**CS Senior Design Specification Report**

**APAD: ARP POISONING ATTACK DETECTOR**

**NICO LARA, JOHN BILELLA, NICK MASCI**

**ADVISOR: DR. FU**

**CSC 198S**

**Fall 2023**

**12/17/2023**

**Abstract**

Hacking techniques continue to become more prevalent in daily life and can affect virtually everybody. Some techniques rely on sophisticated methods of back-engineering code to allow for infiltration or perhaps can take advantage of basic networking concepts. ARP poisoning attacks are one of the most common cyberattacks and first appeared in the 1990’s. These attacks are relatively easy to conduct with modern tools, however opportunities to employ one of these attacks are limited as they require a device to have access to a network. “Ettercap” is one of these tools, and is the tool that we have used in our experiments while developing our project. To confront the security concern posed by ARP poisoning attacks, our goal is to develop a method of catching an attack that can easily be used by a network administrator and is reliable. To do this, we utilize a C program operating on the backend that reads the IP/MAC data from the ARP cache from the network administrators computer. The unique IP/MAC pairings are added to an array of devices within the program. If a change is made to any of these pairings, the program will alert on an attack and then block the attacking device from your network. The results of this project were successful, we were able to detect an ARP poisoning attack very easily and prevent the attacker from stealing any network data. Additionally, while developing this project, our team learned a lot about the topic and developed our project management skills.

**Chapter 1: Project Description and Goals**

**1.1 - Project Scope and Vision**

The age of the internet has brought about vast changes in the way that people communicate, behave, and interact with society. In the days of paper-and-pen communications, before the modern age, a message had to be physically passed from one person to another, until it had reached its intended reader. For this message to be intercepted, it had to be physically taken from a person in custody of the message. Today, data is passed around from a user’s device, to a series of routers and modems, satellites and communication towers [12]. At every step of a message’s transmission, its security must be ensured, just as if it were being personally carried by a courier in the old days. Ensuring this security proves to be extremely complicated, as threats are constantly emerging and new methods of intercepting communications between a sender and an intended receiver are being discovered. In the world of cybersecurity – we call this a Man-In-The-Middle (MITM) attack.

The first recorded occurrence of a MITM attack did not, however, happen in the age of the internet, but in 1586, and was known as the Babington Plot [13]. This plot, to assassinate Queen Elizabeth I, was foiled by intercepting the messages between its conspirators and making changes to the message contents in order to reveal the identities of the plotters. In modern times, this kind of attack is carried out in a different manner, but the idea is the same. In 2019, a well known attack occurred when a group of hackers stole $1 Million from an Israeli startup company[14]. This MITM attack became known as the ‘Ultimate’ MITM attack. The hackers were able to intercept the communications between the Israeli startup and the Chinese venture-capital firm they were using, and change the $1 Million payment recipient from the Israelis’ account, to an account that the attackers could access.

A MITM attack is a rather broad term, referring to many different methods of intercepting communications. One of these methods is the ARP poisoning attack, which involves exploiting the ways by which computers know which other computers they are communicating with. A host reads the ARP cache, which is a mapping between MAC and IP addresses, and uses it to connect to destinations on the network [12]. A device’s IP address is the number assigned to each device on a network in order to indicate which traffic goes to what device. This may change periodically, depending on the type of network connection the router has. The MAC address of a device, however, is the unique number assigned to the hardware of a device. Under normal circumstances, this should never change. An ARP poisoning attack occurs when an attacker infiltrates a network, and changes their IP or MAC address to match that of a legitimate user on the network, thus receiving data intended for their victim.

There are several methods for preventing such attacks on your network, including: controlling your network’s security, using network switch security, and utilizing static ARP tables. To employ an ARP poisoning attack, an attacker must have access to your network, so properly controlling network access is the best way to mitigate this risk. Network switch security, also known as DAI (Dynamic ARP Inspection), is a feature utilized by most managed ethernet switches that evaluates each ARP message and drops packets that appear suspicious. Static ARP tables statically map each MAC address to their proper IP address, which is effective for preventing an ARP poisoning attack, but this also adds a large administrative burden [11].

If a motivated attacker bypasses some of these defenses, or your network does not utilize switches featuring DAI, our project will be there to catch the attacker red-handed, and will prompt the network admin to remove that device from your network [11]. Our project does this by analyzing ARP packets being sent on your network and compares pairings of MAC/IP addresses. If an IP address that is already claimed by another MAC address is detected in a new pairing, this will cause an “ARP Poisoning Attack Alert”.

**1.2 - Effect on Society**

This program helps ensure the security of network operations which is a vital aspect of all networks, whether it is a school network, corporate, or a private network. This tool can be run by an administrator in the background, utilizing minimum resources as they continue about their business, and can reliably catch an ARP poisoning attack. By preventing this attack, the network remains safe and no data is lost or stolen, which is good for the institution employing this tool.

**1.3 - Design Approach**

APAD’s backend was designed using the C libraries libpcap and netinet, and Java Swing and the Apache Netbeans IDE were used for the User Interface. Libpcap is a well-known, widely available C library that allows for our application to interface with the ARP table and analyze incoming packets. The netinet library made it significantly easier to process these packets, as it contains information and data structures relating to various communication protocols.

Java Swing and the Netbeans IDE were crucial in quickly developing a coherent User Interface. Netbeans’ drag-and-drop functionality allowed us to skip a lot of the repetitive, time consuming work that goes into creating a UI.

We chose to have each part of the application be its own separate program - log.c, monitor.c, and GUI.java - so that the backend can run at a higher frequency, while the UI only has to update once per second. This increases the program’s efficiency, as the GUI is more resource intensive than the backend programs, which do not need to display any graphics.

**1.4 - Results**

We aimed to create a program that can detect, block, and notify the user of ARP Poisoning Attacks. This goal was successfully attained: log.c analyzes the ARP table and extracts IP/MAC pairings, monitor.c analyzes changes to these pairings, and the GUI displays this information in an easy-to-read format. In addition, user friendliness was increased further by adding a “color blind mode” setting to the User Interface, changing the color palette of the device nodes to be easier to see by those who are color blind. Other settings include enabling debug features, and the ability to select the directory where monitor.c and log.c are running from. The ability to choose a directory, as opposed to having one hard-coded into the application, significantly increases the program’s versatility and usability on multiple devices.

**Chapter 2: Project Goals**

* Develop a program that will detect an ARP Poisoning Attack as soon as possible
* Develop an easy-to-use User Interface for maximum user friendliness
* Create and maintain a running log for historical reference

**Chapter 3: Requirements Specification**

Chapter 3 will serve as a brief overview of the system components, to be described in detail in the succeeding chapters, and will detail the users of the system and the roles each one plays in the system’s functionality. Section 3.1, the system perspective, will illustrate these interactions in a diagram, and will describe who each user is. Section 3.2, the User Stories, will describe every way the user interacts with the system in detail. Section 3.3 will then describe in detail the inputs and outputs of each part of the system. Lastly, sections 3.4 and 3.5 will detail the functional and non-functional requirements of the system, respectively. The functional requirements are features that the user is able to interact with, while the non-functional requirements occur behind the scenes, likely unnoticed by the user, but important nonetheless.

**3.1 - System Perspective**

The system perspective diagram, shown below, is a basic overview of how all parts of the system interact with each other. Each user plays a different role in the system’s functionality.

