

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

1) Exercise Trace

2) What is assembly construction?

3) Identification

4) **Bonus Trace:**Lines of Code Explained

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 CONSTRUCTION?

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++:

```
#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.

IMPLEMENTATION OF THE CODE

C++

```

1  #include <iostream>
2  #include <winsock2.h>
3  #include <Windows.h>
4
5  #pragma comment(lib, "ws2_32.lib")
6
7  void handleSuccess() {
8      std::cout << "Success: Internet Connection\n";
9  }
10
11 void handleFailure() {
12     std::cout << "Failure: Internet Connection\n";
13 }
14
15 int main() {
16     BOOL internetConnected;
17     DWORD dwReserved = 0;
18     DWORD lpdwFlags = 0;
19     DWORD internetStatus;
20
21     // Launch Winsock
22     WSADATA wsaData;
23     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
24         std::cerr << "Failure: Winsock not created\n";
25         return 1;
26     }
27
28     // Create socket
29     SOCKET sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
30     if (sock == INVALID_SOCKET) {
31         std::cerr << "Failure: Error Sock connection\n";
32         WSACleanup();
33         return 1;
34     }
35
36     // Create IP server address
37     sockaddr_in serverAddr;
38     serverAddr.sin_family = AF_INET;
39     serverAddr.sin_addr.s_addr = inet_addr("172.217.22.206"); // IP www.google.com
40     serverAddr.sin_port = htons(80); // Port HTTP
41
42     // Connection to server
43     if (connect(sock, reinterpret_cast<sockaddr*>(&serverAddr), sizeof(serverAddr)) != 0) {
44         internetConnected = FALSE;
45     } else {
46         internetConnected = TRUE;
47         closesocket(sock); // Closes the socket after connection for connection control.
48     }
49
50     if (internetConnected) {
51         // Check internet connection status
52         if (internetStatus != 0) {
53             handleSuccess();
54         } else {
55             handleFailure();
56         }
57     } else {
58         // Try connect to www.google.com
59         if (InternetGetConnectedState(&dwReserved, lpdwFlags)) {
60             handleSuccess();
61         } else {
62             handleFailure();
63         }
64     }
65
66     // Clear and Close Winsock
67     WSACleanup();
68
69     return 0;

```

ASSEMBLY

```

1 section .data
2     serverAddr db "172.217.22.206", 0
3     httpPort dw 80
4     internetConnected db 0
5     dwReserved dd 0
6     lpdwFlags dd 0
7     internetStatus dd 0
8
9 section .text
10    extern WSAStartup
11    extern socket
12    extern connect
13    extern closesocket
14    extern InternetGetConnectedState
15    extern WSACleanup
16
17 global _start
18
19 _start:
20     ; Initialize Winsock
21     call WSAStartup
22     test rax, rax
23     jnz _error_wsa_startup
24
25     ; Create a socket
26     call socket
27     test rax, rax
28     jr _error_create_socket
29     mov rbx, rax ; Save the socket in rbx
30
31     ; Set the server address
32     mov rdi, serverAddr
33     mov rsi, rbx
34     mov rdx, httpPort
35     call set_server_address
36
37     ; Connect to the server
38     call connect
39     test rax, rax
40     jnz _error_connect
41
42     ; Successful connection
43     mov byte [internetConnected], 1
44     call closesocket
45     jmp _check_internet_status
46
47 _error_connect:
48     ; Connection failed
49     mov byte [internetConnected], 0
50     call closesocket
51     jmp _check_internet_status
52
53 _check_internet_status:
54     ; Check the Internet connection status
55     call InternetGetConnectedState
56     test eax, eax
57     jnz _success
58     jmp _failure
59
60 _success:
61     ; Successful Internet connection
62     ; Add code here to handle the success case
63     jmp _cleanup
64
65 _failure:
66     ; Failed Internet connection
67     ; Add code here to handle the failure case
68     jmp _cleanup
69
70 _cleanup:
71     ; Clean up and close Winsock
72     call WSACleanup
73     ; Exit the program
74     mov rax, 0x60 ; syscall number for exit
75     xor rdi, rdi ; exit code 0
76     syscall
77
78 _error_wsa_startup:
79     ; Error handling during Winsock initialization
80     ; Add code here to handle the error
81     jmp _cleanup
82
83 _error_create_socket:
84     ; Error handling during socket creation
85     ; Add code here to handle the error
86     jmp _cleanup
87
88 set_server_address:
89     ; Set the server address
90     ; Arguments:
91     ; rdi: Server IP address
92     ; rsi: Socket
93     ; rdx: Port
94     ; Returns:
95     ; Socket ready for connection
96     ; Architecture-specific implementation for x86-64
97     ; Add code here to set the server address
98     ret
99

```

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 CLOSSES 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**