PREPARED BY
PHANTOM SRL

REPORT ASSEMBLY

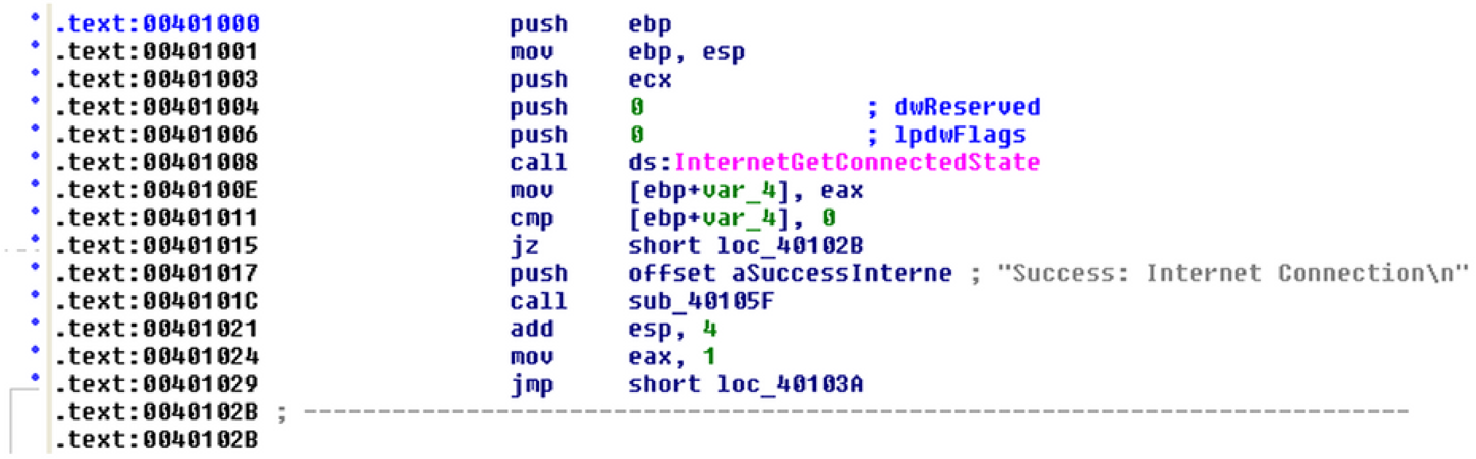
PIGNATELLO GIUSEPPE
IANNONE LUCA
ALESSIO D’OTTAVIO
FRANCESCO PIO SCOPECE

PHANTOM s.r.l
IMPOSSIBLE IS
OUR TARGET

CONTENTS

EXERCISE TRACE

THE FOLLOWING FIGURE SHOWS AN EXCERPT OF MALWARE CODE. IDENTIFY KNOWN CONSTRUCTS SEEN DURING THE THEORY LESSON.



```
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx
.text:00401004                 push    0               ; dwReserved
.text:00401006                 push    0               ; lpdwFlags
.text:00401008                 call    ds:InternetGetConnectedState
.text:0040100E                 mov     [ebp+var_4], eax
.text:00401011                 cmp     [ebp+var_4], 0
.text:00401015                 jz      short loc_40102B
.text:00401017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                 call    sub_40105F
.text:00401021                 add     esp, 4
.text:00401024                 mov     eax, 1
.text:00401029                 jmp     short loc_40103A
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
```

WHAT IS ASSEMBLY COSTRUCTION?