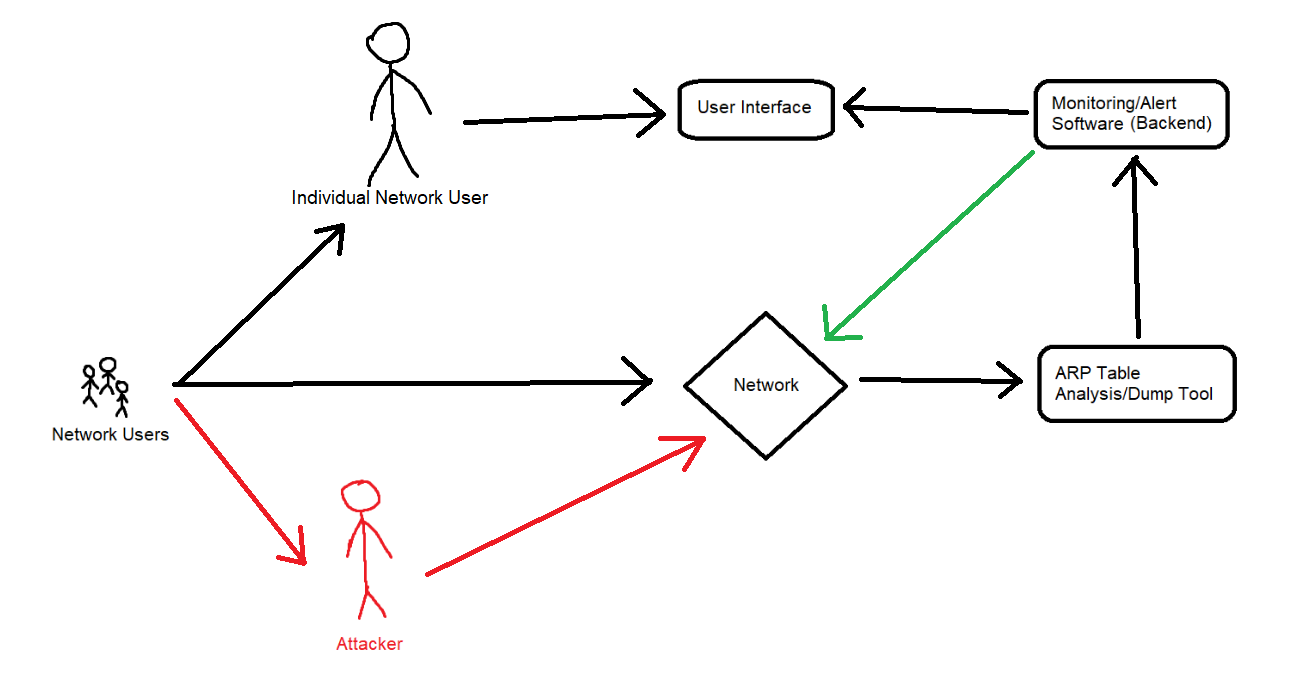
****

Figure 3.1.1: System Diagram

**Users**

There are two types of users that interact with this system in some way. The Network Users are simply any device connected to the network. Each of these users are an Individual Network User and, possibly, an Attacker. The application is to be installed on each Network User’s device; as attacks are blocked by dropping packets directly from each device. The Attacker is any Network User that is trying to launch an ARP poisoning attack on the network. When this happens, the system will send an alert to the Individual Network Users, and they will be able to drop all packets sent to and from the Attacker..

**User Interactions**

As stated previously, this software is to be installed on the computer of each Network User. The software will constantly monitor the network and alert the User if an ARP poisoning attack is detected. All devices connected to the network - the Network Users - will be present on the ARP Table, and their IP and MAC addresses will be tracked by the Analysis/Dump Tool. If one of these Network Users changes their IP address to one that already exists in the ARP Table, they will be identified as an Attacker. This will cause an alert to be sent to the Users, who will then have packets sent to and from the attacker be automatically dropped. The User Interface will be largely click-based, with buttons for the user to navigate between different screens, and checkboxes for changing settings. This simple, graphical, click-based interface allows for a wider range of potential users of the software, as those unfamiliar with networking or computers in general will be able to navigate with ease.

**3.2 - User Stories**

In this section, each user interaction with the system will be described in detail. For a given user story, the name, description, reason, user type, steps, priority, story points, and references will be detailed. Priority and story points range from 1 to 5, with 1 being the lowest priority/easiest to program, and 5 being highest priority/most difficult to program, respectively.

| **Name** | US-1: Set Target directory |
| --- | --- |
| **Description** | User must provide GUI with the directory containing monitor.c and log.c |
| **Rationale** | The GUI uses monitorOUT.txt and monitorHistory.txt, which are created in the same folder as monitor.c when run. Monitor.c also depends on log.c being in the same folder, as monitor.c uses log.txt |
| **Users** | Network Users |
| **Step-by-Step Description** | 1. User types in proper directory and either presses ENTER/RETURN or clicks the “Save” button 2. GUI attempts to open monitorOUT.txt and monitorHistory.txt from this location and display their information |
| **Priority** | 1 |
| **Story Points** | 2 |
| **Cross References** | SC-1, UI-2 |

Table 3.2.1: US-1

| **Name** | US-2: Monitor Network |
| --- | --- |
| **Description** | Constantly analyzes IP/MAC addresses on ARP Table |
| **Rationale** | To find suspicious changes made to the ARP Table. Nodes will be green on the GUI if no threats are detected. If a threat is detected, that device’s respective node will turn red |
| **Users** | Network Users |
| **Step-by-Step Description** | 1. System continuously compares previous IP/MAC address pairings from SC-4 2. If a suspicious change has been made, add iptables rule to block packets to and from attacker |
| **Priority** | 5 |
| **Story Points** | 8 |
| **Cross References** | SC-1, SC-2, SC-3, SC-4, SC-5, UI-1 |

Table 3.2.2: US-2

| **Name** | US-4: Change Settings |
| --- | --- |
| **Description** | User can view and change various settings |
| **Rationale** | Allows user to customize their experience so it better suits their needs |
| **Users** | Network Users |
| **Step-by-Step Description** | 1. User clicks “Settings” button 2. System displays UI-2 3. User toggles a setting 4. System saves this change and acts accordingly |
| **Priority** | 1 |
| **Story Points** | 3 |
| **Cross References** | SC-1, UI-2 |

Table 3.2.4: US-4

| **Name** | US-5: View History |
| --- | --- |
| **Description** | User views all past events on network |
| **Rationale** | Allows user to refer back to previous network events in case further action must be taken |
| **Users** | Network Users |
| **Step-by-Step Description** | 1. User clicks “History” button 2. System displays UI-3 3. User is provided constantly-updating list of events and can look back at all past events |
| **Priority** | 3 |
| **Story Points** | 3 |
| **Cross References** | SC-1, SC-2, SC-4, UI-3 |

Table 3.2.5: US-5

**3.3 - External Interfaces:**

The external interfaces, shown below, are the system’s input/output methods, as well as the UIs that are output by each part of the system. Table 3.3.1 shows the system’s inputs, while Table 3.3.2 shows the outputs. The subsequent five figures are each of the User Interfaces created by the system by the inputs and outputs from Tables 3.3.1 and 3.3.2.

| **Input # and name** | **Source** | **Description** | **Units of measure** | **Data format** | **Cross references** |
| --- | --- | --- | --- | --- | --- |
| UI 2.1 main settings button | button | click | click | click | US-2 |
| UI 2.2 main history button | button | click | click | click | US-2 |
| UI 4.1, 5.1 back to main button | button | click | click | click | US-4, US-5 |
|  | button | Stories 4/5 reuse UI 2.1, 2.2 for settings and history |  |  |  |
| UI 4.2, 4.3 | checkbox | settings toggle; click | on/off | on/off | US-4 |
| UI 4.4 | button | “SAVE” button | click | click | US-4 |
| UI 4.5 | text box | Target Directory entry | character | String | US-4 |

Table 3.3.1: Summary of User Interfaces

| **Output Interfaces** | **Destination** | **Valid range, accuracy, and/or tolerance** | **Units of measure** | **Data formats** | **Cross References** |
| --- | --- | --- | --- | --- | --- |
| O2.1 System OK | Screen | none | 0, 1 | Red/green | US-2 |
| O2.2 Threat | Screen | none | 0, 1 | Red/green | US-2 |
| O4.1 Settings | Screen | none | char | text | US-4 |
| O5.1 History | Screen | none | char | text | US-5 |

Table 3.3.2: Summary of Outputs

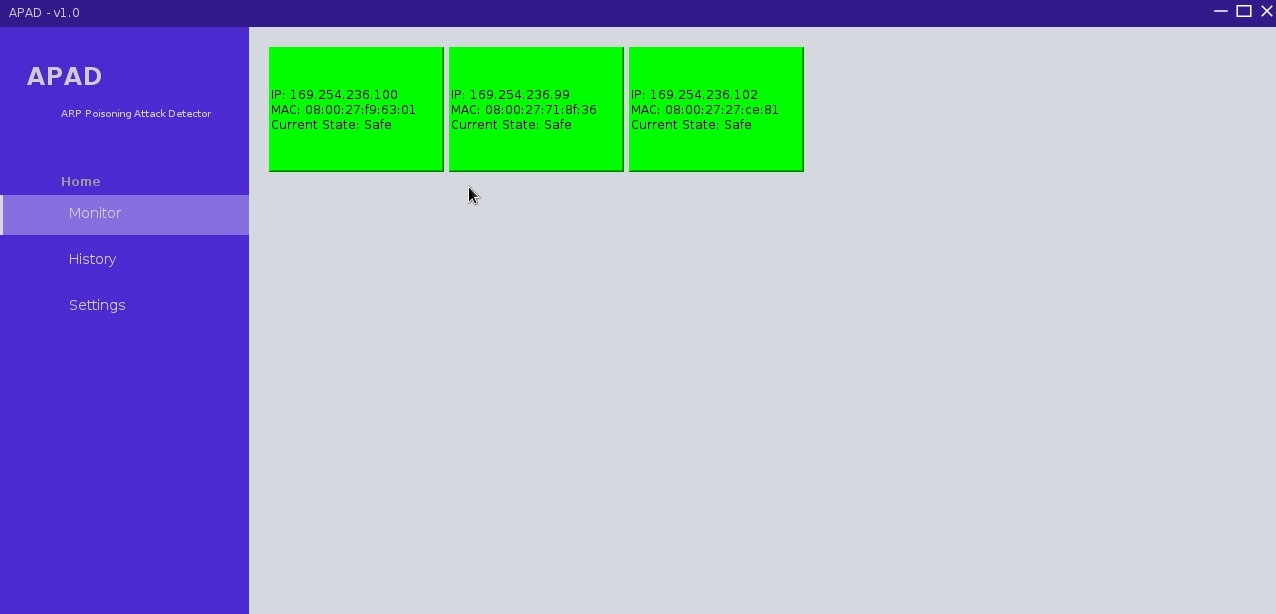
****

Figure 3.3.1: UI-1, Main Screen

Figure 2.3.1 shows our design for the main page of our program. This will be where the user will be able to monitor their network. Figure 2.3.2, shown below, demonstrates what will to be shown to the user when an attack is detected.

