TERMINAL USING PROTECTION SECURITY MANAGEMNENT

MINI PROJECT REPORT

Submitted by

GOKULANAND.P(202209013)

ARUNKUMAR.K(202209006)

KANAGA BALA.R(202209023)

in

19AD481 –OPERATING SYSTEMS PRINCIPLE

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

MAY 2024

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **"GOKULANAND P(202209013),ARUNKUMAR.K(202209006),KANAGABALA.R(202209023) "** for the mini project titled **"TERMINAL PROTECTION SECURITY MANAGEMENT"** in 19AD481 – Operating System Principles during the fourth semester January 2024 – May 2024 under my supervision.

SIGNATURE

Mrs.p.Swathika,
Assistsnt Professor(Sr.G.),
AI&DS Department,
Mepco Schlenk Engg., College,
Sivakasi - 626005.

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Professor & Head,
AI&DS Department
Mepco Schlenk Engg., College,
Sivakasi - 626005

ABSTRACT

The rapid advancement and widespread adoption of terminal devices in various industries necessitate robust protection and security measures to safeguard sensitive data and ensure operational integrity. This paper explores the critical aspects of terminal security, focusing on both hardware and software components. It delves into the challenges posed by potential threats, including physical tampering, unauthorized access, malware, and network-based attacks. The study highlights current best practices in terminal protection, such as encryption, authentication mechanisms, secure boot processes, and intrusion detection systems. Additionally, it examines the role of regulatory frameworks and standards in shaping security protocols. By analyzing case studies and emerging technologies, this paper aims to provide a comprehensive overview of effective strategies for enhancing terminal security, thereby contributing to the development of more resilient and secure terminal infrastructures across various sectors.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

1.INTRODUCTION:

The program is a terminal-based application designed to provide users with a suite of command-line tools for managing files, directories, and user accounts. It simulates a simplified operating system environment where users can perform tasks such as reading, writing, and editing files, as well as executing utility commands like calculating the greatest common divisor or converting temperatures. This application supports user management, enabling the creation of new user accounts, secure login with password protection, and user-specific directories. It offers a personalized experience by maintaining separate home directories for each user, ensuring that each user's files are organized and accessible only to them.

Upon launching the terminal application, users are greeted with a welcome message and the current date and time, establishing an engaging start screen. The application then enters an interactive loop, continuously prompting users for commands. A comprehensive help command is available to guide users through the various functionalities, making the application user-friendly and accessible even to those less familiar with command-line interfaces.

1 .1 PROJECT DESCRIBTION:

In today's digital age, ensuring the security of terminals (computers, servers, etc.) is paramount. terminals serve as the primary means of accessing and interacting with digital information. Whether it's personal computers, enterprise servers, or mobile devices, terminals play a crucial role in our daily lives and business operations. However, this reliance on terminals also exposes us to a myriad of cyber threats that seek to exploit vulnerabilities in these systems.Cyber attacks have become increasingly sophisticated, ranging from malware infections and phishing scams to ransomware attacks and advanced persistent threats. As such, protecting terminals against these threats is no longer an option but a necessity. Terminal protection and security management aim to safeguard these critical assets and the data they hold from unauthorized access, manipulation, and theft.

1.2 OBJECTIVES:

1.2.1 Enhanced Security:

Enhancing terminal security involves implementing a multi-layered approach that addresses various attack vectors. This includes securing network connections, hardening system configurations, and deploying robust security solutions such as encryption and intrusion detection systems.

1.2.2 Risk Mitigation:

Identifying and mitigating potential security risks is essential to prevent security breaches and data leaks. This involves conducting regular risk assessments, vulnerability scans, and penetration testing to identify weaknesses in terminal systems and infrastructure.

1.2.3 Compliance:

Adhering to industry standards and regulatory requirements is crucial for ensuring the security and privacy of sensitive information. Compliance with standards such as PCI DSS, HIPAA, GDPR, and ISO 27001 helps organizations demonstrate their commitment to protecting customer data and maintaining trust.

1.2.4 Efficiency:

Efficient security management is about balancing security controls with usability and performance. This involves implementing automated security processes, optimizing security configurations, and minimizing the impact on system performance and user productivity.

