RAJALAKSHMI ENGINEERING COLLEGE, THANDALAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CS6711 – SECURITY LAB MANUAL
Regulations 2013

B.E. Computer Science & Engineering Course

Prepared by:
Benedict J.N. and Roxanna Samuel, C.S.E. Department, Rajalakshmi Engineering College

INDEX

Department of Computer Science and Engineering

Vision

To promote highly ethical and innovative computer professionals through excellence in teaching, training and research.

Mission

To produce globally competent professionals, motivated to learn the emerging technologies and to be innovative in solving real world problems.

To promote research activities amongst the students and the members of faculty that could benefit the society.

To impart moral and ethical values in their profession.

Programme Educational Objectives (PEOs)

PEO I

To equip students with essential background in computer science, basic electronics and applied mathematics.

PEO II

To prepare students with fundamental knowledge in programming languages and tools and enable them to develop applications.

PEO III

To encourage the research abilities and innovative project development in the field of networking, security, data mining, web technology, mobile communication and also emerging technologies for the cause of social benefit.

PEO IV

To develop professionally ethical individuals enhanced with analytical skills, communication skills and organizing ability to meet industry requirements.

Prepared by:
Benedict J.N. and Roxanna Samuel, C.S.E. Department, Rajalakshmi Engineering College

Programme Outcomes (POs)

(a) The graduates will demonstrate knowledge of Mathematics, Science and Engineering.

(b) The graduates will demonstrate an ability to design and conduct experiments, analyze and interpret data.

(c) The graduates will demonstrate knowledge in C, C++ and Java programming.

(d) The graduates will demonstrate their skill in applying software engineering methodologies in their project work.

(e) The graduates will be able to develop applications in networking using network simulators.

(f) The graduates will be able to comprehend the concepts of security threats and mechanisms to overcome them.

(g) The students will be able to groom themselves to the requirement of corporate challenges.

(h) The graduates will be capable of developing web based applications in specific verticals.

(i) The students will be able to understand and develop mobile applications and value added services for the 3G systems.

(j) The graduates will be able to demonstrate their understanding in the emerging areas of data mining and web mining.

(k) The graduates will be able to develop software components using emerging technologies like JAVA, .NET, PYTHON, PERL, PHP etc.

(l) The graduates will be able to demonstrate their competitive skills among their peers.

Mapping of PEO's with Programme Outcomes (PO's)

PEOs	Programme Outcomes											
	a	b	c	d	e	f	g	h	i	j	k	l
I	√	√	√	√	√						√	√
II		√	√	√	√		√	√		√		√
III		√	√	√	√	√	√	√	√	√		
IV						√					√	√

Prepared by:
Benedict J.N. and Roxanna Samuel, C.S.E. Department, Rajalakshmi Engineering College

Graduate Attributes

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialisation for the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design and development solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

4. **Investigation of complex anslysis:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modelling to complex engineering activities, with an understanding of the limitations.

6. **Engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with t h e society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Prepared by:
Benedict J.N. and Roxanna Samuel, C.S.E. Department, Rajalakshmi Engineering College

12. **Lifelong learning:** Recognise the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Mapping of Graduate Attributes with Programme Outcomes (PO's)

GA	Programme Outcomes											
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**
1	√	√										√
2		√							√		√	√
3			√	√	√	√	√	√		√		√
4		√				√			√			
5				√	√	√	√	√	√	√		
6	√	√		√							√	√
7	√	√	√								√	√
8						√					√	√
9	√	√	√	√	√	√	√	√	√	√	√	√
10				√	√	√			√		√	√
11	√										√	√
12		√	√	√	√	√	√	√	√	√		

Course Objectives

1. Be exposed to the different cipher techniques.
2. Learn to implement the algorithms DES, RSA, MD5, SHA-1.
3. Have hands on experience to perform wireless security audit on access points.
4. Learn to use network security tools like GnuPG, Kismet, Snort.
5. Be familiar with firewall configuration.

Mapping of Course Objectives with Programme Outcomes

COs	Programme Outcomes											
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**
1	√		√			√					√	√
2	√	√	√			√					√	√
3		√	√			√	√		√		√	√
4		√	√		√	√	√				√	√
5		√				√	√				√	√

Mapping of Course Objectives with Programme Educational Objectives (PEO's)

COs	Programme Educational Objectives			
	I	**II**	**III**	**IV**
1	√	√		
2	√	√		
3	√	√	√	√
4	√	√	√	√
5		√		√

Prepared by:
Benedict J.N. and Roxanna Samuel, C.S.E. Department, Rajalakshmi Engineering College

Course Outcomes

On completion of this course:

a. Students are able to solve different cipher techniques.

b. Students are able to implement cryptographic algorithms.

c. Students are able to perform wireless security audit of access points.

d. Students are able to use various network security tools.

e. Students are able to configure firewalls.