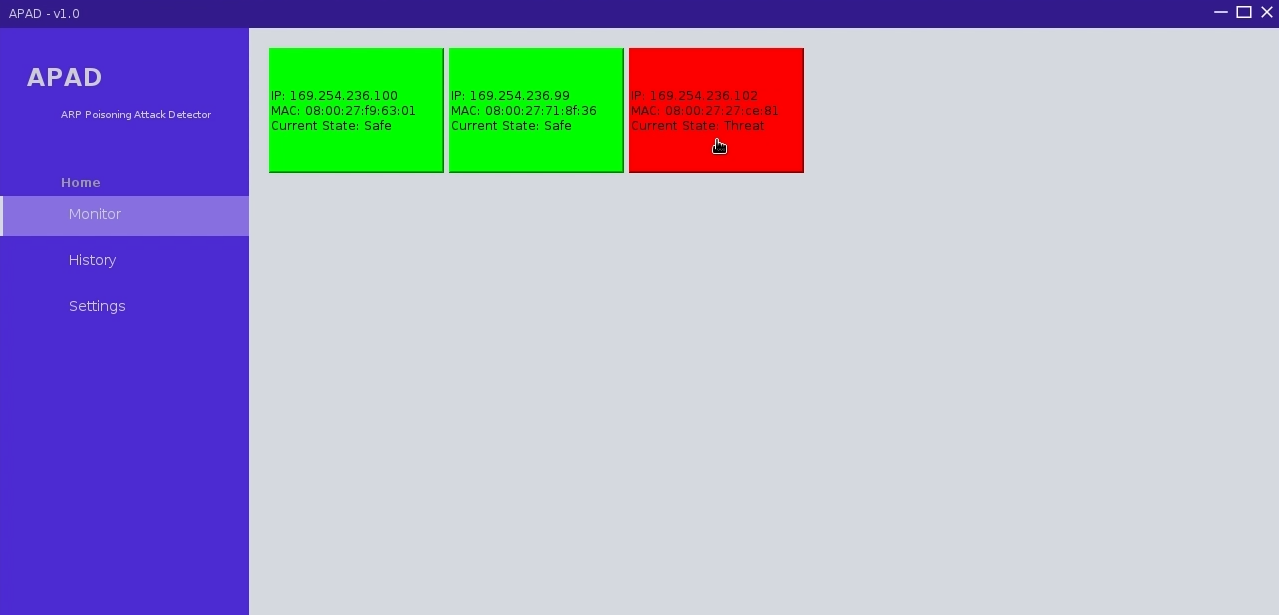
****

Figure 3.3.2: UI-1, Main Screen (Attack)

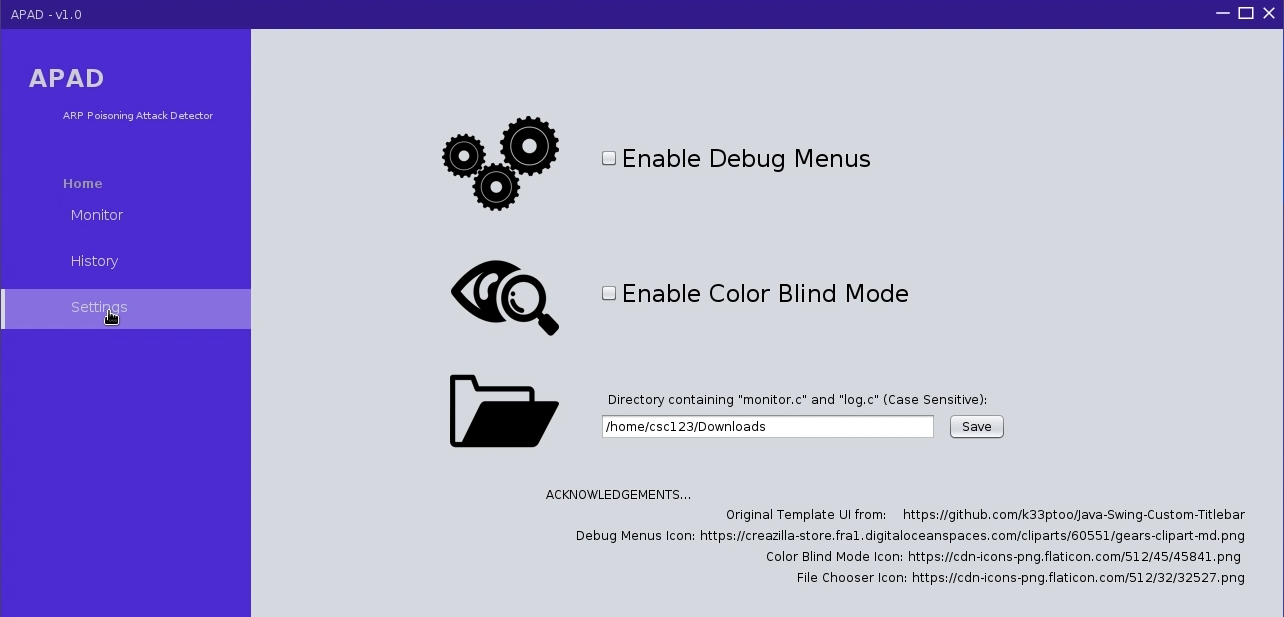
****

Figure 3.3.3: UI-2, View Settings

Figure 2.3.3 shows our design for the System Settings page. Here, the user can make personal changes to their window.

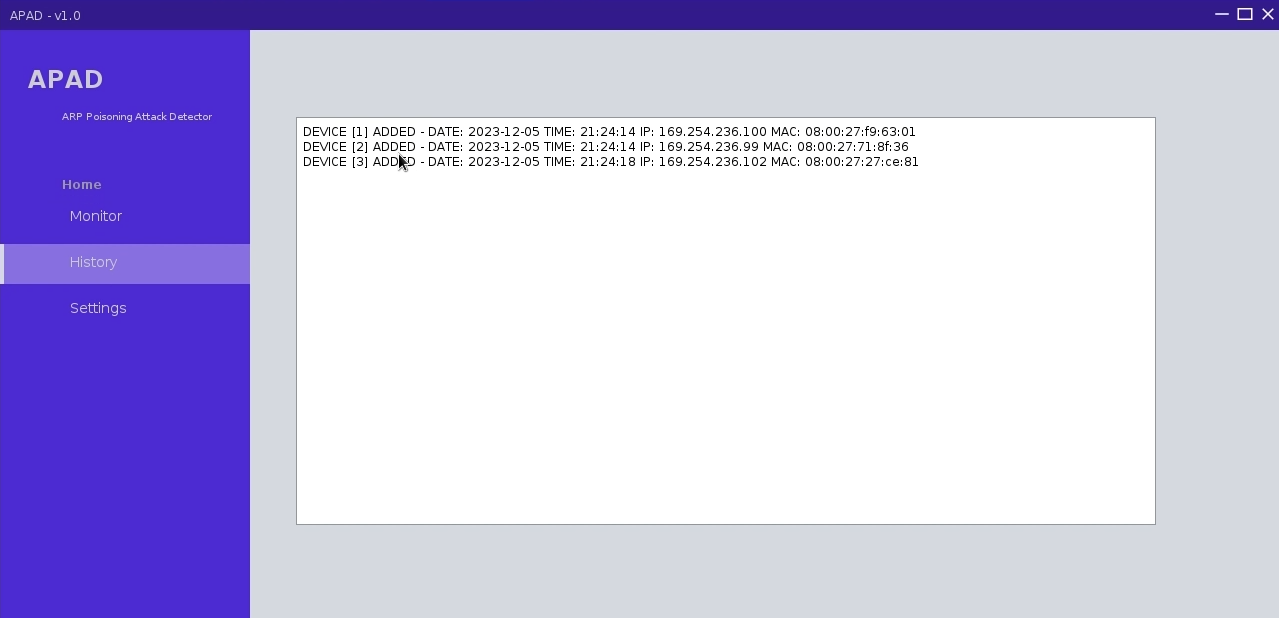
****

Figure 3.3.4: UI-3, View History

The “View History” screen, detailed in the above figure, shows the screen that will allow the user to view past data. This will be accessed from the main page.

**3.4 Functional Requirements:**

The functional requirements of our project are requirements that directly impact the user’s experience. They include features the user can see and interact with directly.

| **Name** | FR-1-Run |
| --- | --- |
| **Description** | Gathers MAC/IP address from ARP table and sends them to .txt file. |
| **Rationale** | Sends data from ARP table to .txt file for analysis by monitor.c |
| **System Behavior** | log.c will send IP/MAC addresses and timestamps to a .txt file. |
| **References** | US2 |

Table 3.4.1: FR-1

| **Name** | FR-2 - Monitor |
| --- | --- |
| **Description** | Detect changes between IP/MAC pairings and block attacks |
| **Rationale** | Detects ARP attack |
| **System Behavior** | Monitor.c will analyze this text file and alert if an IP address already in use by another MAC address is taken by a new MAC address. |
| **References** | US2 |

Table 3.4.2: FR-2

**3.5 - Non-Functional Requirements**

Non-functional requirements are the requirements that take place “behind the scenes”. They are necessary for the system to work as intended, but the user will not be able to directly interact with them.

