Ex. No.: 1 **Date:**

CAESAR CIPHER

Aim:

To implement Caesar Cipher technique using C.

Algorithm:

1. Declare two arrays to store plaintext and ciphertext
2. Prompt the user to enter plaintext
3. Loop till the end-of line marker comes
   a. get one plaintext character & put the same in plaintext[] array and increment i
   b. apply caesar 3 key shift cipher on the character and store in ciphertext[] array and increment x.
4. Print the ciphertext

Program Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int encrypt();
int decrypt();
char plaintext[50]={0},ciphertext[50]={0};
int c,d;
int main()
{
  int option;
  do
  {
   printf("Press 1 to Encrypt Plaintext \n");
   printf("Press 2 to Decrypt Ciphertext \n");
   printf("Press 9 to Exit \n");
   printf("Your option=");
   scanf("%d",&option);
   switch(option)
   {
    case 1: encrypt();
            printf("Ciphertext : %s \n",ciphertext);
            break;
    case 2: decrypt();
            printf("Plaintext : %s \n",plaintext);
          break;
    case 9: break;
    default: printf("Enter valid option! \n");
   }
  }while(option!=9);
  return 0;
}
```

```
int encrypt()
{
   static int i=0,j=0;
   printf("\nPlaintext : ");getchar();
   while((c=getchar()) != '\n')
   {
      plaintext[i++]=(char)c;
      ciphertext[j++]=(char)(c+3);
   }
   return 0;
 }

int decrypt()
{
   static int p=0,q=0;
   printf("\nCiphertext : ");getchar();

   while((d=getchar()) != '\n')
   {
         ciphertext[p++]=(char)d;
         plaintext[q++]=(char)(d-3);
   }
   return 0;
}
```

Output:

```
[root@localhost security lab]#  gcc caesar.c
[root@localhost security lab]#  ./a.out
Press 1 to Encrypt Plaintext
Press 2 to Decrypt Ciphertext
Press 9 to Exit
Your option=1

Plaintext : INDIA
Ciphertext : LQGLD
Press 1 to Encrypt Plaintext
Press 2 to Decrypt Ciphertext
Press 9 to Exit
Your option=2

Ciphertext : LQGLD
Plaintext : INDIA
[root@localhost security lab]#
```

Result:

Ex. No.: 2 **Date:**

RAIL-FENCE

Aim:

To implement Rail-Fence Cipher technique using C.

Algorithm:

1. Get the plaintext string from the user.
2. Take the string length of the plaintext.
3. Remove whitespaces from the string.
4. For each plaintext character do -
   a. If ch % 2 == 0 put in a[] array
   b. Else put in b[] array
5. Take each character in a[] array and put in s[] array and increment the index.
6. After all characters in a[] array are copied, then copy each character from b[] array and put into s[] array and increment the index.
7. Print the contents of s[] array to get ciphertext.

Program Code:

```
#include<stdio.h>
#include<string.h>
void main()
{
 int i,j,k=0,l=0,m=0,n,count=0;
 char str[20],s[20],a[10],b[10];

 printf("PLAIN TEXT :  ");
 scanf("%[^\n]s",str);   //it sets the delimiter for the scanned string as \n
 n=strlen(str);

 //Whitespace removal from string
 for (i = 0; i<n; i++)
       if (str[i] != ' ')
          s[count++] = str[i];

 //Transposition
 n=strlen(s);
 for(i=0;i<n;i++)
 {
          if(i%2==0)
          {
                    a[k]=s[i];
                    k++;
          }
```

```
        else
        {
                b[l]=s[i];
                l++;
        }
 }


 //Display section
 for(i=0;i<k;i++)
 {
        printf("%c ",a[i]);     //Row 1 print
        s[m]=a[i];
        m++;
 }
 printf("\n");
 for(i=0;i<l;i++)
 {
        printf(" %c",b[i]);    //Row 2 print
        s[m]=b[i];
        m++;
 }
 printf("\n\nCIPHER TEXT :  %s",s);
 getchar();
}
```

Output:

```
[root@localhost security lab]# gcc railfence.c
[root@localhost security lab]# ./a.out

PLAIN TEXT :  INDIA IS MY COUNTRY

I D A S Y O N R
 N I I M C U T Y

CIPHER TEXT :  IDASYONRNIIMCUTY

[root@localhost security lab]#
```

Result:

Ex. No.: 3 **Date:**

VIGNERE CIPHER

Aim:

To implement Vignere Cipher technique using C.

Algorithm:

1. Get the length of the key from the user.
2. Get the length of the plaintext from the user.
3. Next get the plaintext string from the user.
4 For each plaintext character do ch mod 65 or ch mod 97 and store result in s[] array.
5. For each value store in s[] array, take ch mod 26 and add the value to 65 & store in cipher[] array.
6. Print the contents of the cipher[] array to get ciphertext for the given plaintext.

Program Code:

```
//This program works for plaintext and keyword entered in Capital Letter only
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
char plaintext[20],keyword[20],ciphertext[20];
void encrypt();

void main()
{
        int i,j=0,n,k;

        printf("\nEnter the plaintext : ");
        scanf("%s",plaintext);
        n=strlen(plaintext);
        printf("\nEnter the keyword : ");
        scanf("%s",keyword);
        k=strlen(keyword);

      if(n>k)
      {
                for(i=k;i<n;i++)
                {
                        keyword[i]=keyword[j];  //fill the keyword till the size of plaintext
                        j++;
                }
        }

        encrypt(n);
        printf("Ciphertext:  %s\n",ciphertext);
}
```

```
void encrypt(int count)
{
        int i,sum;
        for(i=0;i<count;i++)
        {
                sum= plaintext[i]+keyword[i];
                if(sum<130||sum>180)
                {
                        printf("\nEnter input in CAPS only \n");
                        exit(0);
                }
                else ciphertext[i]= (sum % 26) + 65;
        }
}
```

Output:

```
[root@localhost security lab]# gcc bensvignere.c
[root@localhost security lab]# ./a.out

Enter the plaintext : WEAREDISCOVEREDSAVEYOURSELF

Enter the keyword : DECEPTIVE
Ciphertext:  ZICVTWQNGRZGVTWAVZHCCMEZIPY
[root@localhost security lab]#
```

Result:

Ex. No.: 4 **Date:**

HILL CIPHER

Aim:

To implement Hill Cipher technique using C.

Algorithm:

1. Store the 3x3 matrix contents for the encryption key k in an array a.
2. Store the 3x3 matrix contents of key inverse k^{-1}
3. Get the plaintext message from the user.
4. Compute the length of the plaintext message.
5. For each character of plaintext message's ascii value is subtracted with 65 & store in array c
6. For each element in array c, multiply arrays c & a
   a. Store result in t.
   b. Take t mod 26 and store in array d
7. Add array element d value and 65 and print the corresponding ciphertext character.

Program Code:

```
#include<stdio.h>
#include<string.h>
int check(int );
void calc(int );
//a[][] denotes encryption key K
//b[][] denotes K inverse
unsigned int a[3][3]={{17,17,5},{21,18,21},{2,2,19}};
unsigned int b[3][3]={{4,9,15},{15,17,6},{24,0,17}};
unsigned int c[20],d[20];
int n;
int main()
{
int i=0,t=0,j,q;
char msg[20];

printf("\nEnter plain text\n");
scanf("%s",msg);
n=strlen(msg);
q=check(n);

if(q==1) return 0;
else
{
for(i=0;i<n;i++)
{
```

```
        c[i]=msg[i]-65;
        printf("%d ",c[i]);
}

while(t<n)
{
  calc(t);
  t=t+3;
}

printf("\nEncrypted Cipher Text :");
for(i=0;i<n;i++)
        printf(" %c",d[i]+65);
for(i=0;i<3;i++)
{
        t=0;
        for(j=0;j<3;j++)
        {
                t=t+(d[j]*b[j][i]);
        }
        c[i]=t%26;
}
printf("\nDecrypted Cipher Text :");
for(i=0;i<n;i++)
        printf(" %c",c[i]+65);
printf("\n");
}
return 0;
}

void calc(int k)
{
   int i,j,t;

   for(i=0;i<3;i++)
   {
        t=0;
        for(j=0;j<3;j++)
        {
                t=t+(c[j+k]*a[j][i]);
        }
        d[i+k]=t%26;
   }
}

int check(int p)
{

 if(n%3!=0)
 {
```

```
    printf("\nEnter Plaintext in multiples of three \n");
    return 1;
  }
  else return 0;
}
```

Output:

```
[root@localhost security lab]# gcc hillfinal.c
[root@localhost security lab]# ./a.out

Enter plain text
PAYMOREMONEY
15 0 24 12 14 17 4 12 14 13 4 24
Encrypted Cipher Text : R R L M W B K A S P D H
Decrypted Cipher Text : P A Y M O R E M O N E Y
[root@localhost security lab]#
```

Result:

Ex. No.: 5 **Date:**

PLAYFAIR CIPHER

Aim:

To implement Playfair Cipher technique using C.

Algorithm:

1. Initialize the contents of the matrix.
2. Get the keyword and plaintext from the user.
3. Insert the keyword into the matrix except duplicate elements.
4. If I & J are encountered in the keyword then fill both in the same location in matrix.
5. Fill the remaining entries of the matrix with the character not already entered into it.
6. If the plaintext has an odd number of characters, append an 'x' to the end to make it even.
7. Break the plaintext into pairs of letters.
8. Locate the letters in the key matrix using the rules and encrypt with letters from matrix-
   a. If the letters appear on the same row of the table, replace them with the letters to their immediate right
   b. If the letters appear on the same column of the table, replace them with the letters immediately below
   c. If the letters are in different rows and columns, replace the pair with the letters on the same row.
   d. If two letters are same then use filler x in place of one letter.
9. Display the encrypted text

Program Code:

```
#include<stdio.h>
#include<string.h>
void initialize();
void fillMatrix();
void fillRemaining(int);
void displayMatrix();
void encrypt();
void searchMatrix(int,int);
char matrix[5][5],keyword[25],plaintext[25],ciphertext[25],flag[26];
int row,col,row1,col1,row2,col2;

void main()
{
        initialize();
        printf("\nEnter Keyword: ");
        scanf("%s",keyword);
        printf("\nEnter Plaintext:");
        scanf("%s",plaintext);
        fillMatrix();
        encrypt();
```

```
	displayMatrix();
	printf("\nCiphertext: %s \n",ciphertext);
}


void initialize()
{
	int i,j;

	for(i=0;i<26;i++)
		flag[i]=0;
	for(i=0;i<5;i++)
		for(j=0;j<5;j++)
			matrix[i][j]=0;
}

void fillMatrix()
{
	int k=0,i,j,p,m=0,n,val;

	flag[9]=1;  //since J is put in I
	n=strlen(keyword);
	for(i=0;i<5;i++)
	{
		for(j=0;j<5;j++)
		{
		b:	if(k<n)
			{
				val=keyword[k]-65;
				if(flag[val]==0)
				{
					matrix[i][j]=keyword[k];
					flag[val]=1;k++;
				}
				else
				{
					k++;
					goto b;
				}
			}
			else
			{
				col=j;
				goto c;
			}
	   }
	}

c:	row=i;
	  while(m<26)
```

```
        {
                if(flag[m]==0)
                        fillRemaining(m);
                m++;
      }
}

void fillRemaining(int m)
{
        if(row<5)
        {
                if(col<5)
              {  matrix[row][col]=m+65;

                }
                else
                {
                        col=0;
                        row++;
                        matrix[row][col]=m+65;
                }
        }
        col++;
}


void encrypt()
{
        int psize,i=0,j=1;

        psize=strlen(plaintext);
        if(psize%2!=0)   //Adding filler character X since plaintext is odd in size
                strcat(plaintext,"X");
        while(j<=psize)
        {
                searchMatrix(i,j);
                if(row1==row2)  //same row plaintext elements
                {
                        if(plaintext[i]==plaintext[j])    //Adding $ if two plaintext elements refer to
same character
                        {
                                ciphertext[i]=matrix[row1][(col1+1)%5];
                                ciphertext[j]='$';
                        }
                        else
                        {
                                ciphertext[i]= matrix[row1][(col1+1)%5];
                                ciphertext[j]= matrix[row2][(col2+1)%5];
                        }
                }
```

```
                else if(col1==col2)  //same col plaintext elements
                    {
                        ciphertext[i]= matrix[(row1+1)%5][col1];
                        ciphertext[j]= matrix[(row2+1)%5][col2];
                    }
                    else // different row and different col plaintext elements
                    {
                        ciphertext[i]= matrix[row1][col2];
                        ciphertext[j]= matrix[row2][col1];
                    }
                i+=2;
                j+=2;printf("\n");
        }
}

void searchMatrix(int i,int j)
{
        int r,t,foundRow,foundCol;

        for(r=0;r<5;r++)
        {
                for(t=0;t<5;t++)
                {
                        if(plaintext[i]==matrix[r][t])
                        {
                                row1=r;col1=t;
                                foundRow=1;
                        }
                        if(plaintext[j]==matrix[r][t])
                        {
                                row2=r;col2=t;
                                foundCol=1;
                        }
                }
                if((foundRow==1)&&(foundCol==1))
                        break;
        }
}

void displayMatrix()
{
        int i,j;
        printf("\n");
        for(i=0;i<5;i++)
        {
                for(j=0;j<5;j++)
                        printf("%c ",matrix[i][j]);
                printf("\n");
        }
}
```

Output:

```
[root@localhost security lab]# gcc playfair.c
[root@localhost security lab]# ./a.out
Enter Keyword: MONARCHY
Enter Plaintext:INDIA

M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z

Ciphertext: GABKBA
```

Result:

Ex. No.: 6 **Date:**

DES

Aim:

To implement DES symmetric cipher technique using openssl library in C.

Algorithm:

1. Initialize the value of the key and the plaintext.
2. Allocate memory for plaintext and key.
3. Print the hexadecimal value of the plaintext
4. Print the hexadecimal value of the key
5. Prepare the key for use with DES_cfb64_encrypt
6. Set the parity of the key passed to odd using DES_set_odd_parity( ).
7. Check the passed key is weak by using DES_set_key_checked( ).
8. Encrypt the message in Cipher Feedback Mode using DES_cfb64_encrypt( ).
9. Print the obtained cipher text message.
10. Decrypt the ciphert text using DES_cfb64_decrypt( ).
11. Print the obtained plain text message.

Program Code:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <openssl/des.h>

char *Encrypt( char *Key, char *Msg, int size)
{
	static char*	Res;
	int		n=0;
	DES_cblock	Key2;
	DES_key_schedule schedule;

	Res = ( char * ) malloc( size );

	/* Prepare the key for use with DES_cfb64_encrypt */
	memcpy( Key2, Key,8);
	DES_set_odd_parity( &Key2 );
	DES_set_key_checked( &Key2, &schedule );

	/* Encryption occurs here */
	DES_cfb64_encrypt( ( unsigned char * ) Msg, ( unsigned char * ) Res,size, &schedule,
&Key2, &n, DES_ENCRYPT );
	return (Res);
}
```

```
char *Decrypt( char *Key, char *Msg, int size)
{
	static char*	Res;
	int		n=0;
	DES_cblock	Key2;
	DES_key_schedule schedule;
	 Res = ( char * ) malloc( size );

	/* Prepare the key for use with DES_cfb64_encrypt */
	memcpy( Key2, Key,8);
	DES_set_odd_parity( &Key2 );
	DES_set_key_checked( &Key2, &schedule );

	/* Decryption occurs here */
	DES_cfb64_encrypt( ( unsigned char * ) Msg, ( unsigned char * ) Res,size, &schedule,
&Key2, &n, DES_DECRYPT );
	return (Res);
}

int main()
{
	  char key[]="BENEDICT";
	  char plaintext[]="ENJOYCNS";
	  char *decrypted;
	  char *encrypted;
	  encrypted=malloc(sizeof(plaintext));
	  decrypted=malloc(sizeof(plaintext));
	  printf("\nPlaintext\t\t\t : %s",plaintext);
	  printf("\nPlaintext in Hex Code \t\t : %X",plaintext);
	  printf("\nKey in Hex Code \t\t : %X",key);
	  memcpy(encrypted,Encrypt(key,plaintext,sizeof(plaintext)), sizeof(plaintext));
	  printf("\nEncrypted Text\t\t\t : %s",encrypted);
	  memcpy(decrypted,Decrypt(key,encrypted,sizeof(plaintext)), sizeof(plaintext));
	  printf("\nDecrypted Text\t\t\t : %s",decrypted);
	  printf("\nDecrypted Text in Hex Code \t : %X \n",decrypted);
	  return (0);
}
```

Output:

```
[root@localhost security lab]# gcc desopen.c -lssl -lcrypto
[root@localhost security lab]# ./a.out
Plaintext                       : ENJOYCNS
Plaintext in Hex Code           : BFD28266
Key in Hex Code                 : BFD2826F
Encrypted Text                  : ����{S��S
Decrypted Text                  : ENJOYCNS
Decrypted Text in Hex Code      : 848D030
[root@localhost security lab]#
```

Result:

Ex. No.: 7 **Date:**

RSA

Aim:

To implement RSA asymmetric key cryptosystem using C.

Algorithm:

1. Select two large prime numbers p and q
2. Compute n=pxq
3. Choose system modulus: Ø(n)=(p-1)x(q-1)
4. Select a random encryption key e such that gcd(e,Ø(n)=1
5. Decrypt by computing d=1 mod Ø(n)
6. Print the public key{e,n}
7. Print the private key{d,n}

Program Code:

```
#include <stdio.h>
#include <math.h>
int power(int,unsigned int,int);
int gcd(int,int);
int multiplicativeInverse(int,int,int);
int main()
{
  int p,q,n,e,d,phi,M,C;

  printf("\nEnter two prime numbers p and q that are not equal : ");
  scanf("%d %d",&p,&q);
  n = p * q;
  phi = (p - 1)*(q - 1);
  printf("Phi(%d) = %d",n,phi);
  printf("\nEnter the integer e : ");
  scanf("%d",&e);
  if(e >= 1 && e < phi)
  {
        if(gcd(phi,e)!=1)
        {
              printf("\nChoose proper value for e !!!\n");
              return 1;
        }
  }

  //Key Generation
  d = multiplicativeInverse(e,phi,n);

  printf("\nPublic Key PU = {%d,%d}",e,n);
```

```
  printf("\nPrivate Key PR = {%d,%d}",d,n);

  //Encryption
  printf("\nMessage M = ");
  scanf("%d",&M);
  C = power(M,e,n);
  printf("\nCiphertext C = %d \n",C);

  //Decryption
  M = power(C,d,n);
  printf("\nDecrypted Message M = %d \n",M);

  return 0;
}

int power(int x, unsigned int y, int p)
{
    int res = 1;      // Initialize result

    x = x % p;  // Update x if it is more than or equal to p

    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
            res = (res*x) % p;

        // y must be even now
        y = y>>1;     // y = y/2
        x = (x*x) % p;
    }
    return res;
}

int gcd ( int a, int b )
{
  int c;
  while ( a != 0 )
  {
          c = a;
          a = b % a;
          b = c;
  }
  return b;
}
```

```
int multiplicativeInverse(int a, int b, int n)
{
        int sum,x,y;

      for(y=0;y<n;y++)
      {
            for(x=0;x<n;x++)
            {
                  sum=a*x + b*(-y);
                  if(sum==1)
                        return x;
            }
      }
}
```

Output:

```
[root@localhost security lab]# gcc rsaben.c
[root@localhost security lab]# ./a.out
Enter two prime numbers p and q that are not equal : 17 11
Phi(187) = 160
Enter the integer e : 7
Public Key PU = {7,187}
Private Key PR = {23,187}
Message M = 88
Ciphertext C = 11
Decrypted Message M = 88
[root@localhost security lab]#
```

Result:

Ex. No.: 8 **Date:**

DIFFIE-HELLMAN KEY EXCHANGE

Aim:

To implement Diffie-Hellman key exchange using C.