Mapping of Course Objectives with Course Outcomes

COs	Course Outcomes					
	a	b	c	d	e	
1	√	√				
2	√	√				
3			√	√		
4		√		√		
5				√	√	

Mapping of Course Outcomes with Programme Outcomes (PO's)

COs	Programme Outcomes											
	a	b	c	d	e	f	g	h	i	j	k	l
a	√		√			√					√	√
b	√	√	√			√	√				√	√
c	√	√	√		√	√	√				√	√
d	√	√	√		√	√	√				√	√
e			√		√	√	√					√

Prepared by:
Benedict J.N. and Roxanna Samuel, C.S.E. Department, Rajalakshmi Engineering College

Mapping of Course Outcomes with Programme Educational Objectives (PEO's)

COs	Programme Educational Objectives			
	I	II	III	IV
a	√		√	
b	√	√	√	
c	√	√	√	√
d	√	√	√	√
e		√	√	√

Mapping of Graduate Attributes with Course Outcomes (CO's)

GAs	Course Outcomes					
	a	b	c	d	e	
1	√	√	√	√		
2		√	√	√	√	
3	√	√	√	√	√	
4	√		√	√		
5			√	√	√	
6				√	√	
7						
8						
9	√	√	√	√	√	
10			√	√	√	
11				√	√	
12	√	√	√	√	√	

CS6711 SECURITY LABORATORY L T P C

0 0 3 2

OBJECTIVES:

The student should be made to:

-Be exposed to the different cipher techniques

-Learn to implement the algorithms DES, RSA,MD5,SHA-1

-Learn to use network security tools like GnuPG, KF sensor, Net Strumbler

LIST OF EXPERIMENTS:

1. Implement the following SUBSTITUTION & TRANSPOSITION TECHNIQUES concepts:
   a) Caesar Cipher
   b) Playfair Cipher
   c) Hill Cipher
   d) Vigenere Cipher
   e) Rail fence – row & Column Transformation
2. Implement the following algorithms
   a) DES
   b) RSA Algorithm
   c) Diffiee-Hellman
   d) MD5
   e) SHA-1

5 Implement the SIGNATURE SCHEME - Digital Signature Standard

6.Demonstrate how to provide secure data storage, secure data transmission and for creating digital signatures (GnuPG).

7.Setup a honey pot and monitor the honeypot on network (KF Sensor)

8.Installation of rootkits and study about the variety of options

9.Perform wireless audit on an access point or a router and decrypt WEP and WPA.(Net Stumbler)

10.Demonstrate intrusion detection system (ids) using any tool (snort or any other s/w)

TOTAL: 45 PERIODS

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

SOFTWARE:

C / C++ / Java or equivalent compiler

GnuPG, KF Sensor or Equivalent, Snort, Net Stumbler or Equivalent

HARDWARE:

Standalone desktops- 30 Nos.

(or)

Server supporting 30 terminals or more.

LIST OF EXPERIMENTS

1. Write a C program to implement Caesar Cipher.
2. Write a C program to implement Playfair Cipher.
3. Write a C program to implement Hill Cipher.
4. Write a C program to implement Vigenere Cipher.
5. Write a C program to implement Rail Fence Technique.
6. Write a C program to implement DES Technique.
7. Write a C program to implement RSA Technique.
8. Write a C program to implement Diffie-Hellman Key Exchange.
9. Write a C program to implement MD5 Hash Technique.
10. Write a C program to implement SHA-1 Hash Technique.
11. Write a C program to implement Digital Signature Scheme.
12. Demonstrate secure data transmission using GnuPG.
13. To setup Honeypot and monitor in network using Honeyd.
14. Install rootkits and study various options.
15. Perform wireless audit on access point or router and decrypt WEP and WPA using Kismet.
16. Demonstrate firewalls using Iptables.

Software Details:

Operating System – Fedora 22 Linux distribution with kernel version 4.1.3

Tools - kismet-0.0.2013.03.R1-4.fc22.i686
gnupg-1.4.19-1.fc22.i686
honeyd-1.5c-21.fc20.i686
iptables-1.4.21-14.fc22.i686

LAB PLAN

S. No.	Name of the Experiment	Batch I	Batch II
1	Caesar Cipher implementation		
2	Playfair Cipher implementation		
3	Hill Cipher implementation		
4	Vigenere Cipher implementation		
5	Rail Fence Technique implementation		
6	DES implementation		
7	RSA implementation		
8	Diffie- Hellman implementation		
9	MD5 implementation		
10	SHA-1 implementation		
11	Digital Signature Scheme implementation		
12	a. Study of GnuPG		
	b. Secure Data Transmission using GnuPG		
13	a. Study of Honeyd		
	b. Honeypot setup & monitor using Honeyd		
14	Rootkits installation and its study		
15	a. Study of Iptables *		
	b. Firewall Demonstration using Iptables *		
16	a. Study of Snort IDS		
	b. Demonstration of Snort IDS		

*** Denotes Content Beyond Syllabus**

CAESAR CIPHER

Aim:

To write a C program to implement Caesar Cipher technique.