**3.5.1 - Performance Requirements**

**NF-1**: Alerts should be sent to the Network Administrator within 50 milliseconds of the attack being detected, assuming the user’s computer is able to run the software without any slowdowns

**NF-2**: Alerts should only be sent if a suspicious IP change is detected; a new device joining the network should not trigger an alert.

**3.5.2 - Logical Database Requirements**

**NF-3**: The software needs to have access to the data contained within the ARP Table

**NF-4**: The software needs to be able to read from and write to a text file functioning as the log for all network events

**3.5.3 - Engineering Requirements**

**NF-5**: An older wi-fi router is required for this software to function properly; newer hardware already has the protections provided by this software

**NF-6**: At least three machines are required for this software to demonstrate all functionality; The host, running the software, the victim, who exchanges information with the host, and the attacker, who intercepts the victim using an ARP poisoning attack

**3.5.4 - Security Requirements**

**NF-7**: Text-based user input should be minimized to prevent other attacks such as command injection. The user interface will be largely click-operated.

**3.5.5 - Reliability Requirements**

**NF-8**: The software must be able to run continuously for at least 8 hours, the length of a normal work day, without failure.

**NF-9**: The software should be able to catch attacker devices reliably, with minimal false positives or negatives.

**3.5.6 - Robustness Requirements**

**NF-10**: The software should be able to handle all error conditions without crashing. For example, if the log.txt file becomes too large, older data should be overwritten.

**Chapter 4: Standards & Constraints**

In this chapter, the standards upheld by, and constraints imposed upon, our project, will be detailed in full. However brief, it is important to ensure there are no violations in these constraints.

**4.1 - Standards:**

Engineering standards are constraints imposed by software, hardware, and communication. Our project will follow standard TCP/IP for communications with other devices (Internet protocol) [15]. In addition to this, the user’s password will be protected with a common hashing algorithm, AES (Advanced Encryption Standard), to ensure that the password is not accessible to an outside entity [16].

**4.2 Constraints:**

Additionally, our project poses a few ethical and legal constraints that we will have to navigate. Because the user of our project is expected to be a legitimate network admin, this person will have access to everyone’s data that is on that network, so it is important that these privileges are not taken advantage of. Ethical and legal risks are amplified depending on the type of data being sent over your network, whether it is intellectual property, HIPAA information, etc. For these reasons, it is important that the access of information on your network is carefully controlled and monitored.

**RC-1**: Ethical - Information passed along the network must purely be used to identify ARP poisoning attacks.

**RC-2:** Legal - Sensitive information passed along the network must be properly protected by the system; the user should not be allowed to actually read information being passed along the network.

**RC-3:** Legal - Launching an ARP poisoning attack on a real-world network is sure to cause some legal issues. In order to avoid launching a cyberattack on a real-world network, we will be testing our project using virtual machines, connected to a virtual network.

**Chapter 5: Project Design**

In this chapter, each part of the system, their classes, functions, and interactions, will be described in much more detail. Below is a diagram of all parts of the system and the functions contained within.

**5.1 - Overview of System Components**

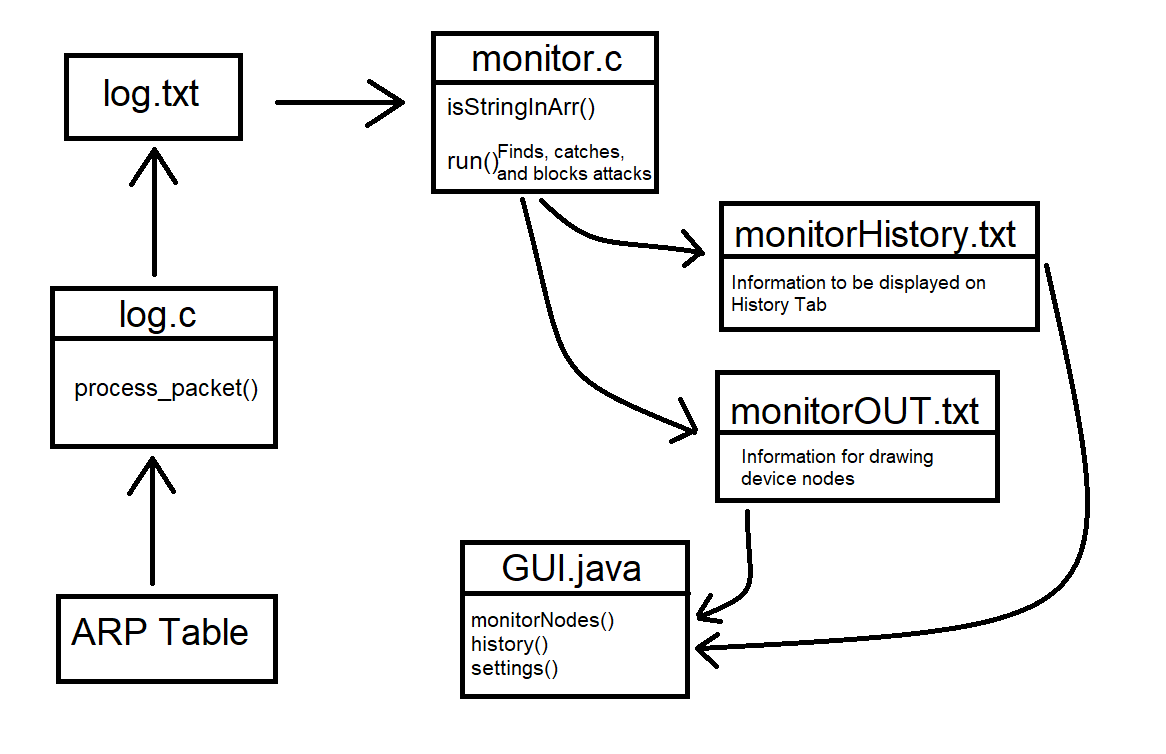
****

Figure 5.1.1: Class Diagram

The ARP Table is a system file containing the IP and MAC addresses of all devices connected to the network. The information contained within is cleaned-up by log.c in order to create and maintain log.txt, which contains only the data necessary for the other parts of the system to run. Monitor.c is the core of the system, analyzing the information contained in log.txt and interacting with the user through their input from GUI.java. Monitor.c is also responsible for creating monitorHistory.txt and monitorOUT.txt, used by the GUI in order to show the running log, and to draw nodes with device information, respectively. The above figure (figure 4.1.1), describes this relationship.

**5.2 - Structure and Relationship**

The following section will go further into detail about each part of the system and how they interact with each other.

**ARP Table**

A system file found in any device connected to the network. This file contains the IP and MAC addresses of all devices connected to the network. This information is used by log.c - all other information contained within is unnecessary, and therefore discarded - to create and maintain log.txt. This allows the system to determine whether or not an ARP poisoning attack is taking place.

**log.c**

This program reads from the ARP Table and analyzes incoming packets using third-party libraries such as libpcap and netinet [3] [4]. It grabs the IP and MAC addresses of devices on the network, and writes them to log.txt.

**log.txt**

log.txt is the text file created by log.c. It contains only the IP and MAC addresses of devices on the network, and is updated whenever a new event takes place on the network. It is used by GUI.java to show the user the history of events on the network, and by monitor.c in order to detect suspicious activity.

**monitor.c**

This C program is the core of the system. It is responsible for continuously analyzing the network information and alerting the user of any suspicious changes to the ARP Table. Additionally, it creates monitorOUT.txt and monitorHistory.txt, to be used by GUI.java in order to visually display information to the user.

**GUI.java**

The user interface of the project, written in Java using the Swing library [2]. The user interface is mostly click-based, with the exception of the user’s login information being typed. User input is sent to monitor.c as commands; monitor.c will have the user interface update when certain events take place as well, such as a given device’s node turning red when an attack is detected.

**monitorOUT.txt**

A text file containing the IP, MAC, and current state of each device on the network. It is used by the GUI to draw green (safe) or red (threat) nodes on the screen, corresponding to each device, and displaying their IP and MAC address within.

**monitorHistory.txt**

A text file containing the key events that took place on the network while the program was being run, such as threats being detected or new devices joining the network. It is used by the GUI to display this information in a neat, concise way to the user.

