name** | TEST-1 Verify that SYN scan works |
| **Requirement** | US-1: Scan Network |
| **Preconditions** | That the scan has been given an ip address and ports to scan |
| **Basic course of events** | 1. User clicks Edit Scan 2. User enters ip address & ports to scan 3. Scan Output displays results from scan |
| **Expected results** | A display of status from each port specified for the computer, status being open, filtered, or closed, port being an integer |
| **Acceptance criteria** | That each port specified for an ip address has a line returned stating its current status |

|  |  |
| --- | --- |
| **Name** | TEST-2 Verify that Monitor works |
| **Requirement** | US-8: Monitoring Tab |
| **Preconditions** | The database has information for computers and ports from those computers to use in the monitor process |
| **Basic course of events** | 1. User clicks Start Monitor 2. Monitor runs, displaying output in Monitor Output 3. User clicks Stop Monitor 4. Monitor stops displaying output in Monitor Output |
| **Expected results** | Monitor Output will display IP address followed by a list of ports that do not have status states that were expected (if database entry for expected status of a port is “open” and output from monitor scan returns “closed”, this will be displayed as an alert in Monitor output. |
| **Acceptance criteria** | That all ports with status from monitor scan not equal to their expected status from database will be displayed in Monitor Output with Alert tag |

|  |  |
| --- | --- |
| **Name** | TEST-3 Verify that initsettings.sh works |
| **Requirement** | None |
| **Preconditions** | The XML file is not present |
| **Basic course of events** | 1. User starts application 2. The application runs initsettings.sh and creates default XML settings file |
| **Expected results** | The settings.xml file is created and present in settings directory |
| **Acceptance criteria** | The settings.xml file has xml text content, and is usable with Test Case 4 |

|  |  |
| --- | --- |
| **Name** | TEST-4 Verify that XML settings file works |
| **Requirement** | US-3: Update Database with XML |
| **Preconditions** | The XML file is present and has valid data for database upload |
| **Basic course of events** | 1. User clicks Database dropdown 2. User clicks Update (XML) 3. Database updates with data taken from XML settings file |
| **Expected results** | The MySQL database is updated with data present in the XML settings file |
| **Acceptance criteria** | MySQL database has computers table filled with computers specified in XML settings file, has ports table filled with ports (port number, port expected status) from XML settings file |

|  |  |
| --- | --- |
| **Name** | TEST-5 Verify that database restore works |
| **Requirement** | US-10: Restore Database |
| **Preconditions** | The database has information for computers and ports from those computers available |
| **Basic course of events** | 1. User clicks Database dropdown 2. User clicks Restore Database 3. Database drops portscan table 4. User opens and closes application, no computers are displayed under Monitored Computers, or, user runs monitor application by clicking Start Monitor, and monitor returns there are no computers to monitor |
| **Expected results** | The portscan table is no longer available in database |
| **Acceptance criteria** | The monitor returns there are no computers to monitor, there are no computers displayed under Monitored Computers |

**7.3 Test Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint** | **Tasks** | **Test Cases** | **Person Responsible** |
| Sprint 1 (1/22 - 2/4) | T1, T2, T3  T16, T17, T18, T19 | None  None | Matthew  William |
| Sprint 2 (2/5 - 2/18) | T4, T5  T20, T21, T22, T23 | None  None | Matthew  William |
| Sprint 3 (2/19 - 3/4) | T6, T7  T24, T25, T26 | None  None | Matthew  William |
| Sprint 4 (3/5 - 3/18) | T8, T9  T27, T28, T29 | None  TEST-3 | Matthew  William |
| Sprint 5 (3/19 - 4/1) | T10, T11, T12  T30, T31, T32, T33 | TEST-1  TEST-1, TEST-2 | Matthew  William |
| Sprint 6 (4/2 - 4/29) | T13, T14, T15  T34, T35, T36, T37, T38 | None  TEST-4, TEST-5 | Matthew  William |

**8.0 Project Implementation & Results**

**8.1 Implementation Strategy**

Communication for this project was done through Discord, text messages, and weekly meetings. We did not use a program to track work instead we used text documents to list what we needed to get done and checked it off in the shared file inside of google drive. For version control we used Github. Our work was split by frontend/java and backend/C. William worked on the GUI and did work in java while Matthew worked on the C code and the database. The testing was split up by code with Matthew testing the C code and William testing the java code.