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>
int main()
{
   char plaintext[100]={0}, ciphertext[100]={0};
    int c;
   printf("Plaintext:");
   while((c=getchar()) != '\n')
   {
      static int x=0, i=0;
      plaintext[i++]=(char)c;
      ciphertext[x++]=(char)(c+3);
   }
    printf("Cipher text:");
   printf("%s\n",ciphertext);
   return 0;
}
```

Output:

```
[root@localhost security lab]# gcc caes.c -o caesar
[root@localhost security lab]# ./caesar
Plaintext:  abc
Cipher text: def
```

PLAY FAIR CIPHER

Aim:

To write a C program to implement Playfair Cipher technique.

Algorithm:

1. Initialize the contents of the table to zero.
2. Get the length of the key
3. Get the key string from the user.
4. Insert each element of the key into the table.
5. Fill the remaining entries of the table with the character not already entered into the table.
6. Enter the length of the plaintext.
7. Get the plaintext string.
8.

Program Code:

```
#include<stdio.h>
int check(char table[5][5],char k)
{
        int i,j;
        for(i=0;i<5;++i)
        for(j=0;j<5;++j)
        {
                if(table[i][j]==k)
                        return 0;
        }
        return 1;
}

void main()
{
        int i,j,key_len;
        char table[5][5];
        for(i=0;i<5;++i)
            for(j=0;j<5;++j)
                table[i][j]='0';
printf("**********Playfair Cipher**********\n\n");
```

```
printf("Enter the length of the Key. ");
scanf("%d",&key_len);

char key[key_len];
printf("Enter the Key. ");
for(i=-1;i<key_len;++i)
{
        scanf("%c",&key[i]);
        if(key[i]=='j')
                key[i]='i';
}
int flag;
int count=0;
// inserting the key into the table
for(i=0;i<5;++i)
{
        for(j=0;j<5;++j)
        {
                flag=0;
                while(flag!=1)
                {
                        if(count>key_len)
                                goto l1;
                        flag=check(table,key[count]);
                        ++count;
                }// end of while
                table[i][j]=key[(count-1)];
        }// end of inner for
}// end of outer for

l1:printf("\n");
int val=97;
```

```
//inserting other alphabets
for(i=0;i<5;++i)
{
  for(j=0;j<5;++j)
  {
        if(table[i][j]>=97 && table[i][j]<=123)
        {}
        else
        {
              flag=0;
              while(flag!=1)
              {
                    if('j'==(char)val)
                          ++val;
                    flag=check(table,(char)val);
                    ++val;
              }// end of while
              table[i][j]=(char)(val-1);
        }//end of else
  }// end of inner for
}// end of outer for

printf("The table is as follows:\n");
for(i=0;i<5;++i)
{
  for(j=0;j<5;++j)
  {
        printf("%c ",table[i][j]);
  }
printf("\n");
}

int l=0;
```

```
printf("\nEnter the length of plain text.(without spaces) ");
scanf("%d",&l);
printf("\nEnter the Plain text. ");
char p[l];
for(i=-1;i<l;++i)
{
        scanf("%c",&p[i]);
}

for(i=-1;i<l;++i)
{
        if(p[i]=='j')
                p[i]='i';
}

printf("\nThe replaced text(j with i)");
for(i=-1;i<l;++i)
printf("%c ",p[i]);
count=0;
for(i=-1;i<l;++i)
{
if(p[i]==p[i+1])
count=count+1;
}
printf("\nThe cipher has to enter %d bogus char.It is either 'x' or 'z'\n",count);
int length=0;
if((l+count)%2!=0)
length=(l+count+1);
else
length=(l+count);
printf("\nValue of length is %d.\n",length);
char p1[length];
```

```
//inserting bogus characters.
char temp1;
int count1=0;
for(i=-1;i<l;++i)
{
p1[count1]=p[i];
if(p[i]==p[i+1])
{
        count1=count1+1;
        if(p[i]=='x')
                p1[count1]='z';
        else     p1[count1]='x';
}
count1=count1+1;
}
//checking for length
char bogus;
if((l+count)%2!=0)
{
if(p1[length-1]=='x')
p1[length]='z';
else
p1[length]='x';
}

printf("The final text is:");
for(i=0;i<=length;++i)
printf("%c ",p1[i]);
char cipher_text[length];
int r1,r2,c1,c2;
int k1;
for(k1=1;k1<=length;++k1)
{
```