**5.3 - DETAILED COMPONENT DESCRIPTION**

This section, and the tables contained within, describe in full detail each system component, and the processes contained within each one (if applicable).

| **SC-1** | GUI.java |
| --- | --- |
| **Type** | Java Program |
| **Purpose** | Provides a user-friendly graphical interface for the user to interact with the program |
| **Inputs** | ARP Table and alerts from monitor.c  monitorOUT.txt  monitorHistory.txt  Clicks from User; String corresponding to directory of monitor.c and log.c |
| **Outputs** | Displays constantly-updating network nodes and logs from log.txt |
| **Data** | N/A |
| **Internal**  **Structure** | monitorHistory.txt provides the log and all network events to the GUI to display  monitorOUT.txt provides information to create and display nodes |
| **Processing** | (US-1) S1. User enters target directory info; This is used to load monitorHistory.txt and monitorOUT.txt from the proper location  (US-2, US-5) S2. monitor.c will update monitorOUT.txt and monitorHistory.txt periodically  S3. GUI.java will display the latest updates to monitorHistory.txt as they occur  S4. GUI.java will also display each device connected to the network as its own node using information from monitorOUT.txt  (US-2) S5. Suspicious devices will be marked with a RED node and packets to and from this device will be automatically dropped using an iptables rule  (US-4) S6. GUI.java will display all available settings to the user  S9. If User changes settings, GUI will adjust behavior accordingly  Color Blind Mode → Change colors of nodes from green/red to blue/scarlet in order to better accommodate color blind users  Enable Debug Menus → Allows user to utilize various testing buttons to ensure the program is working as intended |
| **Dependencies** | Needs to know the current state of the network and all connected devices |
| **Resources** | Java, monitorOUT.txt, monitorHistory.txt, monitor.c |
| **Cross-reference** | All UIs and User Stories |

Table 5.3.1: GUI.java

| **SC-2** | log.c |
| --- | --- |
| **Type** | C Program |
| **Purpose** | Monitors network, maintains log.txt |
| **Inputs** | Network information from ARP table |
| **Outputs** | log.txt |
| **Data** | N/A |
| **Internal**  **Structure** | Uses a sniffer library (libpcap, netinet) to capture packets and analyze the ARP table |
| **Processing** | (US-2) S1. log.c reads and analyzes ARP table  S2. Any changes in the ARP table will be added to log.txt |
| **Dependencies** | Network connection is required |
| **Resources** | libpcap, netinet |
| **Cross-reference** | US-2, SC-3, SC-4 |

Table 5.3.2: log.c

| **SC-3** | ARP Table |
| --- | --- |
| **Type** | System File |
| **Purpose** | Contains information about all devices on network |
| **Inputs** | N/A |
| **Outputs** | N/A |
| **Data** | IP addresses, MAC addresses |
| **Internal**  **Structure** | Contains the MAC and IP address of all devices connected to the network |
| **Processing** | None |
| **Dependencies** | Network connection is required |
| **Resources** | None |
| **Cross-reference** | SC-2 |

Table 5.3.3: ARP Table

| **SC-4** | log.txt |
| --- | --- |
| **Type** | text file |
| **Purpose** | Contains network events and changes |
| **Inputs** | Data from log.c |
| **Outputs** | None |
| **Data** | None |
| **Internal**  **Structure** | None; plain text |
| **Processing** | None |
| **Dependencies** | None |
| **Resources** | None |
| **Cross-reference** | SC-1, SC-2 |

Table 5.3.4: log.txt

| **SC-5** | monitor.c |
| --- | --- |
| **Type** | C program |
| **Purpose** | Monitors network, identifies threats, maintains monitorOUT.txt and monitorHistory.txt |
| **Inputs** | log.txt |
| **Outputs** | monitorOUT.txt (Node information) and monitorHistory.txt (Running log) |
| **Data** | N/A |
| **Internal**  **Structure** | Analyzes updates to log.txt and automatically drops packets in response to any suspicious changes to the ARP Table. |
| **Processing** | (US-2) S1. If a suspicious IP address change is detected, adds an iptables rule to automatically block packets to/from the offending device |
| **Dependencies** | Network connection is required |
| **Resources** | log.txt |
| **Cross-reference** | US-2 |

Table 5.3.5: monitor.c

| **SC-6** | monitorHistory.txt |
| --- | --- |
| **Type** | text file |
| **Purpose** | Contains key network events to be read by GUI in History tab |
| **Inputs** | None |
| **Outputs** | Read by GUI to keep a running log of events |
| **Data** | None |
| **Internal**  **Structure** | None; plain text |
| **Processing** | None |
| **Dependencies** | monitor.c |
| **Resources** | log.txt |
| **Cross-reference** | SC-1, SC-2, SC-5 |

Table 5.3.6: monitorHistory.txt

| **SC-7** | monitorOUT.txt |
| --- | --- |
| **Type** | text file |
| **Purpose** | Contains data to be used by GUI to draw nodes containing device information |
| **Inputs** | None |
| **Outputs** | Read by GUI to generate and color nodes |
| **Data** | None |
| **Internal**  **Structure** | None; plain text |
| **Processing** | None |
| **Dependencies** | monitor.c |
| **Resources** | log.txt |
| **Cross-reference** | SC-1, SC-2, SC-5 |

Table 5.3.7: monitorOUT.txt

**log() - Back End**

This function will begin the process of reading data from the ARP table and writing it to a text file for further analysis. When this program is run, it will begin the monitor.c program, which analyzes the data that we write to log.txt. Inputs for this program are text strings and timestamp data from the ARP table. This applies to User-Story 2.

**monitor() - Back End**

This function analyzes timestamp, MAC, and IP data that is written to log.txt. This program will not alert on network activity that does not match the symptoms of an ARP poisoning attack. When a MAC or IP address is already paired together, and a different MAC attempts to use an IP address that is in use by another MAC, this will cause an ALERT to the user. Inputs for this program are text strings and timestamp data from the ARP table. This applies to User-Story 3.

**blockDevice() - Back End**

This function, is the blocking function of monitor.c. It goes through each of the IP/MAC pairings, and checks what each one *was*, to what each one *is*. If there is a suspicious change detected, a new iptables rule will be added to drop packets to, and from, the offending device. This applies to User-Story 3.

**5.4 – Tools and Methods**

Our methodology for testing involves using 3 contained virtual machines [7], all connected to the same virtual LAN network. One virtual machine is treated as a typical user on the network. The second virtual machine is treated as a User which is the admin of the network, and will be running the client-sided detection software. The last machine is the hacker, who is using Ettercap in order to send poisoned ARP packets through the first user [5]. For testing if the ARP poisoning attack from the hacker machine was successful, we will be using Wireshark Network Analyzer in order to capture the packets being sent [10]. For using Wireshark, we need to select the LAN interface and start capturing packets. Then start the Ettercap interface on the hacker machine. Finally, select ‘Targets’ and scan for hosts on the network, select the IP address of the user machine, and then click ‘Start’ and ‘Launch ARP Poisoning Attack’ [6]. If done correctly, the Wireshark analysis will start picking up packets from the IP address of the hacker machine, confirming that the ARP Poisoning Attack was successful. Our virtual testing environment will directly translate to real-world implementations within the constraints defined in Chapter 3.2.

**5.5 - Design Alternatives**

Initially, we had a design that utilized one singular C program for the back-end services. This program was responsible for parsing through the ARP table, eliminating needless information, and analyzing the data. This however, proved too difficult and was not manageable. To overcome this, we implemented a design where we will utilize two C programs to do these jobs. This makes the task of analyzing data from the ARP table simpler and fits our needs better.

**Chapter 6: Test Plan**

In this chapter, our plans for testing the system will be detailed. There are unit tests, functionality tests, and integration tests, which will be further described in section 6.1.

**6.1 - Test Plan**

First, our testing must be organized to verify that our program does what it is supposed to do. Each user story will have a test associated with it to verify that the system functions properly. We must identify which tests are unit tests, which tests are for the functionality of our program, and which tests are integration tests - which verify that our program integrates with its different parts correctly.

**Unit tests of code:**

* Verify that our code functions as intended
* Verify that our login function works properly; that our log.c program works properly, and that our monitor.c program works properly.
* Integration tests for multiple design components interacting with each other.

**Integration:**

* Ensure that our program properly handles data from the ARP table
  + Be able to read timestamp, MAC, and, IP data
* Ensure that our program properly maintains settings.txt and log.txt. These are two text files that are continuously maintained

**Non-Functional:**

* Ensure that our program operates as intended, and is user friendly
* Ensure security of our program, due to the fact that our program may be used to access sensitive material.
* Ensure that we catch as many errors as possible
* Input sanitation, where applicable

**6.2 Test Cases**

The following five tables show the five main tests that we will perform on our system. These extensive tests cover all parts of the system, and will allow us to ensure everything is running smoothly.

| **Name** | **TC - 1. Verify that directory selection works** |
| --- | --- |
| **Type** | unit |
| **Description** | Test that the proper directory can be loaded, and that inputs are not overly restrictive |
| **Preconditions** | App is opened, settings screen is selected |
| **Basic course of events** | 1. Enter directory information.  2. User hits “ENTER” or clicks “SAVE” button next to text box |
| **Expected results** | Proper directory will load information to be displayed in main and history tabs |
| **Acceptance criteria** | Program does not crash |
| **Cross-references** | US-1, UI-2 |
| **Scheduled Sprint** | 3 |

Table 6.2.1: Directory Selection Testing

| **Name** | **TC - 2. Verify that program can properly handle ARP data** |
| --- | --- |
| **Type** | integration |
| **Description** | Test that program can properly parse, iterate, and read data from the ARP table, in the way we expect |
| **Preconditions** | program is opened, user logged in, and run. |
| **Basic course of events** | 1. User opens app.  2. User logs in  3. RUN action selected |
| **Expected results** | Timestamp, MAC, IP data is read from ARP table and we are able to analyze it. |
| **Acceptance criteria** | Data is properly formatted for analysis |
| **Cross-references** | US2,UI-1 |
| **Scheduled Sprint** | 0 |

