Challenge: Android in the Middle

Challenge Description :

Years have passed since Miyuki rescued you from the graveyard. When Virgil tells you that he needs your help with something he found there, desperate thoughts about your father and the disabilities you developed due to the disposal process come to mind. The device looks like an advanced GPS with AI capabilities. Riddled with questions about the past, you are pessimistic that you could be of any value. After hours of fiddling and observing the power traces of this strange device, you and Virgil manage to connect to the debugging interface and write an interpreter to control the signals. The protocol looks familiar to you. Your father always talked about implementing this scheme in devices for security reasons. Could it have been him?.

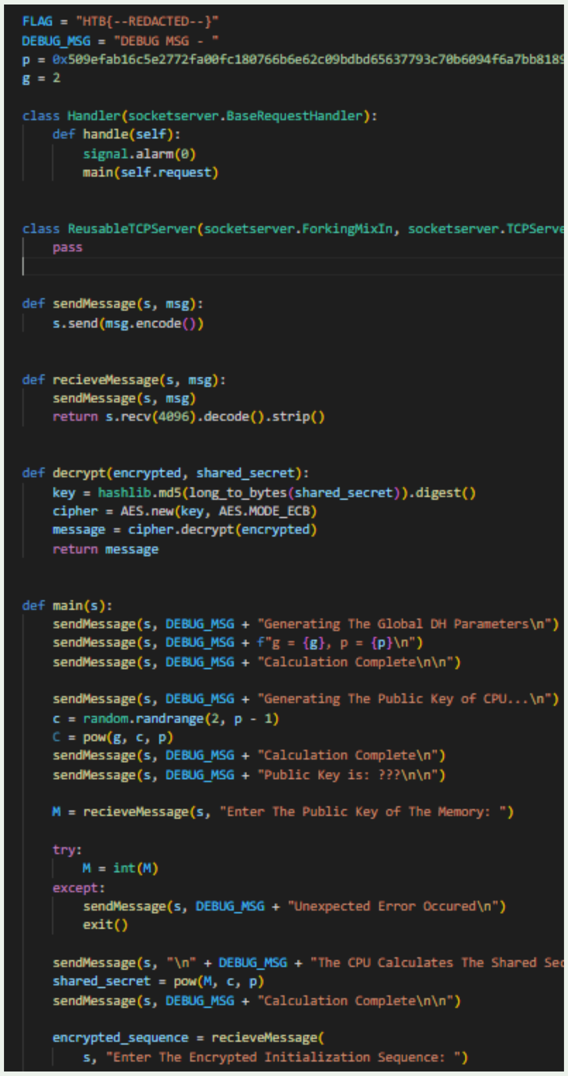
Context :

- We are given a python script with a flawed implementation of a Diffie-Hellman-like key exchange mechanism to retrieve a hidden flag from a remote server.

Flag :

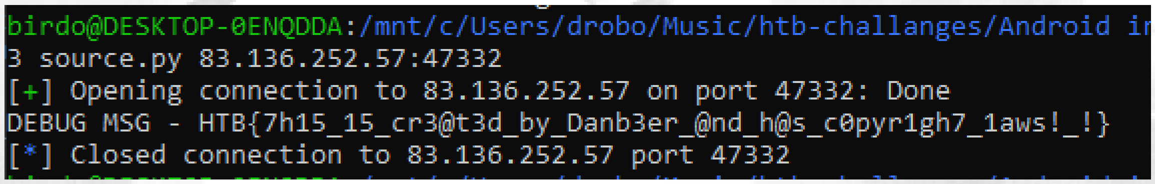
- First examine the provided Python script used for interacting with the challenge. The server prompts the user to input a public key MMM, which is subsequently utilized to compute a shared secret. This shared secret plays a critical role in decrypting an encrypted message, and our primary goal is to ensure that the decryption yields a specific plaintext: "Initialization Sequence - Code 0".

- The strategy involves exploiting our control over the public key MMM to manipulate the shared secret computation, thereby exploiting the vulnerability in the key exchange protocol to retrieve the flag from the server.



```python
FLAG = "HTB{--REDACTED--}"
DEBUG_MSG = "DEBUG MSG - "
p = 0x509efab16c5e2772fa00fc180766b6e62c09bdbd65637793c70b6094f6a7bb818
g = 2

class Handler(socketserver.BaseRequestHandler):
    def handle(self):
        signal.alarm(0)
        main(self.request)


class ReusableTCPServer(socketserver.ForkingMixIn, socketserver.TCPServe
    pass


def sendMessage(s, msg):
    s.send(msg.encode())


def recieveMessage(s, msg):
    sendMessage(s, msg)
    return s.recv(4096).decode().strip()


def decrypt(encrypted, shared_secret):
    key = hashlib.md5(long_to_bytes(shared_secret)).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    message = cipher.decrypt(encrypted)
    return message


def main(s):
    sendMessage(s, DEBUG_MSG + "Generating The Global DH Parameters\n")
    sendMessage(s, DEBUG_MSG + f"g = {g}, p = {p}\n")
    sendMessage(s, DEBUG_MSG + "Calculation Complete\n\n")

    sendMessage(s, DEBUG_MSG + "Generating The Public Key of CPU...\n")
    c = random.randrange(2, p - 1)
    C = pow(g, c, p)
    sendMessage(s, DEBUG_MSG + "Calculation Complete\n")
    sendMessage(s, DEBUG_MSG + "Public Key is: ???\n\n")

    M = recieveMessage(s, "Enter The Public Key of The Memory: ")

    try:
        M = int(M)
    except:
        sendMessage(s, DEBUG_MSG + "Unexpected Error Occured\n")
        exit()

    sendMessage(s, "\n" + DEBUG_MSG + "The CPU Calculates The Shared Se
    shared_secret = pow(M, c, p)
    sendMessage(s, DEBUG_MSG + "Calculation Complete\n\n")

    encrypted_sequence = recieveMessage(
        s, "Enter The Encrypted Initialization Sequence: ")
```

- The script initiates by reading messages from a file named messages.txt and stores them in a variable named MSG. Following this, the main function of the script is executed to begin the interaction with the server.
- To summarize, the script interacts with the server by first reading message inputs, then executing a key exchange process involving user-provided public keys MMM. By exploiting the flaw in the key exchange protocol, we strategically manipulate MMM to control the shared secret computation, ensuring the decryption of a specific message that leads to the retrieval of the flag from the server. This approach effectively demonstrates the exploitation of cryptographic vulnerabilities through careful manipulation of key exchange parameters.
- The link for the script to complete this is here : [Link]



```
birdo@DESKTOP-0ENQDDA:/mnt/c/Users/drobo/Music/htb-challanges/Android ir
3 source.py 83.136.252.57:47332
[+] Opening connection to 83.136.252.57 on port 47332: Done
DEBUG MSG - HTB{7h15_15_cr3@t3d_by_Danb3er_@nd_h@s_c0pyr1gh7_1aws!_!}
[*] Closed connection to 83.136.252.57 port 47332
```

- The Flag : HTB{7h15_15_cr3@t3d_by_Danb3er_@nd_h@s_c0pyr1gh7_1aws!_!}