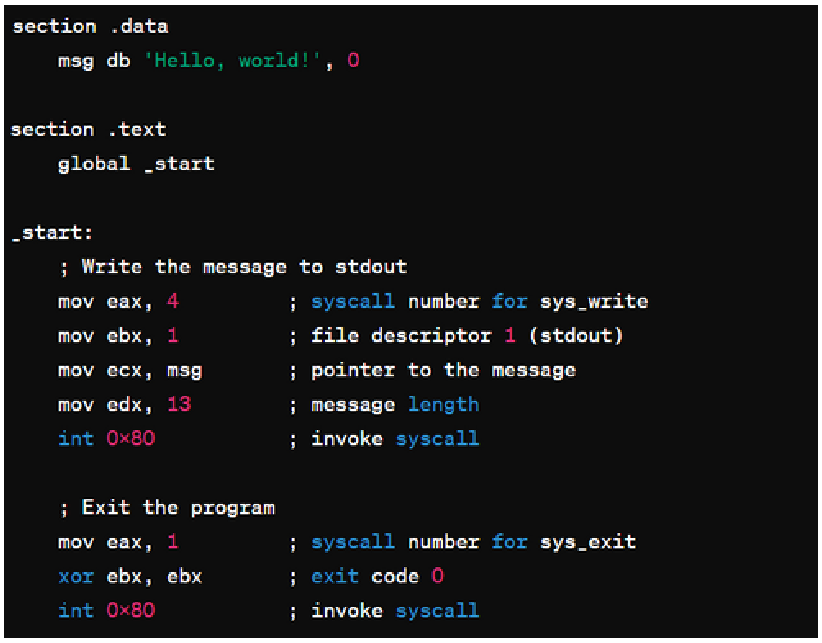
In the context of computer programming, a "construction" in assembly language typically refers to a sequence of instructions written in assembly language that perform a specific task or operation. Assembly language is a low-level programming language that closely corresponds to the machine language instructions understood by a computer's CPU (Central Processing Unit).

In assembly language, each instruction corresponds to a specific machine language instruction that the CPU can execute directly. These instructions typically involve operations such as arithmetic computations, memory access, control flow (e.g., branching), and input/output operations.

Constructing a program in assembly language involves writing these instructions in a sequential manner to achieve the desired functionality. For example, a construction in assembly language might involve:

1. Loading data from memory into registers.
2. Performing arithmetic or logical operations on the data.
3. Storing the result back into memory.
4. Branching or jumping to different parts of the program based on certain conditions.

Assembly language constructions are often used when programmers require precise control over the hardware or need to optimize performance, as assembly language instructions directly map to machine code and can be more efficient than higher-level languages for certain tasks. However, programming in assembly language can be more complex and error-prone compared to higher-level languages due to its low-level nature and lack of abstraction.



```
section .data
    msg db 'Hello, world!', 0

section .text
    global _start

_start:
    ; Write the message to stdout
    mov eax, 4          ; syscall number for sys_write
    mov ebx, 1          ; file descriptor 1 (stdout)
    mov ecx, msg        ; pointer to the message
    mov edx, 13         ; message length
    int 0x80            ; invoke syscall

    ; Exit the program
    mov eax, 1          ; syscall number for sys_exit
    xor ebx, ebx        ; exit code 0
    int 0x80            ; invoke syscall
```

LINES OF CODE EXPLAINED

The provided code snippet appears to be assembly language instructions. It seems to be a snippet from a program that checks for an internet connection. Here's a breakdown of what each line does:

- `.text: 00401000 push ebp`: Pushes the value of the base pointer onto the stack.
- `.text: 00401001 mov ebp, esp`: Moves the value of the stack pointer into the base pointer.
- `.text: 00401003 push ecx`: Pushes the value of the ECX register onto the stack.
- `.text: 00401004 push 0  ; dwReserved`: Pushes the value 0 onto the stack (dwReserved parameter).
- `.text: 00401006 push 0  ; lpdwFlags`: Pushes the value 0 onto the stack (lpdwFlags parameter).
- `.text: 00401008 call ds: InternetGetConnectedState`: Calls the `InternetGetConnectedState` function to check if the computer is connected to the internet.
- `.text: 0040100E mov [ebp+var_4], eax`: Moves the result of the `InternetGetConnectedState` function call into a variable located at `[ebp+var_4]`.
- `.text: 00401011 cmp [ebp+var_4], 0`: Compares the value stored in `[ebp+var_4]` with 0.
- `.text: 00401015 jz  short loc 40102B`: Jumps to the specified location if the previous comparison result was zero (indicating no internet connection).
- `.text: 00401017 push offset aSuccessInterne; "Success: Internet Connection\n"`: Pushes the address of the string "Success: Internet Connection\n" onto the stack.
- `.text: 0040101C call sub_40105F`: Calls a subroutine (`sub_40105F`) to handle the success case.
- `.text: 00401021 add  esp, 4`: Adjusts the stack pointer to deallocate the parameters pushed earlier.
- `.text: 00401024 mov  eax, 1`: Moves the value 1 into the EAX register.
- `.text: 00401029 jmp  short loc_40103A`: Jumps to the specified location.

Overall, this code snippet is checking for an internet connection using the InternetGetConnectedState function and then performs different actions based on whether the connection is successful or not. If the connection is successful, it calls a subroutine to handle the success case; otherwise, it continues with other instructions.