Table 6.2.2: Testing of log.c and log.txt

| **Name** | **TC - 3. Verify that program can catch ARP attack** |
| --- | --- |
| **Type** | unit |
| **Description** | Test that program can accurately detect ARP attack |
| **Preconditions** | Program is run, ARP attack is happening on network |
| **Basic course of events** | 1. Program is run  2. ARP attack is happening  3. System catches ARP attack by recognizing symptoms  4. Program displays ALERT to user. |
| **Expected results** | A MAC/IP address that is already in use is being used by an attacker, program recognizes this as an attack. |
| **Acceptance criteria** | ARP attack is detected and blocked, and user is alerted |
| **Cross-references** | US2,UI-1 |
| **Scheduled Sprint** | 2 |

Table 6.2.3: Testing of monitor.c

| **Name** | **TC - 4. Verify that program can maintain a running log** |
| --- | --- |
| **Type** | integration |
| **Description** | Test that program can maintain a log of timestamp, MAC, IP data |
| **Preconditions** | Program is run, data is being sent to a text file |
| **Basic course of events** | 1. Program is run  2. Data is sent to text file  3. File is properly written to, line by line |
| **Expected results** | Data is maintained in this log, and does not disappear when closing out of program. |
| **Acceptance criteria** | Data is properly maintained in log and does not delete without users command to do so. |
| **Cross-references** | US4,UI-2 |
| **Scheduled Sprint** | 1 |

Table 6.2.4: Further Testing of log.c

**Chapter 7: Project Plan**

In this chapter, the overall project plan will be discussed, from the month-long sprints, to the risk and teamwork plans.

**7.1 – Sprints**

Our project will be divided into four, four-week long sprints, in which we will focus our efforts on completing a small part of the project. Each sprint has a defined goal and backlog of necessary tasks to be completed.

**Sprint 0: Foundational Work**

* **Goal**: Begin basic development of project to be refined in future sprints
* **References**: All UI, FR-1, FR-2, NF-3, NF-4, NF-12, All US, SC-1, SC-2
* **Starting** **date**: May 2023 **Duration**: 4 weeks
* **Sprint master**: John Bilella
* **Demo goal**: Basic user interface design; log.c has basic functionality
* **Backlog**:

| # | Backlog | Hardware/ Software  resources | Possible team members | Estimated  hours | Points/  Difficulty | Cross-references to requirements, design components |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Develop log.c | Virtual Machines | Nick Lara | 12 | 8 | US-2. US-5, FR-1, NF-3, NF-4, NF-12, SC-2 |
| 2 | Design GUI.java | Virtual Machines | John Bilella, Nick Masci | 8 | 7 | All UI, All US, SC-1 |

Table 7.1.1: Sprint 0 Backlog

**Sprint 1: Completion of Basic Components**

* **Goal**: Get all parts of system to have basic functionality
* **References**: All UI, FR-1, NF-3, NF-4, NF-12, All US, SC-1, SC-2, SC-5
* **Starting** **date**: September 2023 **Duration**: 4 weeks
* **Sprint master**: Nick Lara
* **Demo goal**: Show improved user interface, monitor.c working in command line
* **Backlog**:

| # | Backlog | Hardware/ Software  resources | Possible team members | Estimated  hours | Points/  Difficulty | Cross-references to requirements, design components |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Test log.c | Virtual Machines | Nick Lara | 4 | 6 | US-2. US-5, FR-1, NF-3, NF-4, NF-12, SC-2 |
| 2 | Improve GUI.java | Virtual Machines | John Bilella, Nick Masci | 9 | 7 | All UI, All US, SC-1 |
| 3 | Develop monitor.c | Virtual Machines | All | 15 | 9 | FR-2, SC-5, US-2 |

Table 7.1.2: Sprint 1 Backlog

**Sprint 2: Interconnection of System Components; Documentation**

* **Goal**: Connect all components together, create documentation/manuals, and further optimize programs
* **References**: All UI, All US, FR-2, NF-9
* **Starting** **date**: October 2023 **Duration**: 4 weeks
* **Sprint master**: Nick Masci
* **Demo goal**: Show programs’ improved performance, manuals, etc.
* **Backlog**:

| # | Backlog | Hardware/ Software  resources | Possible team members | Estimated  hours | Points/  Difficulty | Cross-references to requirements, design components |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Interconnect each program | Virtual Machines | All | 5 | 5 | FR-2, All US |
| 2 | Improve GUI | Virtual machines | Nick M, John | 6 | 4 | All UI, NF-9 |
| 3 | Create documentation and manuals | Text editor, virtual machines | All | 6 | 3 | N/A |

Table 7.1.3: Sprint 2 Backlog

**Sprint 3: Final Testing and Optimization**

* **Goal**: Completely finish all system components; test and optimize as needed
* **References**: NF-10, NF-11, NF-12
* **Starting** **date**: November 2023 **Duration**: 4 weeks
* **Sprint master**: John Bilella
* **Demo goal**: Demonstration of our project nearly (if not fully) completed
* **Backlog**:

| # | Backlog | Hardware/ Software  resources | Possible team members | Estimated  hours | Points/  Difficulty | Cross-references to requirements, design components |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Rigorous testing and final bug fixes | Virtual machines | All | 12 | 5 | NF-10, NF-11, NF-12 |
| 2 | Optimize speed and performance to fullest potential | Virtual Machines | All | 5 | 6 | NF-10, NF-11, NF-12 |

Table 7.1.4: Sprint 3 Backlog

**6.2 - Risk Plan**

The risk plan, shown below, highlights some key risks we may run into as we progress through our project, and how we may go about addressing these risks in order to prevent any problems with the system’s functionality later on, and ensure compliance with all standards and constraints.

| **Name** | Risk 1: Vulnerabilities in code |
| --- | --- |
| **Description** | Vulnerable sections of program such as text boxes being subject to command injection attacks |
| **Probability** | Extremely likely |
| **Impact** | Through command injection, a malicious user can take control of the system by typing malicious code into the username and password boxes |
| **Actions Necessary** | Input sanitization will mitigate this risk almost entirely, by preventing suspicious characters that may cause unintended behavior from being entered |

Table 7.2.1: Risk 1

| **Name** | Risk 2: System is difficult to understand by user |
| --- | --- |
| **Description** | Users unfamiliar with the intricacies of networking may be lost and confused as to the purpose of our project |
| **Probability** | Somewhat likely |
| **Impact** | A lack of user friendliness will make this project less impactful on the world and usable by fewer people |
| **Actions Necessary** | Manuals, documentation, and a simple UI will allow users to comfortably navigate the program |

Table 7.2.2: Risk 2

| **Name** | Risk 3: Sensitive Information on Network |
| --- | --- |
| **Description** | Sensitive information may be passed along the network, which is actively being analyzed by log.c and monitor.c |
| **Probability** | Very likely |
| **Impact** | If the user is able to access all sensitive information being passed along the network, it would be in violation of our ethical, and potentially legal, constraints, as it would violate the privacy of other network members’ information |
| **Actions Necessary** | We must ensure log.c discards all information other than IP, MAC, and the timestamp from each packet. |

Table 7.2.3: Risk 3

| **Name** | Risk 4: Dynamic IP Address Problem |
| --- | --- |
| **Description** | There is a chance that dynamically-assigned IP addresses will cause problems for log.c |
| **Probability** | Somewhat likely |
| **Impact** | log.c may be unable to properly process packets passed through a dynamically-assigned IP network |
| **Actions Necessary** | If this is the case, an additional DHCP (Dynamic Host Configuration Protocol) handler function will be added to log.c |

Table 7.2.4: Risk 4

**7.3 - Teamwork Plan**

To accomplish the goals of our project in a timely manner, it is imperative that we plan and divide our work smartly. Work will be divided amongst members equally for all parts of the project. Additionally, the team will utilize a Discord server for communication, as well as meeting at least twice a week for about 30 minutes to discuss our project. To maintain organization of our code, we will use GitHub, where we will be able to submit our code to each other to ensure that all members have the most up-to-date version of our necessary files.

**Chapter 8: Methods**

In Chapter 8, the algorithms, tools, languages, and libraries that were used in the project’s implementation, as well as any new knowledge that group members acquired during the course of the project.

**8.1 – Technical Methods**

The most fundamental algorithm of this application is the ARP Table processing algorithm in log.c. First, the network interface is opened in order to capture and process packets. Then, libpcap’s ARP filter is compiled and applied so that only ARP packets are captured [17]. The output file, log.txt, is then opened within the program to be written to when the packet processing algorithm is called. Finally, an endless loop begins to repeatedly call the packet processing algorithm.

Within the actual packet processing algorithm, the ARP filter is applied to each packet’s data to obtain only the ARP packets, as those are all that are needed to detect ARP poisoning attacks. The ARP header is parsed, and the IP and MAC addresses are converted to strings. The current time is also obtained, so each network event can be timestamped. The target and sender IP and MAC addresses, as well as the timestamp, are then written into a new line in log.txt.