```
for(i=0;i<5;++i)
{
for(j=0;j<5;++j)
{
if(table[i][j]==p1[k1])
{
r1=i;
c1=j;
}
else
if(table[i][j]==p1[k1+1])
{
r2=i;
c2=j;
}
}//end of for with j
}//end of for with i
(r1==r2)
{
cipher_text[k1]=table[r1][(c1+1)%5];
cipher_text[k1+1]=table[r1][(c2+1)%5];
}
else
if(c1==c2)
{
cipher_text[k1]=table[(r1+1)%5][c1];
cipher_text[k1+1]=table[(r2+1)%5][c1];
}
else
{
cipher_text[k1]=table[r1][c2];
cipher_text[k1+1]=table[r2][c1];
}
```

```
k1=k1+1;
}//end of for with k1


printf("\n\nThe Cipher text is:\n ");
for(i=1;i<=length;++i)
printf("%c ",cipher_text[i]);
}
```

Output:

```
[root@localhost security lab]# gcc playfair.c
[root@localhost security lab]# ./a.out
**********Playfair Cipher**********

Enter the length of the Key. 8
Enter the Key. monarchy

The table is as follows:
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

Enter the length length of plain text.(without spaces) 12

Enter the Plain text. moveforwardx

The replaced text(j with i)
 m o v e f o r w a r d x
The cipher has to enter 0 bogus char.It is either 'x' or 'z'

Value of length is 12.
The final text is:
 m o v e f o r w a r d x

The Cipher text is:
 o n u f p h n z r m b z
[root@localhost security lab]#
```

HILL CIPHER

Aim:

To write a C program to implement Hill Cipher technique.

Algorithm:

1. Get the 9 character(ch) key from the user.
2. Take ch%65 or ch%97 and store them in key array.
3. Get the length of the plaintext string.
4. If the length of the string is not in multiples of 3 then add bogus characters to fill.
5. Get the string from the user and store in plaintext array.
6. For each character of string and key do mod 26 and store that value in ciphertext array.
7. Print the ciphertext array contents.

Program Code:

```
#include<stdio.h>
void main()
{
int l,i,j,temp1;
int k[3][3], p[3][1], c[3][1];
char ch;
printf("\nThe cipher has a key of length 9. ie. a 3*3 matrix.\nEnter the 9 character key. ");


for(i=0;i<3;++i)
{
for(j=0;j<3;++j)
{
        scanf("%c",&ch);
        if(65<=ch && ch<=91)
                k[i][j]=(int)ch%65;
        else
                k[i][j]=(int)ch%97;
}
}

for(i=0;i<3;++i)
{
for(j=0;j<3;++j)
{
        printf("%d ",k[i][j]);
}
printf("\n");
}

printf("\nEnter the length of string to be encoded(without spaces). ");
scanf("%d",&l);
```

```
temp1=check(l);

if(temp1>0)
printf("You have to enter %d bogus characters.",temp1);

char pi[l+temp1];
printf("\nEnter the string. ");
for(i=-1;i<l+temp1;++i)
{
scanf("%c",&pi[i]);
}

int temp2=l;
int n=(l+temp1)/3;
int temp3;
int flag=0;
int count;
printf("\n\nThe encoded cipher is : ");

while(n>0)
{
count=0;
for(i=flag;i<flag+3;++i)
{
if(65<=pi[i] && pi[i]<=91)
temp3=(int)pi[i]%65;
else
temp3=(int)pi[i]%97;

p[count][0]=temp3;
count=count+1;
}

int k1;
for(i=0;i<3;++i)
c[i][0]=0;

for(i=0;i<3;++i)
{
for(j=0;j<1;++j)
{
for(k1=0;k1<3;++k1)
c[i][j]+=k[i][k1]*p[k1][j];
}
}

for(i=0;i<3;++i)
{
c[i][0]=c[i][0]%26;
printf("%c ",(char)(c[i][0]+65));
```

```
}

n=n-1;
flag=flag+3;
}
}

int check(int x)
{
  int a,b,c;

        if(x%3==0)
                return 0;

        a=x/3;
        b=3*(a+1);
        c=b-x;
        return c;
}
```

Output:

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

The cipher has a key of length 9. ie. a 3*3 matrix.
Enter the 9 character key. backupabc
1 0 2
10 20 15
0 1 2

Enter the length of string to be encoded(without spaces). 10
You have to enter 2 bogus characters.
Enter the string. retreatnowxx


The encoded cipher is : D P Q R Q E V K P Q L R
[root@localhost security lab]#
```

VIGNERE CIPHER

Aim:

To write a C program to implement Vignere Cipher.

