Ex.NO: 7. Secure key exchange. 05/02/25

Aim:- TO securely exchange the crypto graphic keys over Internet to implement Diffie-Hellman key exchange mechanism.

Algorithm:-

1. Input:-

   p: prime number, g: A primitive root of p.

2. Initialize Classes:

   - class A: Represents Alice and Bob.
   - _init_: Generate a random private number n.
   - calculate and return public value. g^n % p.
   - compute shared secret (gb^n)%. using. gb.
   - class B: Represents Eve do the Same as above.

3. Create Instances:

   - create an instance of A for alice, B for Eve, also A for Bob.
   - private numbers selected by Alice, Bob & Eve are printed.
   - public values are generated and printed.
   - Shared secrets are computed and printed

4. print private Numbers.

5. Generate public values:

6. print public values:

7. Compute shared secrets:

8. print shared secrets.

program:-

```
import random
class A:
det __init__ (self,p,g):
        self.p=p
        self.g=g
        self.n= random.randint (2, p-2)

det publish (self):
        return pow(self.g, self.n, self.p)
```

```
def Compute_secret (self, received_val):
    return pow (received_ val, self.n, self.p)


class B:
    def __init__ (self, P, g):
        self. p=P
        self.g=g
        self. a= random.randint (2, P-2)
        self.b= random.randint (2, P-2)

    def publish(self):
        return pow (self.g self.a, self.p), pow (self.g, self.b; self.p)

    def Compute_secret(self, rec_val, Private_val):
        return pow (rec_val, private_val, self.p)

alice = A(P,g)
bob   = A(P g)
eve   = B(P,g)

print ("\n private Numbers:")
print (f"Alice's private number : {alice.n}")
print (f"Bob's private number: {bob.n}")
print (f"Eve's private numbers: a={eve.a}, b={eve.b}")
- - - - -
```

Output:-

```
private numbers:                          public valuer:
  Alice's private number: 6                 Alice's public value (ga): 8
  Bob's private number: 9                   Bob's public value (gb): 11
  Eve's private number: a=7, b=4            Eve's public valuer (gea, geb): 17,4
```

sult:- Thus, implementation of secure key exchange has been implemented successfully.

EX NO. 8 Digital Signature Generation 12/02/25

Aim: To authenticate a message sent over the Internet using digital. Signature mechanism.

Algorithm :-

1) Generate RSA keys:
- The RSA key pair (private and public keys) is generated with a keysize of 2048 bits.
- The keys are saved to files private.pem and public.pem.

2) Sign Message function:
- imports the private key.
- Creates a SHA-256 hash of the message.
- signs the hash using pkcs1-15 with the private key.
- Returns the signature.

3) verify signature function::
- imports a public key.
- Creates a SHA-256 hash verifies the signature using Pkcs1-15 with public key.
- Return True if valid else false.

program:

```
Const crypto = require("crypto");
const fs = require("fs");
// 1. Generate RSA key pair.
function generatekeys() {.
    Const { Publickey, privatekey} = crypto. generatekeypairsync ("rsa", {.
                    moduluslength: 2048, PublickeyEncoding: {type: "pkcs1",
                    format: "pem"}, Private KeyEncoding: {type: "pkcs1",
                    format: "pem"}, });

  fs.writeFileSync ("Private.pem", privatekey);
  fs.writeFileSync("public.pem", publickey);
```

```
//2. sign Message.
function signMessage (msg){
      Const privatekey= fs.readFileSync ("private.pem", "utf8");
      Const sign= crypto.createSign ("SHA256");
      Sign.update (msg);
      sign.end();
      return sign.sign(private key, "base64");
   }.
//3. verify signature.
function verifySignature (msg, sign){
      //publickey from public.pem
      crypto.create Verify ("SHA256");
      verify.update(msg);
      Verify.end();
      return verify.verify(publickey, sign, "base64");

//usage.
  generatekeys();
  const msg = "Hello, RSA!";
  const sig = signMessage (msg);
  console.log ("Signature valid:", verifySignature (msg, sig));
```

Output:-

```
Signature valid: true.
```

Result:-

Hence, Digital Signature Generation and Verification has Completed successfully.

EXNO: 9. Implementation of Mobile security. 12/02/25.

Aim:- To implement basic mobile security functionalities such as scanning for known malicious apps, encrypting and decrypting sensitive data, monitoring network traffic and authenticating users.

Algorithm:-

1) import required libraries like hashlib, os, socket, ssl, base64 and fernet from cryptography. fernet.

2) Define known malicious App hashes

3) Detect malicious Apps, for each app in app-list: compute MD5 hash, if hash exists add app to malicious_apps and return them.

4) Use Fernet. generate_key() to create a Symmetric encryption key.

5) Input data (plaintext), key (encryption key) to encrypt the data.

6) Use Fernet (key). decrypt (encrypted_data). ~~decode~~ decode () to decrypt data.

7) Monitor Network Traffic.

8) Establish secure Connection:
   Input: host (IP/domain), port (Port number), Create an SSL context
   Establish TCP connection using socket. create_connection(),
   wrap the connection, print negotiated SSL/TLS version.

9) Authenticate user,
   - Input: username, password, stored_hash.
   - Compute SHA-256 hash of password, compare with stored_key.
   - Return true if they match, otherwise False.

program:

```
from Crypto.Cipher import AES
from crypto. Random import get_random_bytes
key = get_random_bytes (16)
iv = get_random_bytes(16)
cipher = AES. new(key, AES.MODE_CBC, iv).
```

```
message = b"Hello, World!"
while len(message) % 16!=0:
      message += b"\x00"

encrypted_msg = cipher.encrypt(message)
decrypted_msg = cipher.decrypt(encrypted_msg)

print("Key:", key.hex())
print("IV:", iv.hex())
print("Message:", message)
print("Encrypted Message:", encrypted_msg.hex())
print("Decrypted Message:", decrypted_msg)
```

Output:-

```
Key: 234567890abc - - - -
IV: fedcba9876543 - - - -
Messge: b'Hello, World! \x00'
Encrypted Message: 7890 abc - - - -
Decrypted Message: b'Hello, World! \x00'
```

sult:-

Thus, Implementation of mobile security has Completed successfully.

Ex-10. Intrusion Detection/Prevention ~~using~~ System with Snort Algorithm.

im:-

To Configure and monitor traffic detect Intnesion attempt log them and report when an Intrusion attempt is detected.

gorithm:-

1. Install Snort.
   1.1 update your system
   ```
   bash
   Sudo apt-get update.
   ```
   1.2 install necessary dependencies:
   ```
   Sudo apt-get Install -y buid-essential libcap-dev.
   lib re3 -bev lib .dumbet -deve. dison flex.
   ```
   1.3 Download and Install snort.
   1.4 verify the Snort Installation.
2. Capture network traffic: use scapy to sniff network packets.
3. Define Rules: use predefined attack signature.
4. Andyze packets: Match packets aganist rules and log alerts.
5. output Alerts: print (or) log detekted Intrusions.

gram:.

```
From scapy. all Import sniff, IP, TCP, UDP.
Import re.
RULES = [
    {"pattern": "Nmap", "msg": "NmapScan Detected'},
    {"pattern": "malicious-payload"; "msg": "Potential Attack Detected"}.
    ].
def packet_callback (packet):
    if packet.hashlayer (ip):
        src_ip= packet[IP].src.
        dst_ip= packet [IP].dst.
        protocol = "TCP" if.
```

```
packet.har layer (TCP) else "UDP" if .
packet. har layer (udP) else "other"
    ~~srcip=p~~
    payload = bytes (packet[TCP].payload) if .
     packet, har layer [TCP] else.
          for rule in ruler:
    if
     re.search (rule ["pattern"].encode())
     payload re. IGNORE CASE();
     print(f"(ALERT) { true['msg'] from {src-ip} to {dst-ip} using
                                                       {protocol}")
    print ("Starting Intrusion detection System.. ")
    snitt ( prn = packet_callback , store = False , filter = "ip" ; count-10)
```

Input :-
packet = IP (src = "192.168.1.5", dst = "192.168.1.10")
TCP (dport=80) (Row Load = "Nmap")

output:-
(ALERT) NMap Scan detected from 192.1681.5 to 192.168.1.10 using TCP.

Result:-
Program for Intrusion Detection/Prevention System with snort Algorithm has executed Succesfully.

Ex-11 Defeating Malware - Building Trojans

:- To build a Trojan and know the harmness of the Trojan malware in a computer system.

rithm:-

1. create a simple Trojan by using windows batch file (.bat).
2. Type these below code in notepad and save it as Trojan.
3. Double click on Trojan.bat file.
4. when the Trojan code executer, It will happen, open ms-paint, notepad, command-prompt, Explorer, etc Infinitly.
5. Restart the computer to step the execution of this Trojan.

cedure:-.

1. Monitor Running processes - Identify unknown or suspicious processes.
2. Analyze network Activity - Detect unusual out bound traffic
3. check file system changes - look for unauthorized modification
4. scan for known Signatures - compare with a malware data base.
5. Isolate and Block - ~~Testimate~~ Terminate suspicious processes and restrict network Accerr.

gram:-

```
import psutil.
import socket
MALICIOUS_PATTERNS = {"Trojan", "malware", "rat", "key logger"}

def detect_suspicious_processes():
  for processes in psutil.process_iter({'pid', 'name', 'connections'}):
  try:
   process_name = processes.info['name'].lower()
  if any(pattern in process_name for pattern in MALICIOUS_PATTERNS):
```

```
print (f"[ALERT] suspicious process detected: {process_name} PID: {process.into
        {'pid')})").
for  conn in process.into['connections'] or [];
 if conn.status = = psutil.CONN_ESTABLISHED:
   remote_ip = conn.raddes.ip.
print (f"[ALERT] procew {process_name} communications with {remote_ip}
      (PID: {process_into['pid']})").
Except (psutil. No such Process, psutil. AccessDenied . psutil.Zombie.
      process);
print ("Scanning for suspicious process...").
detect_suspicious_process().
```

nput:-

Running a process named Trojan exe detecting a process that opens an unusual network connection

put:-

```
[ALERT]. suspicious process detected:
      trojan_exe (PID: 3456)
                192.168.1.100   (PID: 3456).
```

ult:-

program for defeating malware - building trogans has completed and executed successfully.

Ex-12. **Defeating Malware Rootkit Hunter.**

m: TO install a rootkit hunter and find the malwares in a computer.

1. using. Rootkit Hunter on Linux:
   - Install rkhunter using the package manager.
   - update the rkhunter database.
   - Scan the System for rootkits and malware.
   - Analyze the Scan results and take necessary action.

2. Using . GMER Rootkit Tool on window.
   - Download GMER from its official website.
   - Run the executable file with a random name to prevent rootkit interference.
   - click the 'scan' button and wait for the process to complete.
   - Review the Scan result and detect or disable detected rootkits.
   - Restart the System and perform a re-scan to confirm removed.

Steps to execute:-

For rkhunters (Linux);

1. Install rkhunters:

```
sudo apt-get Install rkhunters (for Debian-based os)
```

2. update. rk hunters database.

```
Sudo rkhunter --- update.
```

3. scan the system

```
sudo rkhunter -- check.
```

4. Review warnings and logs.

```
Sudo . cat /var/log/rkhunter-log.
```

ult:-

Thus the ·system was scanned for rootkit and necessary actions were taken to remove any defected threats was successful.

Ex-13 Implement Database Security.

Aim :- To implement database security using Access Control and Authentication in microsoft SQL solves on windows os.

Algorithm :-

1. Install microsoft SQL solver and SQL solver management studio (ssms)
2. create a secure database and define user roles with permissions.
3. Enforce strong password policies and Authentication methods.
4. Test Access Control by verifying user permissions.
5. Enable Audit logging to monitor database Access.

step-by-step

1. Install SQL solves & ssms from the official microsoft website.
2. create a database and user roles in ssms.
3. Enforce password policies

```
    ALERT LOGIN 'admin-uses with
    check-policy = ON;
    ALERT LOGIN read-only-user with
        check-policy = ON;
```

4. Enable SQL solver Authentication

   In ssms got security > logins > New login and choose windows Authentication.

5. Test security.

   'Login in used only user and attempt write operations

   [denied].

output :

- Database uses created successfully.
- Access Control verified authorized modifications are restricted.
- security policies applied to protect data.

Result :

Thus, Implementation of Data base security was executed user. verified. successfully.

Ex-14 Implement Encryption and Integrity Control in Database security

Aim.

TO implement Encryption and Integrity control in SQL server to pr protect Sensitive data and Ensure it Accuracy.

Algorithm

1) Install SQL server and SQL Server management studio.

2) Implement Transparent data Encryption for database Encryption.

3) use column_level Encryption. for sensitive data.

4) Apply data Integrity control using hash functions.

5) verity Encryption and Integrity using SQL queries.

Procedure

1) create master key, certificate.

```
CREATE MASTER KEY ENCRYPTION BY Password = "Strong passwords".
```

2) Enable Transparent Data Encryption (TDE)

```
ALTER DATABASE SECURE DB SET ENCRYPTION;
```

3) verity Encryption.

```
SELECT name, is_Encryption from sys-database WHERE
 name = "Secure DB".
```

4) Encrypt Column Data.

```
INSERT INTO Sensitive Data VALUES: ENCRYPTION BY KEY (KEY-GUID)
('Symmetric-key'), sensitive into'));
```

5) verity Data Integrity.

```
IF (SELECT Data Hash FROM: Data Integrity WHERE ID=1)=
Hash BYTES ('SHA 256 'import Data) print 'Data Integrity-verified';
```

output:-

Encryption Enabled (is_encrypted=1)

sensitive Data Encrypted and decrypted successfully Data Integrity. verified.

Result:-

Thus the implementation of Encryption and Integrity Control in Database security is successfully completed.