System Security Lab - Comprehensive Notes

1. Cryptography Algorithms

1.1 Caesar Cipher

Overview: A substitution cipher that shifts each character by a fixed key value.

Key Concepts:

- **Encryption:** Shifts each letter forward by the key amount (wraps around Z to A)
- **Decryption:** Shifts each letter backward by the key amount (wraps around A to Z)
- **Key range:** 0-25 (modulo 26 alphabet positions)
- **Application:** Historical cipher, educational purposes only (not secure for real use)

Important Code Structure:
```
void encrypt(char message[], int key) {
for(i = 0; message[i] != '\0'; ++i) {
if(ch >= 'a' && ch <= 'z') {
ch = ch + key;
if(ch > 'z') ch = ch - 26;
message[i] = ch;
}
}
}
```

Command to Compile & Run:
```
gcc caesar.c -o caesar
./caesar
```

1.2 Hill Cipher

Overview: A polyalphabetic substitution cipher using matrix multiplication.

Key Concepts:

- **Matrix Operations:** Uses matrix multiplication modulo 26
- **Encryption Process:**
    1. Take key matrix and plaintext vector
    2. Multiply: Encrypted = (Key Matrix × Plaintext Vector) mod 26
    3. Apply modulo 26 to keep values in alphabet range (0-25)
- **Decryption:** Multiply with inverse of key matrix

Important Algorithm:
```
// Matrix multiplication: mul = a * b
for (i = 0; i < r; i++) {
```

```
for (j = 0; j < 1; j++) {
for (k = 0; k < r; k++) {
mul[i][j] += a[i][k] * b[k][j];
}
}
}
// Apply modulo 26 for encryption
enc[j] = mul[i][j] % 26;
```

Example Execution:
Input: 3x3 Key Matrix, plaintext vector
Output: Encrypted values mod 26

1.3 Rail Fence Cipher

Overview: A transposition cipher that arranges plaintext in a zigzag pattern across multiple rails.

Key Concepts:

- **Rails:** Message is written in zigzag pattern across N rails
- **Direction:** Alternates down and up for each row
- **Reading:** Read each rail sequentially to get ciphertext
- **Key:** Number of rails determines security level

Algorithm Logic:

```
// Fill matrix in zigzag pattern
while(j < len){
if (count%2==0) {
for(i=0; i<rails; i++) {
code[i][j]=(int)str[j];
j++;
}
} else {
for(i=rails-2; i>0; i--) {
code[i][j]=(int)str[j];
j++;
}
}
}
```

---

2. Cryptographic Protocol: Diffie-Hellman Key Exchange

Overview: Algorithm for secure exchange of cryptographic keys over public channels.

Key Parameters:

- **P (Prime):** Large prime number (public)
- **G (Generator):** Primitive root modulo P (public)
- **Private Keys:** a (Alice), b (Bob) - kept secret
- **Public Keys:** x = G^a mod P (Alice), y = G^b mod P (Bob)

- **Shared Secret:** ka = y^a mod P (Alice), kb = x^b mod P (Bob)

Why It Works:

- ka = (G^b mod P)^a mod P = G^(ab) mod P
- kb = (G^a mod P)^b mod P = G^(ab) mod P
- Both compute same shared secret: G^(ab) mod P

Implementation Function:

```
long long power(long long base, long long exp, long long mod) {
long long result = 1;
base = base % mod;
while (exp > 0) {
if (exp % 2 == 1)
result = (result * base) % mod;
exp = exp / 2;
base = (base * base) % mod;
}
return result;
}
```

Important Security Commands:

Compile Diffie-Hellman program

```
gcc diffie_hellman.c -o dh
```

Run

```
./dh
```

---

3. Cyclic Redundancy Check (CRC)

Overview: Error-detection algorithm to identify accidental changes in digital data.