1.2.5 Monitoring and Response:

Real-time monitoring of terminal activity and security events is essential for detecting and responding to security incidents promptly. Implementing security information and event management (SIEM) solutions enables organizations to aggregate, correlate, and analyze security data from various sources to identify potential threats and take appropriate action.

1.3 SCOPE OF PROJECT:

1.3.1 Terminal Types:

The project will cover a wide range of terminals, including desktop computers, laptops, servers, mobile devices (smartphones, tablets), and embedded systems (ATMs, POS terminals).

1.3.2 Security Measures:

Security measures to be implemented include access control mechanisms (authentication, authorization), encryption (data encryption at rest and in transit), endpoint protection (antivirus, antimalware), network security (firewalls, intrusion detection/prevention), and security awareness training for users.

1.3.3 User Training:

Educating users about security best practices is essential for creating a security-conscious culture within an organization. Training programs will cover topics such as password hygiene, phishing awareness, device security, and incident reporting procedures.

1.3.4 Integration:

Integrating security solutions seamlessly into existing terminal infrastructure requires careful planning and coordination. This includes compatibility testing, configuration management, and deployment strategies to minimize disruption to normal business operations.

1.3.5 Continuous Improvement:

Security is an ongoing process that requires regular assessment, adaptation, and improvement. Establishing procedures for continuous monitoring, vulnerability management, and incident response ensures that security measures remain effective against evolving threats.

1.4 PROPOSED SYSTEM:

1.4.1 Access Control:

Implementing role-based access control (RBAC) ensures that users have the appropriate permissions to access resources based on their roles and responsibilities within the organization. This reduces the risk of unauthorized access and privilege escalation.

1.4.2 Encryption:

Utilizing strong encryption algorithms (e.g., AES, RSA) protects sensitive data from unauthorized access, interception, and tampering. Encryption techniques such as full-disk encryption, file-level encryption, and SSL/TLS encryption for network communications ensure data confidentiality and integrity.

1.4.3 Antivirus and Anti-malware:

Deploying reputable antivirus and antimalware software helps detect and remove malicious software threats such as viruses, worms, Trojans, and spyware. Regular virus definition updates and scheduled scans ensure that terminal systems remain protected against the latest threats.

1.4.4 Firewalls:

Configuring firewalls to filter incoming and outgoing network traffic based on predefined security rules helps prevent unauthorized access to terminal systems. Firewalls can be implemented at the network perimeter, on individual terminals, and within applications to enforce access control policies and block malicious traffic.

1.4.5 Intrusion Detection and Prevention:

Deploying intrusion detection and prevention systems (IDPS) helps monitor terminal activity for signs of suspicious behavior or security breaches. IDPS solutions can detect and alert on unauthorized access attempts, malware infections, and anomalous network traffic, allowing organizations to respond proactively to potential threats.

1.4.6 Security Auditing:

Conducting regular security audits and assessments helps identify vulnerabilities, misconfigurations, and compliance gaps in terminal systems and infrastructure. Auditing tools and techniques such as vulnerability scanning, penetration testing, and log analysis provide insights into security posture and areas for improvement.

1.5 SYSTEM DESIGN:

1.5.1 Block Diagram:

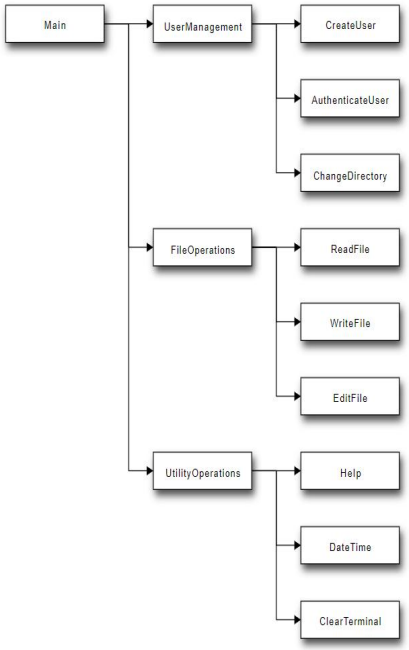

Figure1.5.1: block diagram

1.5.2 Architecture:

Designing a resilient and scalable architecture involves selecting appropriate hardware and software components that meet the security requirements of terminal systems. This includes choosing secure operating systems, hardware security modules, and network appliances that support encryption, access control, and monitoring capabilities.