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:

```
#include<stdio.h>
void main()
{
int I, kl, pl;
char p[pl], k[kl];
printf("Enter the length of the key stream. ");
scanf("%d",&kl);
printf("Enter the length of the plain text stream.(Without spaces) ");
scanf("%d",&pl);
printf("\nEnter the Key. ");
for(i=-1;i<kl;++i)
        scanf("%c",&k[i]);
printf("\nEnter the Plain text. ");
for(i=-1;i<pl;++i)
        scanf("%c",&p[i]);
int s[3][pl];
for(i=0;i<pl;++i)
{
        if(65<=p[i] && p[i]<=91)
                s[0][i]=p[i]%65;
        else
        s[0][i]=p[i]%97;
```

```
}

for(i=0;i<pl;++i)
        printf("%d ",s[0][i]);
int count=0;
while(count<pl)
{
        for(i=0;i<kl;++i)
        {
                if(65<=k[i] && k[i]<=91)
                        s[1][count+i]=k[i]%65;
                else
                        s[1][count+i]=k[i]%97;
        }
        count=count+kl;
}
printf("\n");
for(i=0;i<pl;++i)
        printf("%d ",s[1][i]);
printf("\n");
for(i=0;i<pl;++i)
        printf("%d ",s[2][i]);
printf("\n\nThe cipher text is: ");
char cipher[kl];
for(i=0;i<pl;++i)
{
        s[2][i]=(s[0][i]+s[1][i])%26;
        cipher[i]=(char)(s[2][i]+65);
        printf("%c ",cipher[i]);
}
}
```

Output:

[root@localhost security lab]# gcc vigenere2.c

[root@localhost security lab]# ./a.out

Enter the length of the key stream. 9

Enter the length of the plain text stream.(Without spaces) 27

Enter the Key. deceptive

Enter the Plain text. wearediscoveredsaveyourself

22 4 0 17 4 3 8 18 2 14 21 4 17 4 3 18 0 21 4 24 14 20 17 18 4 11 5

3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4 3 4 2 4 15 19 8 21 4

The cipher text is: Z I C V T W Q N G R Z G V T W A V Z H C Q Y G L M G J

[root@localhost security lab]#

RAIL FENCE TECHNIQUE

Aim:

To write a C program to implement Rail-Fence technique.

Algorithm:

1. Get the plaintext string from the user.
2. Take the string length of the plaintext.
3. For each plaintext character do the following-
   a. If ch % 2 == 0 put in a[] array
   b. Else put in b[] array
4. Take each character in a[] array and put in s[] array and increment the index.
5. After all characters in a[] array are copied, then copy each character from b[] array and put into s[] array and increment the index.
6. 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;
 char s[20],a[10],b[10];

 printf("enter a string:");
 scanf("%s",s);
 for(i=0;i<strlen(s);i++)
 {
    if(i%2==0)    //even position
   {
       a[k]=s[i];
       k++;
    }
   else    //odd position
   {
       b[l]=s[i];
        l++;
    }
 }


 for(i=0;i<k;i++)
 {
      printf("%c ",a[i]);
      s[m]=a[i];
      m++;
 }
 printf("\n");
```

```
 for(i=0;i<l;i++)
 {
     printf(" %c",b[i]);
     s[m]=b[i];
     m++;
 }
 printf("\n\ncipher text is %s",s);
 getchar();
}
```

Output:

```
[root@localhost security lab]# gcc railfence.c
[root@localhost security lab]# ./a.out
enter a string:shootdowntheaircraft
s o t o n h a r r f
 h o d w t e i c a t