Key Concepts:

- **Generator Polynomial:** Binary string determining CRC computation
- **XOR Operation:** Fundamental operation in CRC
- **Remainder:** CRC value appended to data for transmission
- **Receiver Verification:** If remainder is 0, no error detected

Important Algorithm Steps:

1. Pad data with (n-1) zeros where n = length of generator polynomial
2. Perform polynomial division using XOR
3. Get remainder as CRC value
4. Append CRC to original data
5. On receiver: Divide received data by generator polynomial
6. If remainder = 0, data is error-free

Critical Code Pattern:

```
void XOR() {
for (j = 1; j < N; j++) {
check_value[j] = (check_value[j] == gen_poly[j]) ? '0' : '1';
}
}

void crc() {
for (i = 0; i < N; i++) {
check_value[i] = data[i];
}
do {
if (check_value[0] == '1') XOR();
for (j = 0; j < N - 1; j++)
check_value[j] = check_value[j + 1];
check_value[j] = data[i++];
} while (i <= data_length + N - 1);
}
```

---

4. Linux File System & Security Commands

4.1 Directory & File Operations

Critical Commands:

Create directories with subdirectories

```
mkdir -p lab/file # -p flag creates parents
```

List with detailed information

```
ls -l # Show permissions, owner, size, date
ls -ld # Show directory itself, not contents
ls -ld <directory> # Detailed info for specific directory
```

Change directory

```
cd <path> # Change to directory
cd .. # Go to parent directory
cd - # Go to previous directory
pwd # Print current working directory
```

4.2 File Ownership & Permission Modification

Critical Commands:

Change ownership

chown <user>:<group> <file> # Change owner and group
chown -R <user>:<group> <directory> # Recursive change (-R flag)

Example:

sudo chown -R root:root lab # Change lab and all contents to root ownership
sudo chown -R asas:asas cyber # Change back to user asas

Verify ownership changes

ls -ld <file/directory> # Show ownership information

Permission Modes:
chmod 700 file.txt # Owner: read,write,execute; Others: none
chmod 777 directory # Everyone: read,write,execute
chmod o+r file.txt # Add read permission for others (o+r)
chmod u+w file.txt # Add write permission for user (u+w)

Permission Representation:

- **r (read):** 4
- **w (write):** 2
- **x (execute):** 1
- **Owner** | **Group** | **Others:** e.g., 755 = rwxr-xr-x

4.3 User & Account Management

Critical Commands:

Create new user

sudo useradd -m -s/bin/bash <username>

-m: Create home directory

-s: Specify shell (bash)

Set password

sudo passwd <username> # Set or change password

Password expiration policy

sudo chage -M 90 -m 10 <username>

-M 90: Password expires in 90 days

-m 10: Minimum 10 days before password change

---

5. Linux Firewall Configuration (UFW)

5.1 UFW Firewall Setup

Critical Commands:

Enable firewall

sudo ufw enable # Activate firewall at startup

Set default policies

sudo ufw default deny incoming # Deny all incoming by default
sudo ufw default allow outgoing # Allow outgoing (default)

Allow specific ports

sudo ufw allow 22/tcp # Allow SSH (port 22)
sudo ufw allow http # Allow HTTP (port 80)
sudo ufw allow https # Allow HTTPS (port 443)
sudo ufw allow <port>/<protocol> # Generic format

View firewall status

sudo ufw status verbose # Show detailed status with all rules
sudo ufw status # Show basic status

Disable firewall (if needed)

sudo ufw disable # Turn off firewall

UFW Status Output Explanation:
Status: active # Firewall is running
Logging: on (low) # Logging enabled
Default: deny (incoming), allow (outgoing)
New profiles: skip

Port Rules:
22/tcp ALLOW IN Anywhere # SSH allowed from anywhere
80/tcp ALLOW IN Anywhere # HTTP allowed from anywhere

---

6. Network Scanning & Analysis (Nmap)

6.1 Nmap Commands for Network Reconnaissance

Critical Commands:

Ping scan entire subnet

sudo nmap -sn 192.168.1.0/24

-sn: Ping scan only (no port scanning)

Finds all active hosts in range

Port scanning with service detection

sudo nmap -Pn -p 80,443 192.168.1.1

-Pn: Skip host discovery (assume host is up)