1.5.3 Components:

Identifying and integrating security components such as authentication servers, encryption libraries, intrusion detection sensors, and centralized management consoles into the terminal infrastructure ensures comprehensive protection against security threats. Choosing interoperable and vendor-agnostic solutions facilitates seamless integration and interoperability.

1.5.4 Integration:

Integrating security components into the terminal infrastructure requires careful planning and coordination to minimize disruption to normal operations. This involves testing compatibility, configuring settings, and deploying updates in a controlled manner to ensure that security measures are effective and transparent to end-users.

1.5.5 User Interface:

Developing user-friendly interfaces for security management consoles enables administrators to configure, monitor, and troubleshoot security controls efficiently. Intuitive dashboards, alerts, and reporting tools provide visibility into security events and facilitate decision-making to address security incidents promptly.

1.5.5 Monitoring and Reporting:

Implementing monitoring and reporting capabilities enables organizations to track security events in real-time, analyze trends, and generate compliance reports for audit purposes. Integration with SIEM platforms, ticketing systems, and incident response workflows streamlines security operations and facilitates collaboration across teams.

CHAPTER-2

2.1 IMPLEMENTATION

User Structure and Definitions:

Define the User structure to store user information, including username, password, and home_dir.Define constants for maximum username and password lengths.

Utility Functions:

- **hidePassword:**This function hides the password input in the terminal for security purposes using terminal attributes.
- **dt:** Displays the current date and time using the time library.

User Management Functions:

- **create_user:**Checks if the user already exists by attempting to open the corresponding file.Prompts for a password, hides the input, and creates a directory for the user.Saves the credentials to a file.
- **authenticate_user:**Prompts for the password and verifies it against the saved credentials.
- **save_credentials**: Writes the username and password to a file.
- **load_credentials:** Loads the user credentials from a file.
- **logout:** Clears the user data and changes the directory back to a default path.
- **change_user_dir:** Changes the current working directory to the user's home directory.

Main Function and Command Loop:

- The main function initializes the User structure and runs an infinite loop to prompt for user commands.
- Commands are read using scanf and processed in an if-else structure to determine the action to take.
- User-related commands (user, login, logout) are handled separately.
- Commands related to file operations and utilities are executed only if the user is logged in.

File Operations:

- **read_file:** Reads and displays the contents of a specified file.
- **write_file:** Writes user input to a specified file until a termination character (~) is encountered.
- **append:** Appends user input to a specified file until a termination character (~) is encountered.
- **removes:** Deletes a specified file.
- **renames:** Renames a specified file.
- **edit:** Opens a file for editing, similar to writing but allows for modifications.

Utility Commands:

- **clear:** Clears the terminal screen.
- **help:** Displays a list of supported commands.
- **cdir:** Changes the current working directory.
- **copy:** Copies the contents of one file to another.
- **cknow:** Displays information about the C programming language.
- **lknow:** Displays information about Linux.
- **ccount:** Counts the number of characters in a specified file.
- **gcd:** Computes the greatest common divisor of two integers.
- **lcm:** Computes the least common multiple of two integers.
- **tconv:** Converts a temperature between Celsius and Fahrenheit.

2.2 Program Coding:

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>
```

```
#include <math.h>

#include <time.h>

#include <dirent.h>

#include <sys/stat.h>

#include <pwd.h>

#include <termios.h>



#define MAX_USERNAME_LENGTH 50

#define MAX_PASSWORD_LENGTH 50



typedef struct {

    char username[MAX_USERNAME_LENGTH];

    char password[MAX_PASSWORD_LENGTH];

    char home_dir[100];

} User;



void read_file();

void write_file();

void append();
```

```
void clear();

void help();

void dt();

void startscreen();

void removes();

void renames();

void edit();

void cdir();

void copy();

void cknow();

void lknow();

void ccount();

void gcd();

void lcm();

void tconv();

void hidePassword(char *password);

int create_user(User *user, const char *username);

int change_user_dir(const User *user);

int authenticate_user(User *user, const char *username);
```

```
void save_credentials(const char *username, const char *password);
int load_credentials(User *user, const char *username);
void logout(User *user);


void hidePassword(char *password) {
    struct termios oldt, newt;
    printf("\033[8m"); // Hide terminal input
    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~ECHO;
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    scanf("%s", password);
    printf("\033[28m"); // Unhide terminal input
    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    printf("\n");
}


int create_user(User *user, const char *username) {
    char filename[MAX_USERNAME_LENGTH + 5];
```