Algorithm:

1. Get a prime number q as input from the user.
2. Get a value xa and xb which is less than q.
3. Calculate primitive root α
4. For each user A, generate a key Xa < q
5. Compute public key, α pow(Xa) mod q
6. Each user computes Ya
7. Print the values of exchanged keys.

Program Code:

```
//This program uses fast exponentiation function power instead of pow library function
#include <stdio.h>
#include <math.h>
int power( int,unsigned int,int);
int main()
{
  int x,y,z,count,ai[20][20];
  int alpha,xa,xb,ya,yb,ka,kb,q;

  printf("\nEnter a Prime Number \"q\":");
  scanf("%d",&q);
  printf("\nEnter a No \"xa\" which is less than value of q:");
  scanf("%d",&xa);
  printf("\nEnter a No \"xb\" which is less than value of q:");
  scanf("%d",&xb);
  printf("\nEnter alpha:");
  scanf("%d",&alpha);
  ya = power(alpha,xa,q);
  yb = power(alpha,xb,q);
  ka = power(yb,xa,q);
  kb = power(ya,xb,q);
  printf("\nya = %d \nyb = %d \nka = %d \nkb = %d \n",ya,yb,ka,kb);
  if(ka == kb)
          printf("\nThe secret keys generated by User A and User B are same\n");
  else
       printf("\nThe secret keys generated by User A and User B are not same\n");
  return 0;
}
```

```
int power(int x, unsigned int y, int p)
{
   int res = 1;      // Initialize result

   x = x % p;  // Update x if it is more than or equal to p

   while (y > 0)
   {
      // If y is odd, multiply x with result
      if (y & 1)
         res = (res*x) % p;

      // y must be even now
      y = y>>1;     // y = y/2
      x = (x*x) % p;
   }
   return res;
}
```

Output:

```
[root@localhost security lab]# gcc diffie-hellman.c
[root@localhost security lab]# ./a.out

Enter a Prime Number "q":353

Enter a No "xa" which is less than value of q:97

Enter a No "xb" which is less than value of q:233

Enter alpha:3

ya = 40
yb = 248
ka = 160
kb = 160

The secret keys generated by User A and User B are same
```

Result:

Ex. No.: 9 **Date:**

MD5

Aim:

To implement MD5 hash technique using openssl library in C.

Algorithm:

1. Get the value of string from command line argument.
2. Initialize the digest table.
3. Use EVP_get_digestbyname function and pass on the argument MD5
4. Initialize digest context.
5. Use digestUpdate() function to hash bytes of data at d into the digest contexts
6. Use EVP_DigestFinal_ex() function to retrieve the digest value from ctx and places in md.
7. Cleanup the digest context ctx.
8. Print the value of digest computed.

Program Code:

```
#include <stdio.h>
#include <string.h>
#include <openssl/evp.h>

void main(int argc, char *argv[])
{
  EVP_MD_CTX mdctx;
  const EVP_MD *md;
  char input[80];

  unsigned char output[EVP_MAX_MD_SIZE];
  int output_len, i;

  strcpy(input,argv[1]);


  /* Initialize digests table */
  OpenSSL_add_all_digests();

  /* You can pass the name of another algorithm supported by your version of OpenSSL here */
  /* For instance, MD2, MD4, SHA1, RIPEMD160 etc. Check the OpenSSL documentation for
details */
  md = EVP_get_digestbyname("MD5");

  if(!md) {
        printf("Unable to init MD5 digest\n");
        exit(1);
  }
```

```
  EVP_MD_CTX_init(&mdctx);
  EVP_DigestInit_ex(&mdctx, md, NULL);
  EVP_DigestUpdate(&mdctx, input, strlen(input));
  /* to add more data to hash, place additional calls to EVP_DigestUpdate here */
  EVP_DigestFinal_ex(&mdctx, output, &output_len);
  EVP_MD_CTX_cleanup(&mdctx);

  /* Now output contains the hash value, output_len contains length of output, which is 128 bit or
16 byte in case of MD5 */

  printf("Digest is: ");
  for(i = 0; i < output_len; i++) printf("%02x", output[i]);
  printf("\n");
}
```

Output:

```
[root@localhost security lab]# gcc md5final.c -lssl -lcrypto
[root@localhost security lab]# ./a.out REC
Digest is: d6d269952320c4fb5e50f278c94a098c
[root@localhost security lab]#
```

Result:

Ex. No.: 10 **Date:**

SHA-1

Aim:

To implement SHA-1 hash technique using openssl library in C.

Algorithm:

1. Get the input string from command line arguments.
2. Check if the number of arguments is not equal to 2. If so print error and return.
3. Generate hash string for argv[1] by passing it to sha1 function.
4. The value returned is stored in temp variable.
5. Loop through the contents of temp and put into buf variable.
6. Print the contents of buf variable.

Program Code:

```
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>

int main(int argn, char *argv[]) {

   int i = 0;
   unsigned char temp[SHA_DIGEST_LENGTH];
   char buf[SHA_DIGEST_LENGTH*2];

   if ( argn != 2 ) {
      printf("Usage: %s string\n", argv[0]);
      return -1;
   }

   memset(buf, 0x0, SHA_DIGEST_LENGTH*2);
   memset(temp, 0x0, SHA_DIGEST_LENGTH);

   SHA1((unsigned char *)argv[1], strlen(argv[1]), temp);

   for (i=0; i < SHA_DIGEST_LENGTH; i++) {
      sprintf((char*)&(buf[i*2]), "%02x", temp[i]);
   }

   printf("SHA1 of %s is %s\n", argv[1], buf);

   return 0;

}
```

Output:

[root@localhost security lab]# **gcc sha1.c -lssl -lcrypto**
[root@localhost security lab]# **./a.out REC**
SHA1 of REC is 09ebb92a1478021f08e37a2ffe4ce10e8ced419f
[root@localhost security lab]#

Result:

Ex. No.: 11 **Date:**

DSA

Aim:

To implement Digital Signature Algorithm(DSA) using C.