-p: Specify ports to scan

Multiple ports separated by comma

Service version detection

```
nmap -Pn -sV -p 80,443 192.168.1.1
```

-sV: Detect service versions

Common port states:

open: Service accepting connections

closed: Port accessible, no service listening

filtered: Firewall blocking/filtering

Port Scanning Interpretation:

```
80/tcp filtered http # Port blocked by firewall
443/tcp filtered https # Cannot confirm if open/closed
```

6.2 Packet Capture (tcpdump)

Critical Command:

Capture network packets

```
sudo tcpdump
```

Captures packets on default interface

Shows real-time network traffic

Options:

-i eth0: Capture on specific interface

-n: Don't resolve hostnames

-c 100: Capture 100 packets and exit

---

7. System Information & Monitoring Commands

Critical Commands:

System identification

hostname # Display system hostname
whoami # Current user
uname # Operating system name
logname # Logged-in username
uptime # System uptime, load average

Network information

ifconfig # Display network interfaces & IP
ip a # Modern interface configuration
ip route # Show routing table
hostname -I # Show IP address

Process monitoring (CRITICAL FOR LAB EXAMS)

ps -eo pid,%cpu,args | sort -k 2 -r | head -n 11

List top 10 processes by CPU usage

ps -eo pid,%mem,args | sort -k 2 -r | head -n 11

List top 10 processes by memory usage

User/Session information

who | wc -l # Count logged-in users
finger <username> # User details

Disk and resource monitoring

df -h # Disk space (human-readable)
df -h | awk '$NF=="/"{printf "%s",$5}' # Show root partition usage

Log examination

cat /var/log/kern.log # Kernel logs
grep "ERROR" /var/log/kern.log # Find errors in logs
find /var/log -name "*.log" # Find all log files

DNS configuration

grep "nameserver" /etc/resolv.conf | awk '{print $2}' # Show DNS server

---

8. Shell Scripting Fundamentals

8.1 Basic Shell Script Structure

#!/bin/bash # Shebang - specifies bash interpreter

8.2 Variable & Input Operations

Critical Commands:

Reading input

read variable # Read single input
read a b c # Read multiple inputs into variables
read -p "Prompt: " var # Read with prompt message
read -r input # Read with escape character handling

Using variables

```
echo a $b $c" # Multiple variables
```

8.3 Arithmetic Operations

Critical Operations:

Basic arithmetic

```
sum=expr $a + $b + $c # Addition (backticks)
sum=((i + 1)) # Add value
```

Supported operators:

+ (add), - (subtract), * (multiply), / (divide), % (modulo)

8.4 Conditional Statements

Critical Operators:

Numeric comparison

```
-eq # Equal to
-ne # Not equal to
-gt # Greater than
-lt # Less than
-ge # Greater than or equal to
-le # Less than or equal to
```

If-else syntax

```
if [ condition ]; then
echo "True"
else
echo "False"
fi
```

Example:

```
if [ $a -gt b]; thenecho"a is greater"
else
echo "$b is greater"
fi
```

8.5 Loop Structures (CRITICAL FOR LAB EXAMS)

While Loop:

```
i=1
while [ $i -le 5 ]; do
echo ii =((i + 1))
done
```

Until Loop:

```
j=1
until [ $j -gt 5 ]; do
echo jj =((j + 1))
done
```

For Loop (C-style):

```
for ((k=5; k<10; k++)) do
echo $k
done
```

For Loop (Iterative):

```
for i in 1 2 3 4; do
echo $i
done
```

8.6 Array & Pattern Generation Scripts

Important Patterns:

Pattern 1: Square (4x4):

```
for ((i=1; i<=4; i++))
do
for ((j=1; j<=4; j++))
do
echo -n "*"
done
echo # new line
done
```

Pattern 2: Right Triangle:

```
for i in 1 2 3 4; do
for ((j=1; j<=i; j++))
do
echo -n "* "
done
```

```
echo
done
```

Pattern 3: Diamond Triangle:

```
for i in 1 2 3 4; do
for ((j=1; j<=2i-1; j++))
do
echo -n " "
done
echo
done
```