THIS IS THE CODE TRANSCRIBED IN C++:

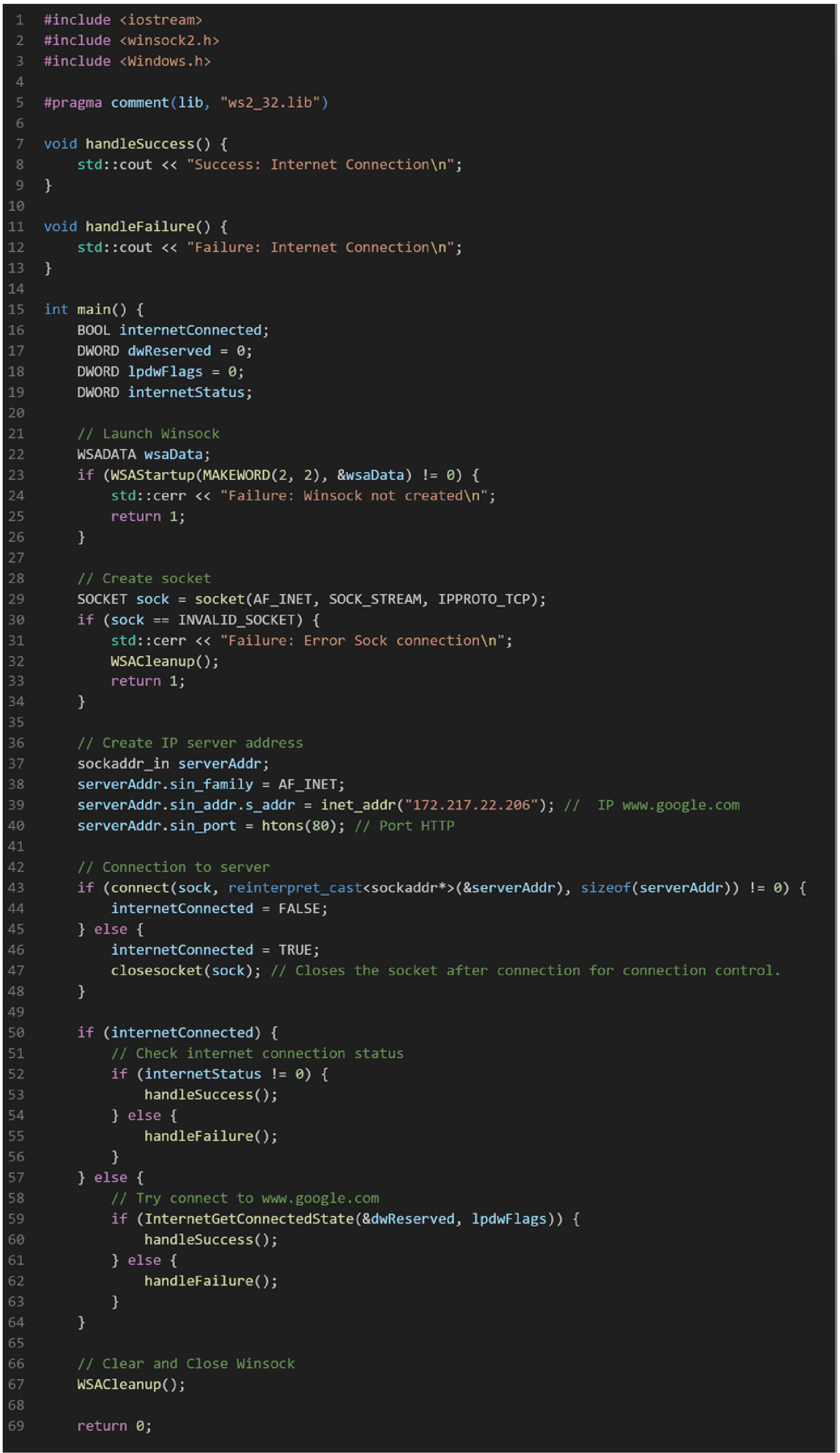
```cpp
#include <iostream>
#include <Windows.h>

int main() {
    BOOL internetConnected;
    DWORD dwReserved = 0;
    DWORD lpdwFlags = 0;

    // Check internet connection
    internetConnected = InternetGetConnectedState(&dwReserved, lpdwFlags);

    if (internetConnected) {
        std::cout << "Success: Internet Connection\n";
        // Call a function to handle the success case
        handleSuccess();
    }
    else {
        // Handle case when there's no internet connection
    }

    return 1;
}
```