Log.txt is then used in monitor.c, where the actual blocking of attacks takes place. The checking algorithm is based upon string comparisons. It can return 0, 1, or 2, depending on the result of the comparison. It compares the IP and MAC pairings of each new line in log.txt, and if one of the pairings is changed, a new iptables rule is created to drop all packets sent to, and received from, the offending device’s MAC address [18]. Additionally, a flag pertaining to whether or not the device is a threat will be activated, marking the device as a threat. A threat will cause the checking algorithm to return 2. If the device is new to the network, the algorithm will return 1. Otherwise, if everything is normal, and the IP and MAC pairing is unchanged, the algorithm returns 0.

The main loop of monitor.c, the monitor algorithm, is as follows. First, monitor.c will open log.txt to read each line of new IP and MAC pairings. It will also create monitorOUT.txt, for the GUI to generate device nodes, and monitorHistory.txt, for the GUI to catalog important network events. Next, each line of log.txt is converted into an array of character strings in order to isolate each IP and MAC address. These IP and MAC addresses are then fed into the checking algorithm. If the checking algorithm returns 1, the monitor algorithm will output both to the console and to monitorHistory.txt that a new device has been added, with its respective IP and MAC address. Additionally, it will be added to monitorOUT.txt to be displayed in the GUI. If the checking algorithm returns 2, the monitor algorithm will output to monitorHistory.txt that an attack has been detected, noting the attacker’s and victim’s IP address, as well as modifying the attacker device’s threat flag in monitorOUT.txt so the GUI can notify the user of the threat.

As for the GUI, there are two algorithms that are used to generate device nodes: the individual node algorithm, and the map drawing algorithm. The individual node algorithm creates nodes at a set size; 175 pixels wide, and 125 pixels tall, and at a given set of coordinates provided by the map drawing algorithm. The algorithm checks the device’s IP, MAC, and current state from monitorOUT.txt, and fills in the information accordingly. If the device is currently in a normal, safe state, it is colored green - or blue, if the color-blind setting is active. If a device is recognized as an attacker, its color is changed to be red - or scarlet, if the color-blind setting is active. Additionally, text is displayed on the node denoting that it is “Safe”, or a “Threat”, below the device’s IP and MAC address.

The map drawing algorithm aims to place each node in a 4-by-4 grid - The application is only intended to display 16 devices at once. It will still attempt to display additional devices, but they may fall out of the visible range of the screen. Each node is placed into an array; the index of each node is used in order to calculate the proper coordinates to place it at. Once the coordinates are calculated, the individual node drawing algorithm is given these coordinates to draw the corresponding node at the proper place.

The other notable algorithm of the GUI, the history algorithm, reads each line of monitorHistory.txt, and displays them in a text area in the GUI’s “History” tab. Each loop, it also checks each line to see if it has already been printed in the text area, to avoid having the same lines repeatedly displayed. If too many lines are printed, the user is able to scroll back and forth within the text area.

As the blocking is done within monitor.c, and the GUI is just there to visually display information to the user, the UI runs each algorithm only once every second, to avoid the application using too much memory.

**8.2 - Tools**

Libpcap was used in the creation of log.c. This open-source C library includes various functions that allow for the capturing of packets sent and received in the network, as well as allowing for these packets to be filtered to only ARP packets [19].

Netinet is another open-source C library that was used in log.c. This library contains definitions for various connection types and protocols, allowing the application to parse and manipulate this information easily. Specifically, ip.h [20], udp.h [21], tcp.h [22], and if\_ether.h [23] were used from netinet.

Apache Netbeans is an open source IDE with drag-and-drop functionality that was used to aid in the designing of GUI.java [24]. Design components can simply be dragged into place, and their properties edited in various ways, and Netbeans will automatically generate code to make these pieces function properly. Code can still be written manually within Netbeans as well, as it is easier to program certain functions by hand as opposed to having them created automatically - such as the node drawing algorithm.

Java Swing is an open source framework for creating GUI components in Java [25]. In tandem with Netbeans, Java Swing is a powerful tool used to build the visual components of the GUI. It also allows actions to be assigned to GUI components, allowing for clickable buttons, toggleable settings, and the ability to switch between each tab of the user interface.

Ettercap, while not used in the creation of any programs, was still an essential part of testing the application. Ettercap is an open source tool that allows the user to perform man in the middle attacks - such as ARP poisoning attacks - on a network [26]. To avoid any legal issues, this was only performed in a virtual network, between virtual machines, but it ensured that monitor.c was able to catch and block any threats to the network.

**8.3 - Programming Languages**

In this section, the programming languages used to create each part of the application will be listed.

* **C** - monitor.c and log.c are both written in C, as it is a low-level language that can interact more easily with system files and is overall more memory-efficient
* **Java** - The User Interface is written in Java, using the Swing library and Netbeans IDE
* **HTML** - A small amount of HTML is used within the User Interface, just to display the text within each device node [27].

**8.4 - Use of Open Source Code**

For the User Interface, an open source template was used as a baseline [28]. This template contained multiple tabs, as well as a stylish, custom title bar. For testing, we utilized Oracle VM VirtualBox and Ettercap, both open source programs. All other functionality, such as settings, nodes, and the display of network history events, were added in as part of the project.

**8.5 - New Tools and Knowledge**

The table below details all the new tools used and knowledge acquired by each group member, including how they learned it and where it was used in the project.

| **Name** | **Type** | **Person who Learned It** | **How They Learned It** | **Where it was Used** |
| --- | --- | --- | --- | --- |
| Netbeans | IDE | Nick M | Documentation [24] | Creation of GUI |
| Java Swing | Framework | Nick M | Documentation [25] | Creation of GUI |
| Ettercap | MITM attack software | Nico L | Documentation [26] | Attacking victim device to test application |
| Ettercap | MITM attack software | Nick M, John B | Taught by Nico | Attacking victim device to test application |

Table 8.5: New Tools and Knowledge

**Chapter 9: Implementation and Results**

**9.1 – Implementation Strategy**

Since we knew our software would be built off of three different programs; and that we had three team members, we approached the project by putting each team member in charge of one of the programs. Nico was in charge of writing log.c and setting up & managing the virtual environments, John was in charge of writing monitor.c, and Nick was in charge of writing the Apache NetBeans UI. When it came to integration, the team members worked together to successfully integrate the three programs together. John & Nick communicated with each other in order to get monitor.c working within the UI, while Nico & Nick worked together to get the UI to read from log.txt generated from log.c. We communicated via Discord and used a shared Google Drive to keep track of the latest versions of our software as well as to keep backups.

**9.2 - Sprints**

In this section, each team member’s specific tasks during each sprint will be detailed.

**9.2.1 - Sprint 1**

* **Goal: Find a new open-source GUI, create nodes randomly and display them on the GUI.**
* **Deadline: 10/5/23**
* **Completion Date: 10/5/23**
* **Backlog: Add log.txt tab to UI, add settings tab, continue modifying the UI from the original template.**

| **Task** | **Task Description** | **Team Member(s)** | **Estimated Hours** | **Actual Hours** |
| --- | --- | --- | --- | --- |
| T1 | Find suitable UI template | All | 15 | 13 |
| T2 | Get template UI running in the NetBeans IDE | Nico L | 2 | 7 |
| T3 | Modify the template UI with the ability to draw nodes and handle right-click events | Nick M | 4 | 9 |
| T4 | Start implementing the modified UI with monitor.c | John B | 3 | 4 |
| T5 | Get log.c working on John’s VM | John B | 4 | 3 |
| T6 | Get UI to interface from log.txt | Nico L | 3 | 6 |
| T7 | Create tab in UI for recording log.c | Nico L | 5 | 6 |
| T8 | Create nodes from a file instead of just randomly generating them | Nick M | 5 | 7 |

Table 9.2.1: Sprint 1 Backlog

**9.2.2 - Sprint 2**

* **Goal: Integrate the three programs, launch attack via ettercap, adapted monitor.c to function with ettercap output.**
* **Deadline: 11/1/23**
* **Completion Date: 11/1/23**
* **Backlog: Test the program, settle on project name**

| **Task** | **Task Description** | **Team Member(s)** | **Estimated Hours** | **Actual Hours** |
| --- | --- | --- | --- | --- |
| T1 | Debug compatibility between monitor.c and UI | Nick M | 4 | 5 |
| T2 | Display log.txt within the UI | Nico L | 6 | 6 |
| T3 | Get monitor.c reading a stream of data from log.c | John B | 5 | 5 |
| T4 | Add history tab to UI | Nick M | 4 | 4 |
| T5 | Show log.txt in history tab | Nico L | 4 | 4 |
| T6 | Edit monitor.c to create output file of important events and timestamps | John B | 5 | 5 |

Table 9.2.2: Sprint 2 Backlog

**9.2.3 - Sprint 3**

* **Goal: Final testing - ensure our project is able to accurately detect and catch ARP Poisoning Attacks**
* **Deadline: 11/29/23**
* **Completion Date: 11/29/23**
* **Backlog: N/A (Last Sprint)**

| **Task** | **Task Description** | **Team Member(s)** | **Estimated Hours** | **Actual Hours** |
| --- | --- | --- | --- | --- |
| T1 | Set up demo environment, modify project | Nick M | 5 | 5 |
| T2 | Run ettercap attack with programs to test if they catch the attack | John B | 6 | 6 |
| T3 | Add color blind setting | Nick M | 5 | 5 |
| T4 | Setup up attack environment on everyone’s machines | Nico L | 6 | 6 |
| T5 | Add debug menu setting | Nick M | 5 | 5 |
| T6 | Run tests (monitor, log, UI) | All | 13 | 12 |
| T6 | Add directory selection setting | Nick M | 1 | 1 |

Table 9.2.3: Sprint 3 Backlog

**9.3 - Results**

**9.3.1 - Sprint 1 Results**

We successfully scrapped our old, outdated UI that we started working on last semester and replaced it with an open-source template that we found online that has a much more modern feel to it. Using Apache NetBeans, we made very few changes to this UI template, mostly focusing on the nodes tab.