Pattern 4: Reverse Triangle:

```
for i in 4 3 2 1; do
for ((j=1; j<=i; j++))
do
echo -n "* "
done
echo
done
```

Pattern 5: Number Triangle:

```
for i in 1 2 3 4; do
for ((j=1; j<=i; j++))
do
echo -n "$j "
done
echo
done
```

Pattern 6: Continuous Number:

```
num=1
for i in 1 2 3 4; do
for ((j=1; j<=i; j++))
do
echo -n "$num "
((num++))
done
echo
done
```

---

9. Searching Algorithms in Shell Script

9.1 Linear Search

Algorithm:

```
#!/bin/bash

echo "Enter the number of elements"
read n
```

```
echo "Enter the array elements"
for ((i=0; i<n; i++))
do
read a[$i]
done

echo "Enter the element to be searched"
read item

j=0
while [ $j -lt $n -a $item -ne j]} ]
do
j=$((j + 1))
done

if [ $j -lt $n -a $item -eq j]} ]
then
echo "$item is present at location ((j + 1))"elseecho"item is not present in array"
fi
```

Time Complexity: O(n)
Space Complexity: O(1)

9.2 Binary Search

Algorithm (Sorted Array Required):

```
#!/bin/bash

echo "Enter sorted, space-separated numbers:"
read -r input
ARRAY=($input)

echo "Enter target number:"
read -r TARGET

LOW=0
HIGH=(({#ARRAY[@]} - 1))
FOUND=-1

while [[ $LOW -le HIGH]]; doMID =(( (LOW + HIGH) / 2 ))
if [[ ${ARRAY[MID]} -eq TARGET]]; thenFOUND =MID
break
elif [[ ${ARRAY[MID]} -lt TARGET]]; thenLOW =((MID + 1))
else
HIGH=$((MID - 1))
fi
done

if [[ $FOUND -ne -1 ]]; then
echo "Found $TARGET at index FOUND"elseecho"TARGET not found"
fi
```

Time Complexity: O(log n)
Space Complexity: O(1)

---

10. Special Shell Script Programs

10.1 Palindrome Checker

palindrome() {
s="$1''rev_s =$(echo "$s" | rev)

```
if [ "$rev_s" = "$s" ]; then
    echo "The string is a palindrome"
else
    echo "The string is NOT a palindrome"
fi
```

}

read -p "Enter a string: " str
palindrome "$str"

10.2 Fibonacci Series Generator

#!/bin/bash

echo "Enter the number of terms:"
read n

a=0
b=1

echo "Fibonacci series up to $n terms:"

i=0
while [$i -lt $n]doecho - n''$a "
fn=$((a+b))a =$b
b=$fni =$((i + 1))
done
echo

10.3 Odd/Even and Positive/Negative Checker

#!/bin/bash

echo "Enter a number:"
read num

if [$((numecho''$num is Even"
else

```
echo "$num is Odd"
fi
```

if [$num - gt0]; thenecho''$num is Positive"
else
echo "$num is Negative"
fi

11. Case Statement in Shell Script

```
#!/bin/bash

echo "Enter the option"
read option

echo "option=$option"
case $option in
```

1. echo "case 1" ;;
2. echo "case 2" ;;
3. echo "case 3" ;;
4. echo "case 4" ;;
   *)echo "Invalid case" ;;
   esac

12. Environment Variables & System Configuration

Important Commands:

Display all environment variables

```
env # Show all environment variables
echo $SHELL # Display current shell
echo $HOME # Show home directory
echo $USER # Show current user
```

Linux distribution information

```
cat /etc/os-release # Show OS details
cat /etc/lsb-release # Show Linux Standard Base info
uname -a # Show all system information
```

13. Software Installation & Management

Critical Commands:

Update package lists

sudo apt-get update # Update available packages list

Install software

sudo apt-get install gcc # Install GCC compiler
sudo apt-get install g++ # Install G++ compiler

Verify installation

gcc --version # Check GCC version
g++ --version # Check G++ version

---

14. Binary File Analysis (Advanced)

Critical Commands:

File type identification

file /bin/ls # Identify file type

Output: ELF 64-bit LSB shared object

ELF header examination

readelf -h /bin/ls # Show ELF header information

Shows: Magic number, Class, Architecture, Entry point

Symbol table inspection

readelf -s /bin/ls # Display symbol table

Shows: Function names, external dependencies

Program headers

```
readelf -l /bin/ls # Show program headers
```

Shows: PHDR, INTERP, LOAD, DYNAMIC segments

Section information

```
readelf -S /bin/ls # Display section headers
```

Shows: .text, .data, .bss, .rodata sections

Object dump utility

```
objdump -x /bin/ls # Comprehensive binary information
objdump -s /bin/ls # Display all sections
```

String extraction

```
strings /bin/ls # Extract printable strings from binary
```

Security features check

```
checksec --file=/bin/ls # Check security mechanisms
```

Detects: ASLR, Stack canaries, NX bit, PIE

---

15. Fuzzing & Security Testing

Fuzzing Program (Buffer Overflow Testing):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void vulnerable_function(char *input) {
char buffer[10];
strcpy(buffer, input); // Unsafe - can cause buffer overflow
printf("Processed: %s\n", buffer);
}
```

```
void fuzzer() {
srand(time(NULL));
int i, j;
for (i = 0; i < 1000; i++) {
int input_length = rand() % 50 + 1;
char *fuzzed_input = (char *)malloc(input_length + 1);
```

```
    for (j = 0; j < input_length; j++) {
        fuzzed_input[j] = (char)(rand() % 256);
    }
    fuzzed_input[input_length] = '\0';

    printf("Fuzzing input (length: %d)\n", input_length);
    vulnerable_function(fuzzed_input);

    free(fuzzed_input);
}
```

```
}
```

```
int main() {
fuzzer();
return 0;
}
```

Purpose: Test program with random inputs to find crashes/vulnerabilities

---

16. Kernel & System Security Settings

Critical Commands:

Check ASLR (Address Space Layout Randomization)

```
cat /proc/sys/kernel/randomize_va_space
```

0: ASLR disabled

1: ASLR partially enabled

2: ASLR fully enabled

Disable ASLR (for security testing)

echo 0 > /proc/sys/kernel/randomize_va_space

Enable full ASLR

echo 2 > /proc/sys/kernel/randomize_va_space

---

17. Quick Reference: Important Commands

Command	Purpose	Example
`gcc`	Compile C code	`gcc file.c -o output`
bash script.sh	Run shell script	bash linear_search.sh
`chmod`	Change permissions	`chmod 755 file`
chown	Change ownership	chown user:group file
`sudo`	Run as superuser	`sudo apt update`
find	Search files	find / -type f -name "*.c"
`grep`	Pattern search	`grep "ERROR" file.log`
readelf	Analyze ELF binaries	readelf -h /bin/ls
`strings`	Extract strings	`strings /bin/ls`
file	Identify file type	file /bin/ls

---

18. Lab Exam Preparation Tips

1. **Cryptography Programs:**
   - Know compilation: `gcc cipher.c -o cipher`
   - Practice with different input values
   - Understand encryption/decryption flow
2. **Shell Scripts:**
   - Master loop syntax (for, while, until)
   - Practice pattern generation
   - Understand array operations
   - Know comparison operators (-eq, -gt, -lt, etc.)
3. **Linux Commands:**
   - Memorize permission chmod values
   - Practice file/directory operations
   - Know firewall (UFW) configuration
   - Understand process monitoring (ps command)
4. **Binary Analysis:**
   - Learn ELF file format basics
   - Know readelf/objdump usage
   - Understand symbol tables and sections
5. **Time Management:**
   - Practice 15-minute time slots for each program
   - Know shortcuts (arrow keys in shell, tab completion)
   - Pre-compile common programs

References & Additional Resources

- Linux Man Pages: `man <command>`
- GCC Documentation: `gcc --help`
- Shell Scripting: BASH manual (online resources)
- Cryptography: Standard algorithm references
- System Security: Linux kernel documentation

Document Version: 1.0
Created: December 2025
For: MCA System Security Lab Exam Preparation