Algorithm:

1. Get the prime number p and its divisor q from the user.
2. Get the value of h from the user.
3. Compute the value of g.
4. Get the private key xa from the user.
5. Compute the user's public key y.
6. Get the per-message secret key k and hash value of message M.
7. Compute the value of z using g, k & p
8. Compute z % q to get the value of r
9. Compute the multiplicative inverse.
10. Compute the value of s.
11. Print the signature (r, s).

Program Code:

```
#include <stdio.h>
#include <math.h>
int power(int,unsigned int,int);
int multiplicativeInverse(int,int,int);
int main()
{
  int p,q,h,g,r,s,t,x,y,z,k,inv,hash;

  printf("\nEnter prime number p and enter q prime divisor of (p-1): ");
  scanf("%d %d",&p,&q);
  printf("\nEnter h such that it greater than 1 and less than (p-1): ");
  scanf("%d",&h);

  //Compute g
  t = (p-1)/q;
  g = power(h,t,p);

  printf("\nEnter user's private key such that it is greater than 0 and less than q : ");
  scanf("%d",&x);

  //Computer user's public key
  y = power(g,x,p);
```

```
  printf("\nEnter user's per-message secret key k such that it is greater than 0 and less than q : ");
  scanf("%d",&k);
  printf("\nEnter the hash(M) value : ");
  scanf("%d",&hash);

  //Signing. Compute r and s pair
  z = power(g,k,p);
  r = z % q;
  inv = multiplicativeInverse(k,q,p);
  s = inv * (hash + x * r) % q;

  //Display
  printf("\n*********Computed Values*********");
  printf("\ng = %d",g);
  printf("\ny = %d",y);
  printf("\nGenerated Signature Sender = (%d, %d) \n",r,s);
}

int power(int x, unsigned int y, int p)
{
    int res = 1;      // Initialize result

    x = x % p;  // Update x if it is more than or equal to p

    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
            res = (res * x) % p;

        // y must be even now
        y = y >> 1;     // y = y/2
        x = (x * x) % p;
    }
    return res;
}

int multiplicativeInverse(int a, int b, int n)
{
        int sum,x,y;
        for(y=0;y<n;y++)
        {
                for(x=0;x<n;x++)
                {
                        sum = a * x + b * (-y);
                        if(sum == 1)
                                return x;
                }
        }
}
```

Output:

```
[root@localhost security lab]# gcc dsa.c
[root@localhost security lab]# ./a.out

Enter prime number p and enter q prime divisor of (p-1): 1279 71

Enter h such that it greater than 1 and less than (p-1): 3

Enter user's private key such that it is greater than 0 and less than q : 15

Enter user's per-message secret key k such that it is greater than 0 and less than q : 10

Enter the hash(M) value : 123

*********Computed Values*********
g = 1157
y = 851
Generated Signature Sender = ( 32, 39)
[root@localhost security lab]#
```

Result:

Ex. No.: 12 **Date:**

GNUPG

Aim:

To do a secure data transmission using GnuPG.

Basic Workflow:

In order to encrypt the file the sender should have a private open key of the person to whom the file is going to be sent. The open key is used by the sender to encrypt the data and cannot be used to decrypt it. The receiver can decrypt the file using his private secret key and a passphrase.

Algorithm:

1. Install GnuPG
2. Generate public key and private key
    a. Select the algorithm for keys to be generated(RSA/DSA)
    b. Set the keysize between 1024 to 4096.
    c. Set validity of the key in terms of days.
    d. Enter your name, email and comments.
    e. Enter a secure passphrase to generate the keys finally.
3. Create a revocation certificate.
4. Import public key of receiver
5. Encrypt the data file with the option -r
6. Decrypt the encrypted data file at the receiver with option -d

Output:

```
[root@localhost rkhunter-1.4.2]# yum install gnupg
[root@localhost rkhunter-1.4.2]# gpg --gen-key
gpg (GnuPG) 1.4.19; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: keyring `/root/.gnupg/secring.gpg' created
gpg: keyring `/root/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
```

```
      0 = key does not expire
    <n>  = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 100
Key expires at Sat 24 Sep 2016 07:39:58 PM IST
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
   "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Ben
Name must be at least 5 characters long
Real name: Benedict
Email address: benedict.jn@rajalakshmi.edu.in
Comment: Message
You selected this USER-ID:
   "Benedict (Message) <benedict.jn@rajalakshmi.edu.in>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.
[root@localhost ~]# gpg --gen-revoke benedict.jn@rajalakshmi.edu.in
[root@localhost ~]# gpg -r benedict.jn@rajalakshmi.edu.in topSecret.txt
[root@localhost ~]# gpg -o topSecret.txt -d topSecret.txt.gpg
```

Result:

Ex. No.: 13a **Date:**

STUDY OF HONEYPOTS

Aim:

To study the concept of honeypots and it's types.

Description:

A honeypot is a deception trap, designed to entice an attacker into attempting to compromise the information systems in an organisation. If deployed correctly, a honeypot can serve as an early-warning and advanced security surveillance tool, minimizing the risks from attacks on IT systems and networks. Honeypots can also analyze the ways in which attackers try to compromise an information system, providing valuable insight into potential system loopholes.

Types of Honeypots:

Honeypots can be classified based on their deployment (use/action) and based on their level of involvement. Based on deployment, honeypots may be classified as: production honeypots and research honeypots.

Production honeypots are easy to use, capture only limited information, and are used primarily by companies or corporations. Production honeypots are placed inside the production network with other production servers by an organization to improve their overall state of security. Normally, production honeypots are low-interaction honeypots, which are easier to deploy. They give less information about the attacks or attackers than research honeypots.

Research honeypots gather information about the motives and tactics of the Black hat community targeting different networks. Based on design criteria, it can be classified as- a) Pure honeypots b) high-interaction honeypots c) low-interaction honeypots.

Pure honeypots are full-fledged production systems. The activities of the attacker are monitored by using a casual tap that has been installed on the honeypot's link to the network. No other software needs to be installed. Even though a pure honeypot is useful, stealthiness of the defense mechanisms can be ensured by a more controlled mechanism.

High-interaction honeypots imitate the activities of the production systems that host a variety of services and, therefore, an attacker may be allowed a lot of services to waste his time. By employing virtual machines, multiple honeypots can be hosted on a single physical machine. Therefore, even if the honeypot is compromised, it can be restored more quickly. In general, high-interaction honeypots provide more security by being difficult to detect, but they are expensive to maintain. If virtual machines are not available, one physical computer must be maintained for each honeypot, which can be exorbitantly expensive. Example: Honeynet.