ONE USES THE INTERNETGETCONNECTEDSTATE FUNCTION PROVIDED BY WINDOWS.H TO CHECK THE INTERNET CONNECTION. IF THE CONNECTION IS ACTIVE, A SUCCESS MESSAGE IS PRINTED, AND THE HANDLESUCCESS() FUNCTION IS CALLED TO HANDLE THE SUCCESS CASE. OTHERWISE, CODE CAN BE ADDED TO HANDLE THE CASE WHERE THERE IS NO INTERNET CONNECTION.

PHANTOM s.r.l
IMPOSSIBLE IS OUR TARGET

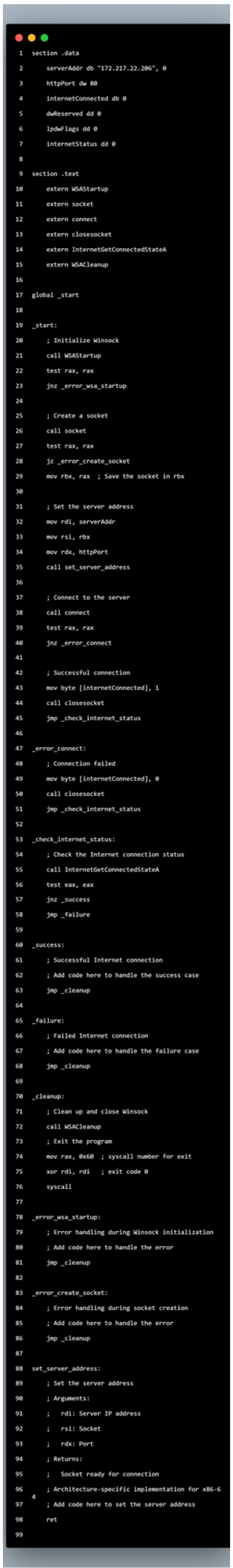
IMPLEMENTATION OF THE CODE

C++



```cpp
#include <iostream>
#include <winsock2.h>
#include <Windows.h>

#pragma comment(lib, "ws2_32.lib")

void handleSuccess() {
    std::cout << "Success: Internet Connection\n";
}

void handleFailure() {
    std::cout << "Failure: Internet Connection\n";
}

int main() {
    BOOL internetConnected;
    DWORD dwReserved = 0;
    DWORD lpdwFlags = 0;
    DWORD internetStatus;

    // Launch Winsock
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Failure: Winsock not created\n";
        return 1;
    }

    // Create socket
    SOCKET sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock == INVALID_SOCKET) {
        std::cerr << "Failure: Error Sock connection\n";
        WSACleanup();
        return 1;
    }

    // Create IP server address
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr("172.217.22.206"); //  IP www.google.com
    serverAddr.sin_port = htons(80); // Port HTTP

    // Connection to server
    if (connect(sock, reinterpret_cast<sockaddr*>(&serverAddr), sizeof(serverAddr)) != 0) {
        internetConnected = FALSE;
    } else {
        internetConnected = TRUE;
        closesocket(sock); // Closes the socket after connection for connection control.
    }

    if (internetConnected) {
        // Check internet connection status
        if (internetStatus != 0) {
            handleSuccess();
        } else {
            handleFailure();
        }
    } else {
        // Try connect to www.google.com
        if (InternetGetConnectedState(&dwReserved, lpdwFlags)) {
            handleSuccess();
        } else {
            handleFailure();
        }
    }

    // Clear and Close Winsock
    WSACleanup();

    return 0;
```