```
snprintf(filename, sizeof(filename), "%s.txt", username);



FILE *file = fopen(filename, "r");

if (file) {

    fclose(file);

    printf("User already exists.\n");

    return 0;

}



char password[MAX_PASSWORD_LENGTH];

printf("Enter password: ");

hidePassword(password);



int result = mkdir(username, 0777);

if (result != 0) {

    perror("mkdir");

    return 0;

}
```

```
    snprintf(user->username, sizeof(user->username), "%s", username);
    snprintf(user->password, sizeof(user->password), "%s", password);
    snprintf(user->home_dir, sizeof(user->home_dir), "%s/%s", ".", username);

    save_credentials(username, password);

    return 1;
}

int change_user_dir(const User *user) {
    return chdir(user->home_dir);
}

int authenticate_user(User *user, const char *username) {
    char password[MAX_PASSWORD_LENGTH];
    printf("Enter password: ");
    hidePassword(password);

    if (load_credentials(user, username) && strcmp(user->password, password) == 0) {
```

```
            return 1;
        } else {
            printf("Invalid credentials.\n");
            return 0;
        }
    }

void save_credentials(const char *username, const char *password) {
    char filename[MAX_USERNAME_LENGTH + 5];
    snprintf(filename, sizeof(filename), "%s.txt", username);

    FILE *file = fopen(filename, "w");
    if (file) {
        fprintf(file, "%s %s\n", username, password);
        fclose(file);
    } else {
        perror("Error opening credentials file");
    }
}
```

```
int load_credentials(User *user, const char *username) {
    char filename[MAX_USERNAME_LENGTH + 5];
    snprintf(filename, sizeof(filename), "%s.txt", username);

    FILE *file = fopen(filename, "r");
    if (file) {
        char uname[MAX_USERNAME_LENGTH];
        char pword[MAX_PASSWORD_LENGTH];
        if (fscanf(file, "%s %s", uname, pword) == 2) {
            if (strcmp(uname, username) == 0) {
                snprintf(user->username, sizeof(user->username), "%s", uname);
                snprintf(user->password, sizeof(user->password), "%s", pword);
                snprintf(user->home_dir, sizeof(user->home_dir), "%s/%s", ".", uname);
                fclose(file);
                return 1;
            }
        }
        fclose(file);
```

```
    }
    return 0;
}

void logout(User *user) {
    memset(user, 0, sizeof(User));
    printf("Logged out successfully.\n");
    chdir("/run/media/root/Data/student/ad/ug/be2022/22bad037/sem4/os");
}

int main() {
    User user;
    char a[100];
    int logged_in = 0;

    printf("Terminal\n\n");

    dt();
    printf("Welcome To Terminal\n");
```

```
printf("Type \"help\" for more things!\n\n");


while (1) {
    printf("---> ");
    scanf("%s", a);
    if (strcmp(a, "user") == 0) {
        char un[MAX_USERNAME_LENGTH];
        printf("Enter username: ");
        scanf("%s", un);
        if (create_user(&user, un)) {
            printf("User created successfully.\n");
        }
    } else if (strcmp(a, "login") == 0) {
        char un[MAX_USERNAME_LENGTH];
        printf("Enter username: ");
        scanf("%s", un);
        if (authenticate_user(&user, un)) {
            change_user_dir(&user);
            logged_in = 1;
```

```
            printf("Login successful.\n");
        }
    } else if (strcmp(a, "logout") == 0) {
        if (logged_in) {
            logout(&user);
            logged_in = 0;
        } else {
            printf("No user is currently logged in.\n");
        }
    } else {
        if (!logged_in) {
            printf("Please login to use the terminal commands.\n");
            continue;
        }

        if (strcmp(a, "read") == 0) {
            read_file();
        } else if (strcmp(a, "write") == 0) {
            write_file();
```