**8.2 Sprints**

Sprint Overview:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **Tasks** | **Person Responsible** | **Hours Projected** | **Hours Actual** |
| Sprint 1 (1/22 - 2/4) | T1, T2, T3  T16, T17, T18, T19 | Matthew  William | 8.5  5 | 9  5 |
| Sprint 2 (2/5 - 2/18) | T4, T5  T20, T21, T22, T23 | Matthew  William | 4  6.5 | 6  7 |
| Sprint 3 (2/19 - 3/4) | T6, T7  T24, T25, T26 | Matthew  William | 5  5 | 5  5 |
| Sprint 4 (3/5 - 3/18) | T8, T9  T27, T28, T29 | Matthew  William | 6  5 | 4  5 |
| Sprint 5 (3/19 - 4/1) | T10, T11, T12  T30, T31, T32, T33 | Matthew  William | 4.5  9 | 5  7 |
| Sprint 6 (4/2 - 4/29) | T13, T14, T15  T34, T35, T36, T37, T38 | Matthew  William | 15  13 | 15  13 |

Matthew 43 hours projected 44 actual

William 43.5 hours projected 43 actual

Individual Sprints:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sprint** | **Task** | **Person Responsible** | **Hours Projected** | **Hours Actual** |
| Sprint 1 (1/22 - 2/4) | T1 | Matthew | 4 | 4 |
| Sprint 1 (1/22 - 2/4) | T2 | Matthew | 1.5 | 2 |
| Sprint 1 (1/22 - 2/4) | T3 | Matthew | 3 | 3 |
| Sprint 1 (1/22 - 2/4) | T16 | William | 2 | 2 |
| Sprint 1 (1/22 - 2/4) | T17 | William | 1 | 1 |
| Sprint 1 (1/22 - 2/4) | T18 | William | 1 | 1 |
| Sprint 1 (1/22 - 2/4) | T19 | William | 1 | 1 |
| Sprint 2 (2/5 - 2/18) | T4 | Matthew | 2 | 3 |
| Sprint 2 (2/5 - 2/18) | T5 | Matthew | 2 | 3 |
| Sprint 2 (2/5 - 2/18) | T20 | William | 4 | 4 |
| Sprint 2 (2/5 - 2/18) | T21 | William | 0.5 | 1 |
| Sprint 2 (2/5 - 2/18) | T22 | William | 1 | 1 |
| Sprint 2 (2/5 - 2/18) | T23 | William | 1 | 1 |
| Sprint 3 (2/19 - 3/4) | T6 | Matthew | 2 | 2 |
| Sprint 3 (2/19 - 3/4) | T7 | Matthew | 3 | 3 |
| Sprint 3 (2/19 - 3/4) | T24 | William | 2 | 2 |
| Sprint 3 (2/19 - 3/4) | T25 | William | 1.5 | 1.5 |
| Sprint 3 (2/19 - 3/4) | T26 | William | 1.5 | 1.5 |
| Sprint 4 (3/5 - 3/18) | T8 | Matthew | 2 | 2 |
| Sprint 4 (3/5 - 3/18) | T9 | Matthew | 4 | 2 |
| Sprint 4 (3/5 - 3/18) | T27 | William | 1.5 | 1.5 |
| Sprint 4 (3/5 - 3/18) | T28 | William | 1.5 | 1.5 |
| Sprint 4 (3/5 - 3/18) | T29 | William | 2 | 2 |
| Sprint 5 (3/19 - 4/1) | T10 | Matthew | 2 | 2 |
| Sprint 5 (3/19 - 4/1) | T11 | Matthew | 2 | 2 |
| Sprint 5 (3/19 - 4/1) | T12 | Matthew | 0.5 | 1 |
| Sprint 5 (3/19 - 4/1) | T30 | William | 2 | 2 |
| Sprint 5 (3/19 - 4/1) | T31 | William | 1 | 1 |
| Sprint 5 (3/19 - 4/1) | T32 | William | 4 | 2 |
| Sprint 5 (3/19 - 4/1) | T33 | William | 2 | 2 |
| Sprint 6 (4/2 - 4/29) | T13 | Matthew | 3 |  |
| Sprint 6 (4/2 - 4/29) | T14 | Matthew | 8 |  |
| Sprint 6 (4/2 - 4/29) | T15 | Matthew | 4 |  |
| Sprint 6 (4/2 - 4/29) | T34 | William | 4 |  |
| Sprint 6 (4/2 - 4/29) | T35 | William | 2 |  |
| Sprint 6 (4/2 - 4/29) | T36 | William | 2 |  |
| Sprint 6 (4/2 - 4/29) | T37 | William | 2 |  |
| Sprint 6 (4/2 - 4/29) | T38 | William | 3 |  |

**8.3 Results**

Sprint 1 (1/22 - 2/4)

For this sprint Matthew implemented tasks T1, T2, and T3.