cipher text is sotonharrfhodwteicat
[root@localhost security lab]#
```

RSA

Aim:

To write a C program to implement RSA cryptosystem.

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<stdlib.h>
#include<math.h>
#include<string.h>
long int p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();

void main()
{
        printf("\nENTER FIRST PRIME NUMBER\n");
        scanf("%d",&p);
        flag=prime(p);
        if(flag==0) {
                printf("\nWRONG INPUT\n");
                getchar();
                exit(1);
```

```
        }
        printf("\nENTER ANOTHER PRIME NUMBER\n");
        scanf("%d",&q);
        flag=prime(q);

        if(flag==0||p==q) {
                printf("\nWRONG INPUT\n");
                getchar();
                exit(1);
        }
        printf("\nENTER MESSAGE\n");
        fflush(stdin);
        scanf("%s",msg);
        for (i=0;msg[i]!=NULL;i++)
        m[i]=msg[i];
        n=p*q;
        t=(p-1)*(q-1);
        ce();
        printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
        for (i=0;i<j-1;i++)
        printf("\n%ld\t%ld",e[i],d[i]);
        encrypt();
        decrypt();
        getchar();
}
int prime(long int pr) {
        int i;
        j=sqrt(pr);
        for (i=2;i<=j;i++) {
                if(pr%i==0)
                   return 0;
        }
```

```
        return 1;
}

void ce() {

        int k=0;
        for (i=2;i<t;i++) {
                if(t%i==0)
                   continue;
                flag=prime(i);
                if(flag==1&&i!=p&&i!=q) {
                        e[k]=i;
                        flag=cd(e[k]);
                        if(flag>0) {
                                d[k]=flag;
                                k++;
                        }
                        if(k==99)
                              break;
                }
        }
}

long int cd(long int x) {
        long int k=1;
        while(1) {
                k=k+t;
                if(k%x==0)
                   return(k/x);
        }
}

void encrypt() {
```

```
	long int pt,ct,key=e[0],k,len;
	i=0;
	len=strlen(msg);
	while(i!=len) {

		pt=m[i];
		pt=pt-96;
		k=1;
		for (j=0;j<key;j++) {
			k=k*pt;
			k=k%n;
		}
		temp[i]=k;
		ct=k+96;
		en[i]=ct;
		i++;
	}
	en[i]=-1;
	printf("\nTHE ENCRYPTED MESSAGE IS\n");
	for (i=0;en[i]!=-1;i++)
	printf("%c",en[i]);
}

void decrypt() {
	long int pt,ct,key=d[0],k;
	i=0;
	while(en[i]!=-1) {
		ct=temp[i];
		k=1;
		for (j=0;j<key;j++) {
			k=k*ct;
			k=k%n;
		}
```

```
            pt=k+96;
            m[i]=pt;
            i++;
        }

        m[i]=-1;
        printf("\nTHE DECRYPTED MESSAGE IS\n");
        for (i=0;m[i]!=-1;i++)
        printf("%c",m[i]);
}
```

Output:

```
[root@localhost security lab]# gcc rsa.c -lm
[root@localhost security lab]# ./a.out
ENTER FIRST PRIME NUMBER
7
ENTER ANOTHER PRIME NUMBER
17
ENTER MESSAGE
hello
POSSIBLE VALUES OF e AND d ARE
5       77
11      35
13      37
19      91
23      71
29      53
31      31
37      13
THE ENCRYPTED MESSAGE IS
cc
THE DECRYPTED MESSAGE IS
hello[root@localhost security lab]#
```

DIFFIE-HELLMAN

Aim:

To write a C program to implement Diffie-Hellman key exchange technique.

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:

```
#include <stdio.h>
#include <math.h>
void main()
{
int q,alpha,xa,xb,ya,yb,ka,kb, x,y,z,count,ai[20][20];
printf("Enter a Prime Number \"q\":");
scanf("%d",&q);
printf("Enter a No \"xa\" which is lessthan value of q:");
scanf("%d",&xa);
printf("Enter a No \"xb\" which is lessthan value of q:");
scanf("%d",&xb);
for(x=0;x<q-1;x++) //Primitive Root Calculation
for(y=0;y<q-1;y++)
ai[x][y] = ((int)pow(x+1,y+1))%q;
for(x=0;x<q-1;x++)
{
count = 0;
for(y=0;y<q-2;y++)
{
for(z=y+1;z<q-1;z++)
if(ai[x][y] == ai[x][z])
{
count = 1;
break;
}
if(count == 1)
break;
}
if (count == 0 )
{
alpha = x+1;
break;
```

```
}
}
printf("alpha = %d\n",alpha);
ya = ((int)pow(alpha,xa))%q; yb = ((int)pow(alpha,xb))%q;
ka = ((int)pow(yb,xa))%q; kb = ((int)pow(yb,xb))%q;
printf("ya = %d\nyb = %d\nka = %d\nkb = %d\n",ya,yb,ka,kb);
if(ka == kb) printf("The keys exchanged are same");
else printf("The keys exchanged are not same");
}
```

Output:

```
[root@localhost security lab]# gcc diffiehellman.c -lm
[root@localhost security lab]# ./a.out
Enter a Prime Number "q":23
Enter a No "xa" which is lessthan value of q:5
Enter a No "xb" which is lessthan value of q:7
alpha = 2
ya = 4
yb = 4
ka = 4
kb = 4
The keys exchanged are same
[root@localhost security lab]#
```

MD-5

Aim:

To write a C program to implement MD-5 hash technique.

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  */
  /* 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]# ./a.out IIT
Digest is: 1ce322ec4920fa4d0f5673f226fa8988
```

SHA-1

Aim:

To write a C program to implement SHA-1 hash technique.

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]#
```

DIGITAL SIGNATURE SCHEME

Aim:

To write a C program to implement digital signature scheme.