ASSEMBLY



```asm
section .data
    serverAddr db "172.217.22.206", 0
    httpPort dw 80
    internetConnected db 0
    dwReserved dd 0
    lpdwFlags dd 0
    internetStatus dd 0

section .text
    extern WSAStartup
    extern socket
    extern connect
    extern closesocket
    extern InternetGetConnectedStateA
    extern WSACleanup

global _start

_start:
    ; Initialize Winsock
    call WSAStartup
    test rax, rax
    jnz _error_wsa_startup

    ; Create a socket
    call socket
    test rax, rax
    jz _error_create_socket
    mov rbx, rax  ; Save the socket in rbx

    ; Set the server address
    mov rdi, serverAddr
    mov rsi, rbx
    mov rdx, httpPort
    call set_server_address

    ; Connect to the server
    call connect
    test rax, rax
    jnz _error_connect

    ; Successful connection
    mov byte [internetConnected], 1
    call closesocket
    jmp _check_internet_status

_error_connect:
    ; Connection failed
    mov byte [internetConnected], 0
    call closesocket
    jmp _check_internet_status

_check_internet_status:
    ; Check the Internet connection status
    call InternetGetConnectedStateA
    test eax, eax
    jnz _success
    jmp _failure

_success:
    ; Successful Internet connection
    ; Add code here to handle the success case
    jmp _cleanup

_failure:
    ; Failed Internet connection
    ; Add code here to handle the failure case
    jmp _cleanup

_cleanup:
    ; Clean up and close Winsock
    call WSACleanup
    ; Exit the program
    mov rax, 0x60  ; syscall number for exit
    xor rdi, rdi   ; exit code 0
    syscall

_error_wsa_startup:
    ; Error handling during Winsock initialization
    ; Add code here to handle the error
    jmp _cleanup

_error_create_socket:
    ; Error handling during socket creation
    ; Add code here to handle the error
    jmp _cleanup

set_server_address:
    ; Set the server address
    ; Arguments:
    ;   rdi: Server IP address
    ;   rsi: Socket
    ;   rdx: Port
    ; Returns:
    ;   Socket ready for connection
    ; Architecture-specific implementation for x86-6
4
    ; Add code here to set the server address
    ret

```

PHANTOM s.r.l
IMPOSSIBLE IS OUR TARGET

CODE DESCRIPTION

THIS C++ IMPLEMENTATION CHECKS THE INTERNET CONNECTION STATUS USING THE WINSOCK LIBRARY ON WINDOWS PLATFORMS. HERE'S WHAT THE CODE DOES:

1. INITIALIZES WINSOCK USING THE `WSASTARTUP` FUNCTION.
2. CREATES A SOCKET USING THE `SOCKET` FUNCTION.
3. SETS THE SERVER ADDRESS TO CONNECT TO (IN THIS CASE, THE IP ADDRESS OF WWW.GOOGLE.COM ON PORT 80).
4. TRIES TO CONNECT TO THE SERVER USING THE `CONNECT` FUNCTION.
5. IF THE CONNECTION SUCCEEDS, CLOSES THE SOCKET AND SETS `INTERNETCONNECTED` TO `TRUE`. OTHERWISE, SETS `INTERNETCONNECTED` TO `FALSE`.
6. IF `INTERNETCONNECTED` IS `TRUE`, CHECKS THE INTERNET CONNECTION STATUS USING THE `INTERNETGETCONNECTEDSTATE` FUNCTION. IF THE CONNECTION IS ACTIVE, CALLS THE `HANDLESUCCESS` FUNCTION; OTHERWISE, CALLS THE `HANDLEFAILURE` FUNCTION.
7. IF `INTERNETCONNECTED` IS `FALSE`, TRIES TO CONNECT DIRECTLY TO THE INTERNET USING THE `INTERNETGETCONNECTEDSTATE` FUNCTION. IF THE CONNECTION IS ACTIVE, CALLS THE `HANDLESUCCESS` FUNCTION; OTHERWISE, CALLS THE `HANDLEFAILURE` FUNCTION.
8. CLEANS UP AND CLOSES WINSOCK USING THE `WSACLEANUP` FUNCTION.

IN SUMMARY, THIS CODE CHECKS THE INTERNET CONNECTION EITHER THROUGH A DIRECT SOCKET CONNECTION TO A SPECIFIC SERVER (IN THIS CASE, GOOGLE) OR VIA THE `INTERNETGETCONNECTEDSTATE` FUNCTION. IF THE CONNECTION IS ACTIVE, IT CALLS THE `HANDLESUCCESS` FUNCTION; OTHERWISE, IT CALLS THE `HANDLEFAILURE` FUNCTION.

PREPARED BY
PHANTOM SRL

THANKS!

PIGNATELLO GIUSEPPE
IANNONE LUCA
D'OTTAVIO ALESSIO
SCOPECE FRANCESCO PIO

PHANTOM s.r.l
IMPOSSIBLE IS
OUR TARGET