T1 involved making revisions to last semester's final report, T2 & T3 involved setting up Connector C library to run database operations in C.

The original report was revised, although setup of the Connector C library was extended into the Sprint 2 by a bit.

William implemented tasks T16-T19.

T16 & T17 involved doing setup - creating a virtual machine network and setting up a GitHub repository to use as version control. Both were setup without any issues.

T18 & T19 involved creating the GUI - designating needed classes and creating skeletons for these classes.These tasks were implemented without issue.

Sprint 2 (2/5 - 2/18)

For this sprint Matthew implemented tasks T4 and T5.

These involved creating a database schema for the project - storing computers, their associated networks, ports to be monitored, and the status and expected status of these ports.

These tasks were implemented with issue, but later needed some revisions.

William implemented tasks T20 - T23.

These tasks involved creating code for various elements of the GUI application (buttons, labels, text boxes), positioning them, creating icons for menu functions, creating a style sheet and creating more skeletons for needed backend classes.

These tasks were completed successfully, but menu icons and style sheet were later revised, as well as new backend classes being created.

Sprint 3 (2/19 - 3/4)

For this sprint Matthew implemented tasks T6 and T7.

At this point in development queries were created for the C programs to use, and setup the scan program using libnet and libpcap libraries.

Both tasks were completed, but the queries later needed to be fixed and refactored into JDBC queries.

William implemented tasks T24 - T26.

These tasks involved coding SettingsManager, which is used to parse the XML file containing computers and ports, and the NetworkNode and Port classes, which are used to store information related to computers and ports:

|  |  |
| --- | --- |
| public class NetworkNode { |  |
|  | private ArrayList<Port> ports = new ArrayList<Port>(); |
|  | private String address = null; |
|  | private String network = null; |

|  |  |
| --- | --- |
| public class Port { |  |
|  | private int number = 0; |
|  | private String status = null; |
|  | private String expected\_status = null; |

These tasks were completed, but the XML needed to be debugged in Sprint 4.

Sprint 4 (3/5 - 3/18)

For this sprint Matthew implemented tasks T8 & T9.

These tasks involved setting up libnet and libpcap libraries on our test system, and modifying the code we used for our SYN scan.

The tasks were completed without error. The following is sample code from the SYN scan:

|  |  |
| --- | --- |
|  | /\* write the packet \*/ |
|  | if ((libnet\_write (l)) == -1) |
|  | { |
|  | fprintf (stderr, "Unable to send packet: %s\n", |
|  | libnet\_geterror (l)); |
|  | exit (1); |
|  | } |
|  |  |
|  | /\* set variables for flag/counter \*/ |
|  | answer = 1; |
|  | tv = time (NULL); |
|  |  |
|  | /\* capture the reply \*/ |
|  | while (answer) |
|  | { |
|  | pcap\_dispatch (handle, -1, packet\_handler, NULL); |
|  |  |
|  | if ((time (NULL) - tv) > 2) |
|  | { |
|  | answer = 0; /\* timed out \*/ |
|  | printf ("Port %d appears to be filtered\n", ports[i]); |
|  | } |
|  | } |

William implemented tasks T27-T29.

These tasks involved creating thread classes for running commands and scans, as well as creating a bash script that initializes needed files if they are not present.

These tasks were completed without error.

Sprint 5 (3/19 - 4/1)

For this sprint Matthew implemented tasks T10-T12.

This involved revising the SYN scan to take in multiple ports to scan (whereas before ports were hardcoded) and creating the skeleton for a C monitor and README file for the project.

The monitor remains unfinished, but tasks T10 and T12 were completed.

William implemented tasks T30-T33.

This included wiring up the SYN scan created in the last sprint to the GUI buttons, and tasks T31-T33 involved creating the monitor in Java as class MonitorThread. These tasks were added during this sprint and other tasks were pushed back since the decision was made to do the monitor in Java instead of C due to difficulties implementing the monitor in C.

The following shows the core functionality of the monitor:

|  |  |
| --- | --- |
|  | //Compare the output from the scan to database |
|  | for(int m = 0; m < ports.size(); m++){ |
|  |  |
|  | //If the output status != expected status for this port, |
|  | //then send an alert to the user |
|  | if(!outputPorts.get(m).getStatus().equals(ports.get(m).getExpectedStatus())){ |
|  | if(alert == false){ |
|  | monitorOutputText.appendText("Alert!\t\t" + "Computer " + ip + ":\n"); |
|  | alert = true; |
|  | } |
|  |  |
|  | //Set spacing for some aesthetically pleasing console output |
|  | if(("Port " + outputPorts.get(m).getNumber() + |
|  | " is " + outputPorts.get(m).getStatus()).length() >= 20){ |
|  | spacing = "\t\t"; |
|  | }else{ |
|  | spacing = "\t\t\t"; |
|  | } |
|  |  |
|  | monitorOutputText.appendText("\t\t\tPort " + outputPorts.get(m).getNumber() + |
|  | " is " + outputPorts.get(m).getStatus() + spacing + |
|  | "(expected: " + ports.get(m).getExpectedStatus() + ")\n"); |
|  |  |
|  | System.out.println("Monitor\t\t\tPort " + outputPorts.get(m).getNumber() + |
|  | " is " + outputPorts.get(m).getStatus() + spacing + |
|  | "(expected: " + ports.get(m).getExpectedStatus() + ")"); |
|  | } |
|  |  |
|  | //If the output status != status in database, |
|  | //then update the database entry for this port |
|  | if(!outputPorts.get(m).getStatus().equals(ports.get(m).getStatus())){ |
|  | dbUtils.updatePortStatus(ip, outputPorts.get(m).getNumber(), outputPorts.get(m).getStatus()); |
|  | } |

All tasks were completed, but debugging the monitor was extended into Sprint 6.

Sprint 6 (4/2 - 4/29)

For this sprint Matthew implemented tasks T13-T15.

These involve modifying our SYN scan code adding multithreading to it, as well as setting up a hacking demo and final presentation for the presentation of the project.

William implemented tasks T34-T38.

These tasks involve some functions that were pushed back, including email alerts, creating reports, and a dialog for creating scans. T34 involves setting up JDBC and a DatabaseUtils class for database functions, and T38 is creation of the project poster.

**8.4 Testing Results**

TEST-1 Verify that SYN scan works

This test case involved ensuring that the SYN scan operates properly. It was tested in Sprint 5 by both Matthew and William, Matthew testing the SYN scan outside the project (and updated SYN scan that takes port ranges) and William testing the SYN scan from GUI calls.

The SYN scan was verified to work properly from command line as it returns each port with status for that port. There were some issues with testing it from the GUI application, and the delimiter used for the port string had to be changed from a semicolon to a colon, due to bash interpreting each port as a command (formerly “21;22;23;” , changed to “21:22:23:”).

This change of the delimiter ensured that the scan would run properly when called from the GUI application. The only reason it worked on command line was because if supplied in quotations bash would not take each port as its own command.

TEST-2 Verify that Monitor works

This test case involved ensuring that the monitor returns relevant and correct information compared to what was stored in the database and the output from the SYN scan. Test 2 was conducted in Sprint 5 by William.

The monitor initially returned faulty information compared to information being returned from the database and SYN scan. It was debugged and fixed - some arraylists were not being flushed properly and the monitor was reusing information from other computers.

After these changes the monitor was verified to work, as it returned the expected alerts.

TEST-3 Verify that initsettings.sh works

This test case involved ensuring that the initialization script worked (setting up needed files and ensuring the needed files are present). Test 3 was conducted in Sprint 4 by William.

The test was run and results were as expected. The initsettings script successfully creates the default XML settings file needed for the application to operate properly.

TEST-4 Verify that XML settings file works

This test case involved verifying that the XML settings file works properly (able to update database with XML elements without issue). Test 4 was conducted by William in Sprint 6.

SettingsManager runs the function that takes the XML elements and inserts them into the database. Initially the test failed as the function was only taking the first computer and updating the database with it. This was due to an issue with the ArrayList containing the computers, which was fixed.

After the fix the test case ran as intended and XML update was verified to work.

TEST-5 Verify that database restore works

This test case involved verifying that the database restore query works. Test 5 was conducted by William in Sprint 6.

The query is run from the DatabaseUtils class. The test ran without fail and the database restore function was verified to run correctly after checking the portscan database in mysql (created empty tables as intended).

**9. Problems & Changes**

**9.1 Design Changes**

There were various small changes during implementation and one major change.

The major change that occured was to not include a vulnerability database. The reason this change occurred was time constraints. The main priority was to get the actual scan up and running and building the scan as well as installing dependencies took longer than expected. The time deviation resulted in the need to remove the vulnerability database. One point to consider is this program is not a vulnerability scanner, it is a port scanner. The program functions as it was intended but only lacking supportive details to the unaware.

The various small changes, two to be exact, took place on the GUI side of the program. The original plan saw a settings option to the top left in the GUI. This was discarded and instead options were added inside each tab. The other was to add a restore database feature to the top left in the case of scanning a new network to avoid a cluttered database.