Algorithm:

1. Generate private key and public key using RSA algorithm.
2. Enable all algorithms using OpenSSL_add_all_algorithms() function.
3. Allocate empty PKEY structure to put the private key.
4. Read the private key and store in PEM format.
5. Check the read RSA private key is valid or not.
6. If valid print the details of the key.

Program Code:

```
#include <stdio.h>
#include <string.h>
#include <openssl/x509v3.h>
#include <openssl/objects.h>
#include <openssl/pem.h>
#include <openssl/evp.h>

int main() {
  EVP_PKEY *privkey;
  FILE *fp;
  RSA *rsakey;

  /* ------------------------------------------------------------ *
   * Next function is essential to enable openssl functions     *
   ------------------------------------------------------------ */
  OpenSSL_add_all_algorithms();

  privkey = EVP_PKEY_new();

  fp = fopen ("test-key.pem", "r");

  PEM_read_PrivateKey( fp, &privkey, NULL, NULL);

  fclose(fp);

  rsakey = EVP_PKEY_get1_RSA(privkey);

  if(RSA_check_key(rsakey)) {
    printf("RSA key is valid.\n");
  }
  else {
    printf("Error validating RSA key.\n");
  }
```

```
  RSA_print_fp(stdout, rsakey, 3);

  PEM_write_PrivateKey(stdout,privkey,NULL,NULL,0,0,NULL);

  exit(0);
}
```

Output:

```
[root@localhost security lab]# openssl genrsa -out test-key.pem 512
Generating RSA private key, 512 bit long modulus
.......++++++++++++
.....++++++++++++
e is 65537 (0x10001)
[root@localhost security lab]# gcc digitalsign.c -lssl -lcrypto
[root@localhost security lab]# ./a.out
RSA key is valid.
  Private-Key: (512 bit)
  modulus:
     00:d6:03:7a:02:19:5b:70:fb:9d:a9:f4:cc:6f:01:
     35:52:48:84:b0:aa:b1:3c:5c:ab:1d:34:95:3d:bd:
     fa:ca:64:ed:67:89:a2:33:83:83:2f:1f:c1:2e:9e:
     d4:13:cc:df:9e:5c:1d:34:f5:60:cf:53:cd:49:01:
     95:11:55:17:ef
  publicExponent: 65537 (0x10001)
  privateExponent:
     00:af:bc:25:18:ca:27:ab:2c:02:38:48:1b:02:df:
     d4:20:20:0a:4d:63:ac:ab:eb:50:5b:68:0d:50:a8:
     ca:e2:1b:e3:b8:aa:41:aa:7c:5a:3e:d5:1d:82:84:
     4b:d6:ea:a3:d9:0d:18:7a:d1:4d:3d:7c:65:63:18:
     2e:fd:8b:eb:d1
  prime1:
     00:f1:89:83:42:b2:38:e6:4c:f7:1f:a7:96:76:f4:
     6b:ba:33:f6:b3:ac:7f:c4:cc:28:90:78:d7:ac:76:
     1b:09:b7
  prime2:
     00:e2:d4:0f:1a:fc:63:a5:48:92:3e:be:9c:2d:71:
     17:f5:d2:aa:7a:26:58:b7:03:ab:8c:bb:da:6b:09:
     3e:43:89
  exponent1:
     3f:3c:67:57:20:dd:f0:bd:99:bd:79:dc:d4:cb:ed:
     20:54:d6:73:f7:e7:83:98:87:ce:3b:35:0b:fb:e7:
     dc:45
  exponent2:
     1e:8a:5e:de:4b:4d:3f:5b:de:15:04:a5:12:99:3f:
     98:a1:9c:c2:85:97:3c:4d:0a:34:10:b6:ff:e2:66:
     b7:c1
  coefficient:
     76:a4:63:4d:e8:af:b3:b1:ac:81:15:13:6f:10:eb:
```

```
      82:f9:c6:6a:b0:c6:b5:39:2e:9b:35:0a:8d:c7:38:
      7d:d1
-----BEGIN PRIVATE KEY-----
MIIBVAIBADANBgkqhkiG9w0BAQEFAASCAT4wggE6AgEAAkEA1gN6AhlbcPudqfT
M
bwE1UkiEsKqxPFyrHTSVPb36ymTtZ4miM4ODLx/BLp7UE8zfnlwdNPVgz1PNSQGV
EVUX7wIDAQABAkEAr7wlGMonqywCOEgbAt/UICAKTWOsq+tQW2gNUKjK4hvju
KpB
qnxaPtUdgoRL1uqj2Q0YetFNPXxlYxgu/Yvr0QIhAPGJg0KyOOZM9x+nlnb0a7oz
9rOsf8TMKJB416x2Gwm3AiEA4tQPGvxjpUiSPr6cLXEX9dKqeiZYtwOrjLvaawk+
Q4kCID88Z1cg3fC9mb153NTL7SBU1nP354OYh847NQv759xFAiAeil7eS00/W94V
BKUSmT+YoZzChZc8TQo0ELb/4ma3wQIgdqRjTeivs7GsgRUTbxDrgvnGarDGtTku
mzUKjcc4fdE=
-----END PRIVATE KEY-----
[root@localhost security lab]#
```