Low-interaction honeypots simulate only the services frequently requested by attackers. Since they consume relatively few resources, multiple virtual machines can easily be hosted on one physical system, the virtual systems have a short response time, and less code is required, reducing the complexity of the virtual system's security. Example: Honeyd.

Examples of Honeypots-

1. **Deception Toolkit:** DTK was the first Open Source honeypot released in 1997. It is a collection of Perl scripts and C source code that emulates a variety of listening services. Its primary purpose is to deceive human attackers.

2. **LaBrea :** This is designed to slow down or stop attacks by acting as a sticky honeypot to detect and trap worms and other malicious codes. It can run on Windows or Unix.

3. **Honeywall CDROM :** The Honeywall CDROM is a bootable CD with a collection of open source software. It makes honeynet deployments simple and effective by automating the process of deploying a honeynet gateway known as a Honeywall. It can capture, control and analyse all inbound and outbound honeynet activity.

4. **Honeyd :** This is a powerful, low-interaction Open Source honeypot, and can be run on both UNIX-like and Windows platforms. It can monitor unused IPs, simulate operating systems at the TCP/IP stack level, simulate thousands of virtual hosts at the same time, and monitor all UDP and TCP based ports.

Result:

Ex. No.: 13b **Date:**

HONEYPOTS

Aim:

To setup and monitor honeypots using honeyd tool.

Algorithm:

1. Install honeyd on one of the system.
2. Create honeyd configuration file.
3. Launch honeyd with options -i -d and -f after configuration files are created.
4. Ping from Windows/Linux machine to the honeyd machine with it's IP address.
5. After honeyd successful deployment, check required port of honeyd machine are open
6. Use nmap to scan the open ports of honeyd machine.
7. If the required ports are open, the honeyd is functioning correctly.

Output:

[root@localhost security lab]# **yum install honeyd**
[root@localhost security lab]# **vi /etc/honeyd.conf**

```
create windows
set windows personality "Microsoft Windows NT 4.0 Server SP5-SP6"
add windows tcp port 80 open
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
set windows default tcp action reset
set windows default udp action reset
set windows ethernet “00:00:24:ab:8c:12”
bind 172.16.8.13 to p4p1
```

[root@localhost security lab]# **honeyd -i p4p1 -d -f honeyd.conf**

```
Honeyd V1.5c Copyright (c) 2002-2007 Niels Provos
honeyd[4546]: started with -i p4p1 -d -f honeyd.conf
Warning: Impossible SI range in Class fingerprint "IBM OS/400 V4R2M0"
Warning: Impossible SI range in Class fingerprint "Microsoft Windows NT 4.0 SP3"
honeyd[4546]: listening promiscuously on p4p1: (arp or ip proto 47 or (udp and src port 67 and dst port 68) or (ip )) and not ether src 48:0f:cf:6d:60:57
honeyd[4546]: Running with root privileges
```

Another terminal

[root@localhost security lab]# **nmap -p 135,139,445,1337 172.16.8.13**

Starting Nmap 5.00 (http://nmap.org) at 2011-05-06 13:13 EDT

Interesting ports on someone (172.20.73.77):

PORT	STATE	SERVICE
135/tcp	open	msrpc
139/tcp	open	netbios-ssn
445/tc	open	microsoft-ds
1337/tcp	closed	waste

MAC Address: 48:0f:cf:6d:60:57 (Connect AS)

Nmap done: 1 IP address (1 host up) scanned in 0.37 seconds

Result:

Ex. No.: 14a **Date:**

STUDY OF KALI LINUX DISTRIBUTION

Aim:

To study about Kali Linux: an advanced penetrating testing and security auditing Linux distribution.

Description:

Kali Linux is a Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali Linux contains several hundred tools aimed at various information security tasks, such as Penetration Testing, Forensics and Reverse Engineering. Kali Linux is developed, funded and maintained by Offensive Security, a leading information security training company.

Kali Linux was released on the 13th March, 2013 as a complete, top-to-bottom rebuild of BackTrack Linux, adhering completely to Debian development standards. Features are listed below-

- **More than 600 penetration testing tools**
- **Free and Open Source Software**
- **Open source Git tree:** All of the source code which goes into Kali Linux is available for anyone who wants to tweak or rebuild packages to suit their specific needs.
- **FHS compliant:** It adheres to the Filesystem Hierarchy Standard, allowing Linux users to easily locate binaries, support files, libraries, etc.
- **Wide-ranging wireless device support:** A regular sticking point with Linux distributions has been support for wireless interfaces. Kali Linux supports many wireless devices.
- **Custom kernel, patched for injection:** As penetration testers, the development team often needs to do wireless assessments and Kali Linux kernel has the latest injection patches included.
- **Developed in a secure environment:** The Kali Linux team is made up of a small group of individuals who are the only ones trusted to commit packages and interact with the repositories, all of which is done using multiple secure protocols.
- **GPG signed packages and repositories:** Every package in Kali Linux is signed by each individual developer who built and committed it, and the repositories subsequently sign the packages as well.
- **Multi-language support:** It has multilingual support, allowing more users to operate in their native language and locate the tools they need for the job.
- **Completely customizable:** It can be customized to the requirements of the users.
- **ARMEL and ARMHF support:** It is suitable for ARM-based single-board systems like the Raspberry Pi and BeagleBone Black.

Security Tools:

Kali Linux includes many well known security tools and are listed below-

- Nmap
- Aircrack-ng
- Kismet
- Wireshark
- Metasploit Framework
- Burp suite
- John the Ripper
- Social Engineering Toolkit
- Airodump-ng

Aircrack-ng Suite:

It is a complete suite of tools to assess WiFi network security.It focuses on different areas of WiFi security:

- Monitoring: Packet capture and export of data to text files for further processing by third party tools.
- Attacking: Replay attacks, deauthentication, fake access points and others via packet injection.
- Testing: Checking WiFi cards and driver capabilities (capture and injection).
- Cracking: WEP and WPA PSK (WPA 1 and 2).

All tools are command line which allows for heavy scripting. A lot of GUIs have taken advantage of this feature. It works primarily Linux but also Windows, OS X, FreeBSD, OpenBSD, NetBSD, as well as Solaris and even eComStation 2.

Result:

Ex. No.: 14b **Date:**

WIRELESS AUDIT

Aim:

To perform wireless audit on Access Point and decrypt WPA keys using aircrack-ng tool in Kalilinux OS.