**9.3.2 - Sprint 2 Results**

Sprint 2 was our group implementation phase. John and Nick worked together to implement monitor.c with the UI, while Nico and Nick collaborated to create a log tab within the UI. We also made some extra modifications to the UI during Sprint 2 to further differentiate the tabs from the template, such as adding a history tab.

**9.3.3 - Sprint 3 Results**

The core of our software was complete by the end of Sprint 2, so Sprint 3 was focused on ensuring all the team member’s virtual environments were set up optimally, as well as testing & adding other cosmetic features, such as a colorblind mode.

**9.4 - Testing Results**

We did regression testing while developing our software to ensure that our software didn’t lose functionality after integration.

| **Test** | **Test Case** | **Associated Sprint** | **Pass/Fail** | **Actions Taken if Failure** |
| --- | --- | --- | --- | --- |
| TC1 | Tested log.c by running it on different networks, both VMs & regular networks. | Sprint 1 | Pass |  |
| TC2 | Tested monitor.c by manually changing the entries in log.txt | Sprint 2 | Pass |  |
| TC3 | Tested monitor.c & log.c by launching an ARP attack via Ettercap | Sprint 3 | Pass |  |
| TC4 | Post-UI integration; tested monitor.c & log.c by manually changing the entries in log.txt | Sprint 3 | Pass, attack reflected within UI as well. |  |
| TC5 | Post-UI integration; tested monitor.c & log.c by launching an ARP attack via Ettercap | Sprint 3 | Pass, attack reflected within UI as well. |  |

Table 9.4: Test Case Results

**Chapter 10: Changes**

In our original project design, we had imagined a login screen for our user interface that would prompt a user for credentials before allowing them to run the program. The purpose of having a login screen was intended to increase program security and keep an unauthorized user from accessing network information. It was decided that this function would not be necessary, as it is implied that only an authorized user would have access to the program in the first place.

**10.1 - Project Goal Changes**

The goal of our project is to develop a method of catching an ARP poisoning attack, so a logon screen for our UI is an additional feature that does not contribute to the functionality of our program overall. Removing this function from our program did not affect our goal.

**10.2 - Design Changes**

Our original design featured a login screen that would launch first upon running the program. After supplying proper credentials, a user can then access the main interface of the program. We cut the login out of the design, so upon running the program, a user is brought directly to the main page of the user interface.

This change was made during the first sprint, when we were designing the user interface. We first started with the main page of the UI, and then debated as to the necessity of the pages that were based off that page. The login screen seemed like it was clunky in our design, and we therefore decided to cut it out.

**10.3 - Sprint Changes**

Our original sprint plans are quite different from how they actually turned out. In the original first sprint, our goal was to improve a hand-made user interface. However, we ended up using a template UI instead, so the goal shifted to working on features to make the template fit the project’s requirements. The other goal for the sprint, developing log.c and monitor.c to a runnable state, remained the same, however.

The second sprint remained largely unchanged from the planning phase to the actual execution. However, the final sprint, which originally was simply testing and optimization, was expanded to also include the addition of settings to the User Interface, and further development of monitor.c to include a more detailed, concise output to be displayed to the user.

**Chapter 11: Conclusions**

During the course of developing this project, the team was constantly learning and adapting to unexpected issues and making modifications to our design. The project started off as ideas on paper, but with the hard work of the team and the guidance of our advisor, it became a successful program that can be very useful to many network administrators. We were able to catch an ARP poisoning attack and block the attacker using the methods that have been discussed. The members of our team developed skills that are important for anybody working on a software related project. We also ran into many of the same software development problems that many run into such as version control, communication, and environmental controls (related to the Virtual Machines). We had to learn how to adapt to issues as they arose and how to work efficiently together to stay on schedule and keep making progress. We avoided communication problems by maintaining a chat on Discord that allowed us to stay up to date and keep members informed if they missed a team meeting. Overall, this project was a success and each member learned a lot about working on a team and project.

**References**

[1] J. Smith. “How to detect ARP poisoning attacks.” Cybersecurity Blog. <https://www.cybersecurityblog.com/arp-poisoning/>

[2] Oracle. “Creating a GUI With JFC/Swing.” Oracle Java Tutorials. <https://docs.oracle.com/javase/tutorial/uiswing/>

[3] The Open Group Base Specifications Issue 6. “netinet/in.h - Internet address family.” The Open Group Library. <https://pubs.opengroup.org/onlinepubs/7908799/xns/netinetin.h.html>

[4] Tcpdump/Libpcap Public Repository. “Documentation.” Tcpdump.org. <https://www.tcpdump.org/index.html#documentation>

[5] Ettercap Project Team. “Ettercap.” Ettercap-project.org. <https://www.ettercap-project.org/>

[6] O. Nelson. “How to Use Ettercap to Intercept Passwords with ARP Spoofing.” Null Byte. <https://null-byte.wonderhowto.com/how-to/use-ettercap-intercept-passwords-with-arp-spoofing-0191191/>

[7] Oracle Corporation. “VirtualBox.” Virtualbox.org. <https://www.virtualbox.org/>

[8] Rapid7 LLC. “Metasploit.” Metasploit.com. <https://www.metasploit.com/>

[9] Offensive Security Ltd. “Kali Linux.” Kali.org. <https://www.kali.org/>

[10] Wireshark Foundation. “Wireshark.” Wireshark.org. <https://www.wireshark.org/>

[11] Josh Lake, “ARP poisoning/spoofing How to detect and prevent” Comparitech. March 18, 2021. <https://www.comparitech.com/blog/information-security/arp-poisoning-spoofing-detect-prevent/>

[12] Robert Grimmic, “ARP Poisoning: What it is and how to prevent ARP Spoofing Attacks” Varonis, August 4, 2022 <https://www.varonis.com/blog/arp-poisoning#:~:text=Physical%20Security,a%20machine%20on%20the%20network>.

[13] Tomasz Andrzej Nidecki, “All About Man In The Middle Attacks”, Acunetix, March 13 2019 <https://www.acunetix.com/blog/articles/man-in-the-middle-attacks/>

[14] Elizabeth Montabano, “Ultimate-MITM Attack Steals 1M From Israeli Startup”, ThreatPost, December 5th, 2019 <https://threatpost.com/ultimate-mitm-attack-steals-1m-from-israeli-startup/150840/>

[15] IEEE Standard 802.2: TCP/IP Communications

[16] National Institute of Standards and Technology, “Advanced Encryption Standard (AES)”, May 9, 2023. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>

[17] The Tcpdump Group, “PCAP-FILTER(7) MAN PAGE”, June 13, 2023 <https://www.tcpdump.org/manpages/pcap-filter.7.html>

[18] Herve Eychenne, “iptables(8) - Linux man page”, <https://linux.die.net/man/8/iptables>

[19] The Tcpdump Group, “PCAP(3PCAP) MAN PAGE”, September 9, 2020, <https://www.tcpdump.org/manpages/pcap.3pcap.html>

[20] Leo Stratus, “netinet/ip.h”, February 5, 2013, <https://github.com/leostratus/netinet/blob/master/ip.h>

[21] Leo Stratus, “netinet/udp.h”, February 5, 2013, <https://github.com/leostratus/netinet/blob/master/udp.h>

[22] Leo Stratus, “netinet/tcp.h”, February 5, 2013, <https://github.com/leostratus/netinet/blob/master/tcp.h>

[23] Leo Stratus, “netinet/if\_ether.h”, February 5, 2013, <https://github.com/leostratus/netinet/blob/master/if_ether.h>

[24] Apache, “Designing a Swing GUI in Netbeans IDE” <https://netbeans.apache.org/tutorial/main/kb/docs/java/quickstart-gui/>

[25] Oracle, “Package javax.swing”, <https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/javax/swing/package-summary.html>

[26] Ettercap Project, “Ettercap”, October 27, 2023, <https://github.com/Ettercap/ettercap>

[27] tddmonkey, answer to “Multiline text in JLabel” on StackOverflow, March 26, 2009, <https://stackoverflow.com/questions/685521/multiline-text-in-jlabel/685631#685631>

[28] Amos Chepchieng, "Java-Swing-Custom-Titlebar", October 31, 2019, <https://github.com/k33ptoo/Java-Swing-Custom-Titlebar>