```
} else if (strcmp(a, "append") == 0) {
    append();
} else if (strcmp(a, "clr") == 0) {
    clear();
} else if (strcmp(a, "help") == 0) {
    help();
} else if (strcmp(a, "dt") == 0) {
    dt();
} else if (strcmp(a, "stscr") == 0) {
    startscreen();
} else if (strcmp(a, "remove") == 0) {
    removes();
} else if (strcmp(a, "rename") == 0) {
    renames();
} else if (strcmp(a, "edit") == 0) {
    edit();
} else if (strcmp(a, "cdir") == 0) {
    cdir();
} else if (strcmp(a, "copy") == 0) {
```

```
        copy();
    } else if (strcmp(a, "cknow") == 0) {
        cknow();
    } else if (strcmp(a, "lknow") == 0) {
        lknow();
    } else if (strcmp(a, "ccount") == 0) {
        ccount();
    } else if (strcmp(a, "gcd") == 0) {
        gcd();
    } else if (strcmp(a, "lcm") == 0) {
        lcm();
    } else if (strcmp(a, "tconv") == 0) {
        tconv();
    } else if (strcmp(a, "exit") == 0) {
        break;
    } else {
        printf("Enter only stated things in help\n");
    }
}
```

```
    }


    return 0;
}


void read_file() {
    char a[100];
    printf("Use .txt or any other extension to read\nEx: hello.txt\n\n");
    FILE *p;
    printf("Enter File Name To Read: ");
    scanf("%s", a);

printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>");


    char c;


    p = fopen(a, "r");


    if (p == NULL) {
```

```
        printf("\nError\n");

        return;

    }



    printf("\n");



    c = fgetc(p);



    while (c != EOF) {

        printf("%c", c);

        c = fgetc(p);

    }




printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>");

    fclose(p);

    printf("\n\n Successfully Read\n");

}
```

```
void write_file() {
    char a[100];
    printf("Use .txt or any other extension to write\nEx: hello.txt\n\n");
    FILE *p;
    printf("Enter File Name To Write: ");
    scanf("%s", a);
    p = fopen(a, "w");
    fflush(stdin);
    printf("Enter ~ to exit from writing\n");
    printf("Start writing: \n");
    int ch = NULL;
    while (ch != '~') {
        ch = getchar();
        if (ch != '~') {
            fputc(ch, p);
        }
    }
    fclose(p);
    printf("\n\nSuccessfully Written\n");
```

```
}

void append() {
    char a[100];
    printf("Use .txt or any other extension to append\nEx: hello.txt\n\n");
    FILE *p;
    printf("Enter File Name To Append: ");
    scanf("%s", a);

    p = fopen(a, "a");
    if (p == NULL) {
        printf("Error\n");
        return;
    }

    printf("Enter ~ to exit from appending\n");
    printf("Start appending: \n");

    int ch = NULL;
```

```
    while (ch != '~') {
        ch = getchar();
        if (ch != '~') {
            fputc(ch, p);
        }
    }

    fclose(p);
    printf("\n\nSuccessfully Appended\n");
}

void clear() {
    system("clear");
    printf("Terminal Cleared\n");
}

void help() {
    printf("Supported commands:\n");
    printf("  read      - Read a file\n");
```

```
printf("  write     - Write to a file\n");

printf("  append    - Append to a file\n");

printf("  clr       - Clear the terminal\n");

printf("  dt        - Display the current date and time\n");

printf("  stscr     - Display the start screen\n");

printf("  remove    - Remove a file\n");

printf("  rename    - Rename a file\n");

printf("  edit      - Edit a file\n");

printf("  cdir      - Change directory\n");

printf("  copy      - Copy a file\n");

printf("  cknow     - Know the C language\n");

printf("  lknow     - Know about Linux\n");

printf("  ccount    - Count characters in a file\n");

printf("  gcd       - Calculate the greatest common divisor\n");

printf("  lcm       - Calculate the least common multiple\n");

printf("  tconv     - Convert temperature\n");

printf("  user      - Create a new user\n");

printf("  login     - Login as an existing user\n");

printf("  logout    - Logout the current user\n");
```

```
    printf("  exit      - Exit the terminal\n");
}

void dt() {
    time_t t;
    time(&t);
    printf("Date and Time: %s\n", ctime(&t));
}