Algorithm:

1. Check the current wireless interface with iwconfig command.
2. Get the channel number, MAC address and ESSID with iwlist command.
3. Start the wireless interface in monitor mode on specific AP channel with airmon-ng.
4. If processes are interfering with airmon-ng then kill those process.
5. Again start the wireless interface in monitor mode on specific AP channel with airmon-ng.
6. Start airodump-ng to capture Initialization Vectors(IVs).
7. Capture IVs for atleast 5 to 10 minutes and then press Ctrl + C to stop the operation.
8. List the files to see the captured files
9. Run aircrack-ng to crack key using the IVs collected and using the dictionary file rockyou.txt
10. If the passphrase is found in dictionary then Key Found message displayed; else print Key Not Found.

Output:

```
root@kali:~# iwconfig
eth0      no wireless extensions.

wlan0     IEEE 802.11bgn  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
lo        no wireless extensions.

root@kali:~# iwlist wlan0 scanning
wlan0     Scan completed :
          Cell 01 - Address: 14:F6:5A:F4:57:22
                    Channel:6
                    Frequency:2.437 GHz (Channel 6)
                    Quality=70/70  Signal level=-27 dBm
                    Encryption key:on
                    ESSID:"BENEDICT"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s
                    Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s
                              36 Mb/s; 48 Mb/s; 54 Mb/s
```

Mode:Master
Extra:tsf=00000000425b0a37
Extra: Last beacon: 548ms ago
IE: WPA Version 1
Group Cipher : TKIP
Pairwise Ciphers (2) : CCMP TKIP
Authentication Suites (1) : PSK

root@kali:~# airmon-ng start wlan0

Found 2 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to kill (some of) them!

PID Name
1148 NetworkManager
1324 wpa_supplicant

PHY	Interface	Driver	Chipset
phy0	wlan0	ath9k_htc	Atheros Communications, Inc. AR9271 802.11n

Newly created monitor mode interface wlan0mon is *NOT* in monitor mode.
Removing non-monitor wlan0mon interface...

WARNING: unable to start monitor mode, please run "airmon-ng check kill"

root@kali:~# airmon-ng check kill

Killing these processes:

PID Name
1324 wpa_supplicant

root@kali:~# airmon-ng start wlan0

PHY	Interface	Driver	Chipset
phy0	wlan0	ath9k_htc	Atheros Communications, Inc. AR9271 802.11n

(mac80211 **monitor mode** vif enabled for [phy0]wlan0 on [phy0]**wlan0mon**)
(mac80211 station mode vif disabled for [phy0]wlan0)

root@kali:~# airodump-ng -w atheros -c 6 --bssid 14:F6:5A:F4:57:22 wlan0mon

CH 6][Elapsed: 5 mins][2016-10-05 01:35][**WPA handshake**: 14:F6:5A:F4:57:

BSSID	PWR	RXQ	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	E
14:F6:5A:F4:57:22	-31	100	3104	10036	0	6	54e.	WPA	CCMP	PSK	B

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
14:F6:5A:F4:57:22	70:05:14:A3:7E:3E	-32	2e-	0	0	10836

root@kali:~# ls -l
total 10348
-rw-r--r-- 1 root root 10580359 Oct 5 01:35 **atheros-01.cap**
-rw-r--r-- 1 root root 481 Oct 5 01:35 atheros-01.csv
-rw-r--r-- 1 root root 598 Oct 5 01:35 atheros-01.kismet.csv
-rw-r--r-- 1 root root 2796 Oct 5 01:35 atheros-01.kismet.netxml

root@kali:~# aircrack-ng -a 2 atheros-01.cap -w /usr/share/wordlists/rockyou.txt
[00:00:52] 84564 keys tested (1648.11 k/s)

KEY FOUND! [rec12345]

Master Key : CA 53 9B 5C 23 16 70 E4 84 53 16 9E FB 14 77 49
A9 7A A0 2D 9F BB 2B C3 8D 26 D2 33 54 3D 3A 43

Transient Key : F5 F4 BA AF 57 6F 87 04 58 02 ED 18 62 37 8A 53
38 86 F1 A2 CA 0D 4A 8D D6 EC ED 0D 6C 1D C1 AF
81 58 81 C2 5D 58 7F FA DE 13 34 D6 A2 AE FE 05
F6 53 B8 CA A0 70 EC 02 1B EA 5F 7A DA 7A EC 7D

EAPOL HMAC 0A 12 4C 3D ED BD EE C0 2B C9 5A E3 C1 65 A8 5C

Result:

Ex. No.: 15 **Date:**

SNORT IDS

Aim:

To demonstrate Intrusion Detection System(IDS) using snort tool.

Algorithm:

1. Download and extract the latest version of snort
2. Install development packages - libpcap and pcre.
3. Install snort
4. Verify the installation is correct.
5. Create the configuration file, rule file and log file directory
6. Create snort.conf and icmp.rules files
7. Execute snort from the command line
8. Ping to yahoo website from another terminal
9. Watch the alert messages in the log files

Output:

```
[root@localhost security lab]# cd /usr/src
[root@localhost security lab]# wget https://www.snort.org/downloads/snort/snort-2.9.8.3.tar.gz
[root@localhost security lab]# tar xvzf snort-2.9.8.3.tar.gz

[root@localhost security lab]# yum install libpcap*  pcre*    -y

[root@localhost security lab]# cd snort-2.9.8.3
[root@localhost security lab]#  . /configure
[root@localhost security lab]#  make
[root@localhost security lab]#  make install

[root@localhost security lab]# snort --version
   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.8.2 GRE (Build 335)
   ""    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
         Copyright (C) 2014-2015 Cisco and/or its affiliates. All rights reserved.
         Copyright (C) 1998-2013 Sourcefire, Inc., et al.
         Using libpcap version 1.7.3
         Using PCRE version: 8.38 2015-11-23
         Using ZLIB version: 1.2.8
[root@localhost security lab]# mkdir /etc/snort
[root@localhost security lab]# mkdir /etc/snort/rules
[root@localhost security lab]# mkdir /var/log/snort
[root@localhost security lab]# vi /etc/snort/snort.conf
          add this line-              include /etc/snort/rules/icmp.rules
```

[root@localhost security lab]# **vi /etc/snort/rules/icmp.rules**
alert icmp any any -> any any (msg:"ICMP Packet"; sid:477; rev:3;)

[root@localhost security lab]# **snort -i p4p1 -c /etc/snort/snort.conf -l /var/log/snort/**