**9.2 Sprint Changes**

Due to the design changes, a few changes in the sprints had to be made. In total there were 8 tasks that were removed due to the objective no longer being needed or a change in nature.

|  |  |
| --- | --- |
| Name | T6 – Test ACK scan |
| Description | Test ACK scan code on a single IP address. |
| Work Effort | 2 hours |

Time constraints caused the ACK scan to go unfinished

|  |  |
| --- | --- |
| Name | T10 – PortScannerApplication: scanNetwork() |
| Description | Wire up scanNetwork() to scanning code, so that it starts a specified scan (SYN or ACK). |
| Work Effort | 2 hours |

ACK scan was never finished and there was an option added to the ScannerPane to control the scans.

|  |  |
| --- | --- |
| Name | T11 – PortScannerApplication: verifyAdministrator() |
| Description | Write a function that will verify that the currently logged-in user is an administrator. |
| Work Effort | 2 hours |

No login feature was needed for this project so there was no need to verify the admin.

|  |  |
| --- | --- |
| Name | T12 – NetworkUtils: findActivePorts() |
| Description | Write a function that will retrieve active, open ports for a specified IP. |
| Work Effort | 3 hours |

All database files were bundled in one new task, T6.

|  |  |
| --- | --- |
| Name | T13 – NetworkUtils: findAvailableNetworks() |
| Description | Write a function that will retrieve networks the host computer can communicate with. |
| Work Effort | 4 hours |

All database files were bundled in one new task, T6.

|  |  |
| --- | --- |
| Name | T15 – SettingsManager: updateSettings() |
| Description | Write a function that, given a Settings object, will overwrite the current saved settings with the new settings. |
| Work Effort | 3 hours |

The settings option was removed.

|  |  |
| --- | --- |
| Name | T17 – Test Scan Options in GUI |
| Description | Ensure that the scans actually run when the Start Scan button is pressed. |
| Work Effort | 2 hours |

Replaced by T30.

|  |  |
| --- | --- |
| Name | T21– Test NetworkMonitor in GUI |
| Description | Ensure that the network monitor actually runs when GUI buttons are used. |
| Work Effort | 2 hours |

Replaced by task T33.

**9.3 Final Results**

The final state of each component/feature is as follows:

**Green** = finished

**Yellow** = incomplete

**Orange** = not started

|  |  |  |  |
| --- | --- | --- | --- |
| Component Name | State | Related Sections | Details |
| Individual Scan (SYN Scan) | **Finished** | US-1 |  |
| Edit Scan Options | **Finished** | US-2 |  |
| Port Monitor | **Incomplete** | US-3, US-8 | Does not save historic port data, does not scan all ports for each computer |
| Help Document | **Not started** | US-4 | Has not been implemented |
| Edit Email Group | **Finished** | US-5 |  |
| Send Report by Email | **Incomplete** | US-6 | Not in a working state |
| Save Report | **Incomplete** | US-7 | Currently no option to name reports |
| Settings/Menu Customization | **Not started** | US-9, US-10 | Has not been implemented |
| GUI Database Options | **Finished** | US-11 | Restore Database, Update Database from XML, GUI displays stored computers |
| Database | **Finished** |  |  |

**10. Risks**

The risk plan saw three potential problems: A need to change the database, a need to change the GUI, or an unfortunate family or medical emergency. Thankfully none of these risks occurred during the course of this project.

As for unforeseen risks, no unexpected events took place nor a need to change any core library or dependency due to issues.

**11. Financial budget**

The estimated budget for this project was it could be completed cost free using open-source libraries and free programs. Through the course of the project there was no need to buy anything additional to complete this project. The financial budget was cost free.

**A1. Project Societal Concerns**

**Economic concerns:** This project uses open-source libraries available to all and builds a unique port scanner. Due to the resources used this project will stay open-sourced to customers.

**Ethical concerns:** A port scanner looks for any open ports on a network. This one is no different and open ports on a network can lead to vulnerabilities. This scanner by itself does not have the capabilities to breach a firewall and has only been tested in an internal network, where it was designed to be used. The purpose of this scanner is purely educational and is not intended to be used for any malicious purposes.

**A2. References**

[1] “Nmap Security Scanner” 1996-2018 Insecure.Com, LLC

<https://nmap.org/>

[2] Charles M. Kozierok “The TCP / IP Guide” No Starch Press, Inc 2005.

[3] Libnet 1998-2004 Mike D. Schiffman; 2009-2013 Sam Roberts

<https://github.com/sam-github/libnet>

[4] Libpcap 2010-2017 Luis MartinGarcia

<https://www.tcpdump.org/>

[5] Network Security Tools, (Clarke, Dhanjani)

<http://books.gigatux.nl/mirror/networksecuritytools/0596007949/toc.html>