void startscreen() {

printf("\n\n<<<<<<<<<<<<<<<<<<<<<<<<<START>>>>>>>>>>>>>>>>>>>>>>>>>
>>\n\n");
    dt();
    printf("WELCOME TO THE TERMINAL\n");
    printf("Type 'help' to see the list of commands.\n\n");
}

void removes() {
    char a[100];
```

```
    printf("Enter File Name to Delete: ");
    scanf("%s", a);
    if (remove(a) == 0) {
        printf("File deleted successfully.\n");
    } else {
        perror("Error deleting file");
    }
}

void renames() {
    char oldname[100], newname[100];
    printf("Enter Existing File Name: ");
    scanf("%s", oldname);
    printf("Enter New File Name: ");
    scanf("%s", newname);
    if (rename(oldname, newname) == 0) {
        printf("File renamed successfully.\n");
    } else {
        perror("Error renaming file");
```

```
    }
}

void edit() {
    char a[100];
    printf("Use .txt or any other extension to edit\nEx: hello.txt\n\n");
    FILE *p;
    printf("Enter File Name to Edit: ");
    scanf("%s", a);

    p = fopen(a, "r+");
    if (p == NULL) {
        printf("Error\n");
        return;
    }

    printf("Enter ~ to exit from editing\n");
    printf("Start editing: \n");
```

```
    int ch = NULL;
    while (ch != '~') {
        ch = getchar();
        if (ch != '~') {
            fputc(ch, p);
        }
    }

    fclose(p);
    printf("\n\nSuccessfully Edited\n");
}

void cdir() {
    char dir[100];
    printf("Enter Directory Name: ");
    scanf("%s", dir);
    if (chdir(dir) == 0) {
        printf("Directory changed successfully.\n");
    } else {
```

```
        perror("Error changing directory");
    }
}

void copy() {
    char source[100], destination[100];
    printf("Enter Source File: ");
    scanf("%s", source);
    printf("Enter Destination File: ");
    scanf("%s", destination);

    FILE *src = fopen(source, "r");
    if (src == NULL) {
        perror("Error opening source file");
        return;
    }

    FILE *dest = fopen(destination, "w");
    if (dest == NULL) {
```

```
        perror("Error opening destination file");
        fclose(src);
        return;
    }

    char c;
    while ((c = fgetc(src)) != EOF) {
        fputc(c, dest);
    }

    fclose(src);
    fclose(dest);
    printf("File copied successfully.\n");
}

void cknow() {
    printf("C is a general-purpose, procedural computer programming language supporting structured programming.\n");
}
```

```
void lknow() {
   printf("Linux is a family of open-source Unix-like operating systems based on the Linux kernel.\n");
}

void ccount() {
   char a[100];
   printf("Use .txt or any other extension to count characters\nEx: hello.txt\n\n");
   FILE *p;
   printf("Enter File Name to Count Characters: ");
   scanf("%s", a);

   p = fopen(a, "r");
   if (p == NULL) {
      printf("Error\n");
      return;
   }

   int count = 0;
   char c;
```

```
    while ((c = fgetc(p)) != EOF) {
        count++;
    }

    fclose(p);
    printf("Total characters: %d\n", count);
}

void gcd() {
    int a, b, t;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    while (b != 0) {
        t = b;
        b = a % b;
        a = t;
    }
```

```
    printf("Greatest common divisor: %d\n", a);
}

void lcm() {
    int a, b;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    int gcd, lcm;
    int t = a, q = b;

    while (b != 0) {
        int r = a % b;
        a = b;
        b = r;
    }
    gcd = a;
    lcm = (t * q) / gcd;
```

```
    printf("Least common multiple: %d\n", lcm);
}

void tconv() {
    double temp;
    char unit;
    printf("Enter temperature (e.g., 36.5C or 97.7F): ");
    scanf("%lf%c", &temp, &unit);

    if (unit == 'C' || unit == 'c') {
        printf("Fahrenheit: %.2lfF\n", (temp * 9 / 5) + 32);
    } else if (unit == 'F' || unit == 'f') {
        printf("Celsius: %.2lfC\n", (temp - 32) * 5 / 9);
    } else {
        printf("Invalid unit.\n");
    }
}
```

2.3 OUTPUT:

```
[22bad037@mepcolinux os]$clear
[22bad037@mepcolinux os]$./a.out
Terminal

Date and Time: Wed May 22 21:32:14 2024

Welcome To Terminal
Type "help" for more things!