Another terminal
[root@localhost security lab]# **ping www.yahoo.com**

Ctrl + C

[root@localhost security lab]# **vi /var/log/snort/alert**

[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:11.187877 192.168.43.148 -> 106.10.138.240
ICMP TTL:64 TOS:0x0 ID:45855 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14680 Seq:64 ECHO

[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:11.341739 106.10.138.240 -> 192.168.43.148
ICMP TTL:52 TOS:0x38 ID:2493 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14680 Seq:64 ECHO REPLY

[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:12.189727 192.168.43.148 -> 106.10.138.240
ICMP TTL:64 TOS:0x0 ID:46238 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14680 Seq:65 ECHO

[**] [1:477:3] ICMP Packet [**]
[Priority: 0]
10/06-15:03:12.340881 106.10.138.240 -> 192.168.43.148
ICMP TTL:52 TOS:0x38 ID:7545 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14680 Seq:65 ECHO REPLY

Result:

Ex. No.: 16a **Date:**

STUDY OF ROOTKITS

Aim:

To study about rootkits and different rootkits scanner in Linux.

Description:

A rootkit is a program (or combination of several programs) designed to take fundamental control (in Unix terms “root” access, in Windows terms “Administrator” access) of a computer system, without authorization by the system’s owners and legitimate managers.

Rootkit scanner is a scanning tool to ensure system is clean of nasty tools. This tool scans for rootkits, backdoors and local exploits by running tests like:

- MD5 hash compare
- Look for default files used by rootkits
- Wrong file permissions for binaries
- Look for suspected strings in LKM and KLD modules
- Look for hidden files
- Optional scan within plaintext and binary files

There are many different versions of rootkits that perform basically the same function. Well known Linux rootkits include LRK, tOrn, and Adore and some Windows Rootkits include NTROOT, NTKap, and Nullsys.

Not only are rootkits designed to hide the presence of an attacker; they are also used to gain future administrator-level (root) access, launch distributed denial of service (ddos), or obtain financial or confidential information. Because rootkits are designed to hide the presence of an attacker, it is necessary to understand how a rootkit functions.

When a rootkit is installed, it overwrites many commands used on a daily basis such as ls, ps, or netstat. By overwriting such commands, the intrusion can be masked from the administrators.

Types of Rootkits

There are many places where a malware can install itself into an operating system. So, mostly the type of rootkit is determined by its location where it performs its subversion of the execution path. This includes:

1. User Mode Rootkits
2. Kernel Mode Rootkits
3. MBR Rootkits/bootkits

User mode rootkits involve system hooking in the user or application space. Whenever an application makes a system call, the execution of that system call follows a predetermined path and a Windows rootkit can hijack the system call at many points along that path.

Kernel mode rootkitsKernel are the tools that run in the kernel space, hence making it

really hard to detect. The entire operating system would be altered in the process, which would help in the process of hiding the fact that the system is compromised.

MBR rootkits/Bootkits can infect startup code like the Master Boot Record (MBR), Volume Boot Record (VBR) or boot sector, and in this way, can be used to attack full disk encryption systems.

There is difference between Virus and Rootkits are listed below-

Virus	**Rootkit**
Primarily runs as an application process	Primarily runs as a part of OS/kernel
Usually gains user level access	Gains root/admin level access
Does not open backdoors	Opens backdoors — viz., port, IP, etc.
Does not provide any remote access	Provides remote access to attacker
Fairly easy to detect and remove	Extremely difficult to detect and remove
Is meant to create nuisance and data damage	Is meant to cause privacy/data theft

Detecting Rootkits in Linux:

There are various tools to detect rootkits in Linux and some of these are mentioned below-

Zeppoo – Zeppoo allows you to detect rootkits on i386 and x86_64 architecture under Linux, by using /dev/kmem and /dev/mem. Moreover it can also detect hidden tasks, connections, corrupted symbols, system calls and so many other things.

Chkrootkit – chkrootkit is a tool to locally check for signs of a rootkit. It is a shell script that checks system binaries for rootkit modification. It can also detect some well-known LKM rootkits.

Rkhunter – rkhunter (Rootkit Hunter) is a Unix-based tool that scans for rootkits, backdoors and possible local exploits. rkhunter is a shell script which carries out various checks on the local system to try and detect known rootkits and malware. It also performs checks to see if commands have been modified, if the system startup files have been modified, and various checks on the network interfaces, including checks for listening applications.

Result:

Ex. No.: 16b **Date:**

INSTALLATION OF ROOTKITS

Aim:

To install and explore the various options of Rkhunter rootkit scanner.

Algorithm:

1. Download rkhunter tool from https://rootkit.nl/projects/rootkit_hunter.html
   or using wget from the command line-
   wget http://downloads.sourceforge.net/project/rkhunter/rkhunter/1.4.2/rkhunter-1.4.2.tar.gz
2. Unzip the file and install rkhunter as a root user.
3. Run the RKH updater to get the latest updates to the database
4. Setting cron job and email alerts
5. Set execute permission on the file rkhunter.sh
6. Scan the entire file system for rootkits.

Output:

[root@localhost rkhunter-1.4.2]# **wget http://downloads.sourceforge.net/project/**
rkhunter/rkhunter/1.4.2/rkhunter-1.4.2.tar.gz

[root@localhost rkhunter-1.4.2]# **gunzip rkhunter-1.4.2.tar.gz**

[root@localhost rkhunter-1.4.2]# **tar xvf rkhunter-1.4.2.tar**

[root@localhost rkhunter-1.4.2]# **cd rkhunter-1.4.2/**

[root@localhost rkhunter-1.4.2]# **./installer.sh --layout default --install**

[root@localhost rkhunter-1.4.2]# **/usr/local/bin/rkhunter --update**

[root@localhost rkhunter-1.4.2]# **/usr/local/bin/rkhunter --propupd**

[root@localhost rkhunter-1.4.2]# **vi /etc/cron.daily/rkhunter.sh**

[root@localhost rkhunter-1.4.2]# **chmod 755 /etc/cron.daily/rkhunter.sh**

[root@localhost rkhunter-1.4.2]# **rkhunter --check**

System checks summary

==========

File properties checks...

Files checked: 136

Suspect files: 0

Rootkit checks...

Rootkits checked : 383

Possible rootkits: 0

Applications checks...
All checks skipped

The system checks took: 2 minutes and 57 seconds

All results have been written to the log file: /var/log/rkhunter/rkhunter.log

One or more warnings have been found while checking the system.
Please check the log file (/var/log/rkhunter/rkhunter.log)

Result: