```python
import socket
import argparse

def start_client(DATE, HOST, PORT, verbose):
    """
    This function is the main function in order to setup the client. The function contains many sub-functions used for finding
    parameters nessasary to display the desired output, the body of this function contains the steps to produce a client and
    calls the nessasary sub-functions to send a request to the server.
    """
    try:
        IP = socket.gethostbyname(HOST)
    except socket.gaierror:
        print(f"Hostname not found: Could not connect to ({HOST}:{PORT})")
        return -1

    def checkInputs(DATE, IP, PORT):
        """ Checks the intergrity of inputs and if they comply to the specifications """
        if not IP:
            print("The Hostname is invalid")
            return False
        if PORT < 1024 or PORT > 64000:
            print("The Port number is not within specified range (1024:64000)")
            return False
        if DATE != 'date' and DATE != 'time':
            print("MSG parameter must be set to `date` or `time`")
            return False
        return True

    def decrypt_message(packet):
        """
        Takes in a packet in the form of a byte array, and decrypts it to pull the relevent data
        that will later be displayed to the client.

        Note: This has a --verbose flag to get the full contents of the packet.
        """
        info = [packet[i:i+1] for i in range(0, len(packet), 1)]
        if len(info) < 13:
            print("Packet does not include minimum headersize")
            return -1

        MagicNo = int.from_bytes(info[0] + info[1], 'big')

        if MagicNo != 0x497E:
            print("MagicNo is incorrect: `{}` recieved, must equal `0x497E`".format(MagicNo))
            return -1
        PacketType = int.from_bytes(info[2] + info[3], 'big')
        if PacketType != 0x0002:
            print("PacketType is incorrect: `{}` received, must equal `0x0002`".format(PacketType))
        LanguageCode = int.from_bytes(info[4] + info[5], 'big')
        if LanguageCode < 0x0001 or LanguageCode > 0x0003:
            print("LanguageCode is incorrect: `{}` received, must be within range (1, 3)".format(LanguageCode))
            return -1
        Year = int.from_bytes(info[6] + info[7], 'big')
        if Year > 2100:
            print("Year is incorrect: `{}` received, must be below 2100".format(Year))
            return -1
        Month = int.from_bytes(info[8], 'big')
        if Month < 1 or Month > 12:
            print("Month is incorrect: `{}` received, must be between 1 and 12".format(Month))
            return -1
        Day = int.from_bytes(info[9], 'big')
        if Day < 1 or Day > 31:
            print("Day is incorrect: `{}` received, must be between 1 and 31".format(Day))
            return -1
        Hour = int.from_bytes(info[10], 'big')
    if Hour < 0 or Hour > 23:
        print("Hour is incorrect: `{}` received, must be within range (0, 23)".format(Hour))
        return -1
    Minute = int.from_bytes(info[11], 'big')
    if Minute < 0 or Minute > 59:
        print("Minute is incorrect: `{}` received, must be within range (0, 59)".format(Minute))
        return -1
    Length = int.from_bytes(info[12], 'big')
    text = bytearray()
    for i in range(13, len(info)):
        text += info[i]
    text = text.decode('utf-8')

    if len(info) != 13 + Length:
        print("Length of packet does not match packet received")
        return -1

    if verbose:
        print("---------------------------------------")
        print(f"MagicNo: {hex(MagicNo)}")
        print(f"PacketType: {hex(PacketType)}")
        print(f"LanguageCode: {hex(LanguageCode)}")
        print(f"Year: {Year}")
        print(f"Month: {Month}")
        print(f"Day: {Day}")
        print(f"Hour {Hour}")
        print(f"Minute: {Minute}")
        print(f"Length: {Length}")
        print(f"Text: {text}")
        print("---------------------------------------")
        print("")

    return text

def format_request(Date):
    """
    Formats the packet into a byte array to send to the server.
    """
    MagicNo = 0x497E
    PacketType = 0x0001
    if Date == 'date':
        RequestType = 0x0001
    elif Date == 'time':
        RequestType = 0x0002
    else:
        return -1
    bytelist = [MagicNo.to_bytes(2, 'big'), PacketType.to_bytes(2, 'big'), RequestType.to_bytes(2, 'big')]

    arrayBytes = bytearray()

    for x in bytelist:
        arrayBytes += x

    return arrayBytes

if checkInputs(DATE, IP, PORT):
    """ Checks if the input date/time is valid """
    request_packet = format_request(DATE)
    if request_packet == -1:
        print("The `date` parameter must be set to either `date` or `time`")
        return -1
    else:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.settimeout(1)
        s.sendto(request_packet, (IP, PORT))
        complete_message = bytearray()

    while True:
        try:
            msg, source = s.recvfrom(1024)

            if len(msg) <= 0:
                break

            complete_message += msg

        except socket.timeout:
            print(f"Client timeout: Could not connect to ({HOST}:{PORT})")
            break

        except socket.error:
            print(f"Client timeout: Could not connect to ({HOST}:{PORT})")
            break

        result = decrypt_message(complete_message)
        if result != -1:
            print(result)
            break
        s.close()
        return result


def Main():
    parser = argparse.ArgumentParser()
    parser.add_argument("MSG", help="The message to receive from server must be `date` or `time`", type=str)
    parser.add_argument("HOST", help="The Hostname to connect to", type=str)
    parser.add_argument("PORT", help="The Port number to connect to", type=int)
    parser.add_argument("-v", "--verbose", action="store_true", help="verbose output: full output of packet recieved")

    args = parser.parse_args()

    if args.verbose:
        start_client(args.MSG, args.HOST, args.PORT, verbose=True)
    else:
        start_client(args.MSG, args.HOST, args.PORT, verbose=False)


if __name__ == "__main__":
    Main()
```