---> help
Please login to use the terminal commands.
---> login
Enter username: arun
Enter password:
Login successful.
---> write
Use .txt or any other extension to write
Ex: hello.txt

Enter File Name To Write: ff.c
Enter ~ to exit from writing
Start writing:
#include<stdio.h>
int main()
{
        print("Welcome to the terminal ");
        return 0;
}~


Successfully Written
```

Figure1.1: user login and writing files

```
---> read
Use .txt or any other extension to read
Ex: hello.txt

Enter File Name To Read: ff.c
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

#include<stdio.h>
int main()
{
        print("Welcome to the terminal ");
        return 0;
}>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

 Successfully Read
---> help
Supported commands:
  read      - Read a file
  write     - Write to a file
  append    - Append to a file
  clr       - Clear the terminal
  dt        - Display the current date and time
  stscr     - Display the start screen
  remove    - Remove a file
  rename    - Rename a file
  edit      - Edit a file
  cdir      - Change directory
  copy      - Copy a file
  cknow     - Know the C language
  lknow     - Know about Linux
  ccount    - Count characters in a file
  gcd       - Calculate the greatest common divisor
  lcm       - Calculate the least common multiple
  tconv     - Convert temperature
  user      - Create a new user
  login     - Login as an existing user
  logout    - Logout the current user
  exit      - Exit the terminal
---> stscr
```

Figure1.2: read files and view commands

```
<<<<<<<<<<<<<<<<<<<<<<<<START>>>>>>>>>>>>>>>>>>>>>>>>

Date and Time: Wed May 22 21:34:42 2024

WELCOME TO THE TERMINAL
Type 'help' to see the list of commands.

---> cknow
C is a general-purpose, procedural computer programming language supporting structured programming.
---> lknow
Linux is a family of open-source Unix-like operating systems based on the Linux kernel.
---> gcd
Enter two integers: 10  5
Greatest common divisor: 5
---> user kumar
Enter username: Enter password:
User created successfully.
---> login
Enter username: kumar
Enter password:
Login successful.
---> logout
Logged out successfully.
---> lcm
Please login to use the terminal commands.
---> login
Enter username: arun
Enter password:
Login successful.
---> lcm
Enter two integers: 40  5
Least common multiple: 40
--->
```

Figure1.3: creating user and do mathetical operations

CHAPTER-3

CONCLUSION

The increasing reliance on terminal devices across numerous industries underscores the critical need for comprehensive protection and security measures. This paper has examined various facets of terminal security, highlighting the importance of a multi-layered approach that encompasses both hardware and software defenses. Key strategies such as encryption, robust authentication mechanisms, secure boot processes, and advanced intrusion detection systems have been identified as essential components of an effective security framework.The study also emphasized the significance of adhering to regulatory standards and industry best practices to ensure consistent and reliable security measures. Through the analysis of case studies, it became evident that organizations implementing a holistic security strategy are better equipped to mitigate risks associated with physical tampering, unauthorized access, malware, and network-based attacks.

REFERENCES

1. National Institute of Standards and Technology (NIST) - "Security Requirements for Cryptographic Modules" https://csrc.nist.gov/publications/detail/fips/140/2/final

2. International Organization for Standardization (ISO) - "ISO/IEC 27001:2013 Information Security Management" https://www.iso.org/isoiec-27001-information-security.html

3. European Union Agency for Cybersecurity (ENISA) - "Guidelines on Network and Information Security" https://www.enisa.europa.eu/topics/iot-and-smart-infrastructures/iot/guidelines

4. SANS Institute - "Critical Security Controls" https://www.sans.org/critical-security-controls/

5. Cybersecurity & Infrastructure Security Agency (CISA) - "Cybersecurity Best Practices" https://www.cisa.gov/cybersecurity-best-practices

6. Center for Internet Security (CIS) - "CIS Controls and Benchmarks" https://www.cisecurity.org/controls/