SECURE DATA TRANSMISSION USING GNUPG

Aim:

To do 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
```

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.

HONEYPOT SETUP AND MONITOR

Aim:

To setup a honeypot using honeyd in Linux machine and test from windows machine.

Algorithm:

1. Install honeyd on one of the system.
2. Create honeyd configuration file.
3. Launch honeyd with options -d and -f after configuration files are created.
4. Ping from windows 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.

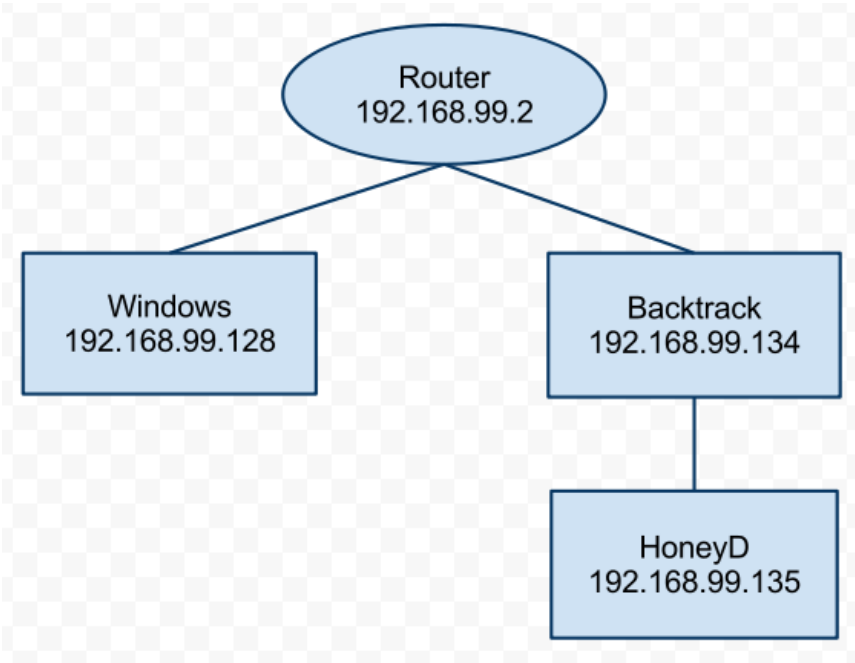

Fig. A Virtual Honeypot

Output:

```
[root@localhost security lab]# dnf install honeyd
[root@localhost security lab]# cd /etc/
[root@localhost security lab]# vi honeyd.conf
```

```
        create default
        set default default tcp action block
        set default default udp action block
        set default default icmp action block

        create windows
        set windows personality "Microsoft Windows XP Professional SP1"
        set windows default tcp action reset
        add windows tcp port 135 open
        add windows tcp port 139 open
        add windows tcp port 445 open

        set windows ethernet "00:00:24:ab:8c:12"
        dhcp windows on eth0
[root@localhost security lab]# honeyd -d -f honeyd.conf
[root@localhost security lab]# nmap -p 135,139,445,1337 192.168.99.135
        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/tcp  open   microsoft-ds
        1337/tcp closed waste
        MAC Address: 00:00:24:26:C4:ED (Connect AS)

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

STUDY OF ROOTKITS

Aim:

To study rootkits and various software to scan for it.

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.

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.

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-
   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)

VIVA QUESTIONS

1. What is zero-day attacks?
2. What are rootkits?
3. What is a virus?
4. What is digital signature?
5. What is WEP and WPA?
6. What are Honeypots?
7. What are the types of Intrusion Detection System?
8. How wireless audit is done using Kismet?
9. What is message digest code?
10. How keys are exchanged in Diffie-Hellman technique?
11. Comparison of SHA1 and MD5.
12. What is playfair cipher?
13. What is Hill cipher?
14. What is Vigenere cipher?
15. What is Affine cipher?
16. What is Rail-Fence technique?
17. What is Authentication?
18. What is Authorization?
19. Compare public key and private key cryptosystem.
20. What are rules and policies?
21. What is access control?
22. What is DAC and RBAC?
23. What is avalanche affect?
24. What is confusion and diffusion?
25. What is SSH?
26. What is IPS?
27. What are the types of attacks?
29. What is a worm?
30. What is malware?
31. What is antivirus?
32. What are firewalls?