EPICODE

EPICODE

CYBERSECURITY COURSE

BY MAX ALDROVANDI

17/05/2024

Web-site: https://epicode.com/it

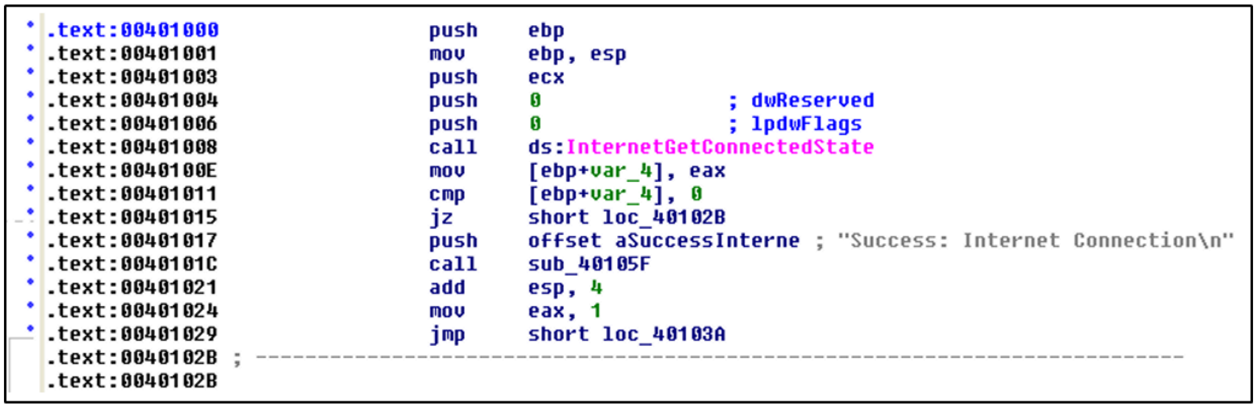
Locality:

Via dei Magazzini Generali,
6 Roma, Lazio 00154, IT

PRACTICE EXERCISE S10/L4

Track:

The following figure shows an excerpt from the code of a malware. Identify known constructs Exercise Assembly language vis ti d ur 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
```

Try to guess what functionality is implemented in the assembly code.

Hint : The function '***internetgetconnectedstate***' whether a machine has access to the Internet.

Goals:

1. Identify known constructs (e s. while, for, if, switch, etc.)

2. Hypothesize functionality - high-level execution

3. BONUS: Study and explain each line of code.

Overview:

This report provides an analysis of a given snippet of assembly code, which appears to be part of a malware program. The code is responsible for checking the presence of an Internet connection and taking specific actions based on the result.

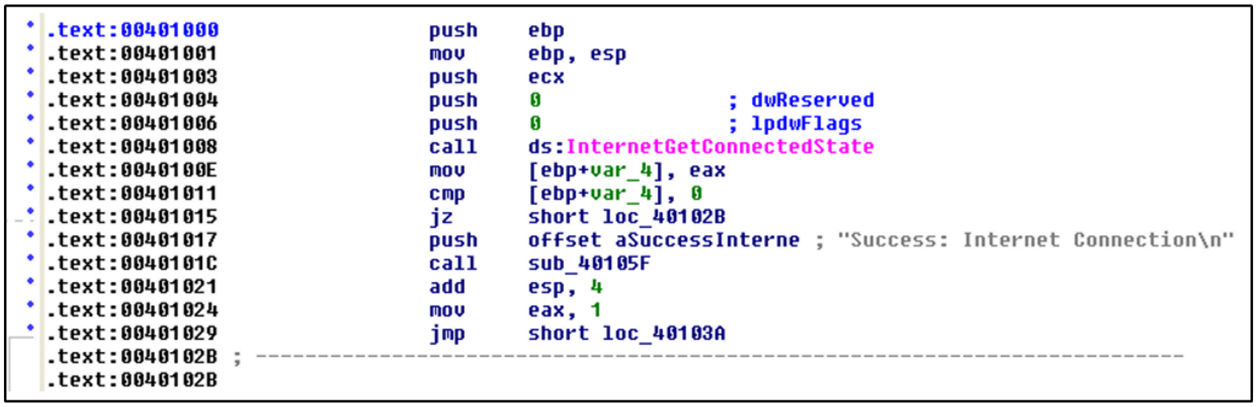
Code Analysis:

The provided code is an assembly routine that performs the following functions:

1. **Set up the stack frame**
2. **Call a system function to check Internet connectivity**
3. **Log or display a success message if connected**
4. **Handle the case where there is no Internet connection**

Here is the detailed analysis:

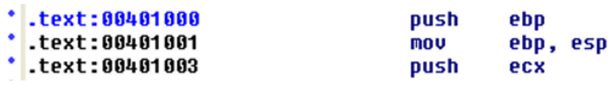
Code Breakdown:



```
.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
```

Detailed Explanation:

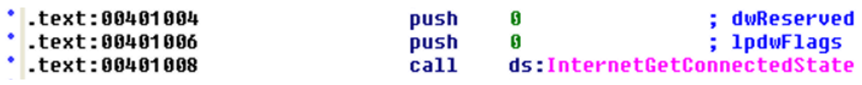
1. **Setup and Initialization**



```
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx
```

- ***push ebp***: Saves the base pointer of the previous stack frame.
- ***mov ebp, esp***: Sets the base pointer to the current stack pointer, establishing a new stack frame.
- ***push ecx***: Saves the current value of the ***ecx*** register on the stack for later use.

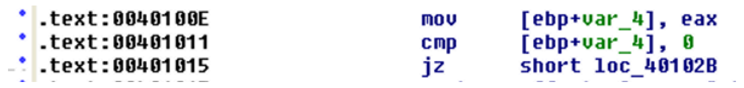
2. **Call to '*InternetGetConnectedState*'**



```
.text:00401004                 push    0               ; dwReserved
.text:00401006                 push    0               ; lpdwFlags
.text:00401008                 call    ds:InternetGetConnectedState
```

- ***push 0***: Pushes ***0*** onto the stack for the ***dwReserved*** parameter.
- ***push 0***: Pushes ***0*** onto the stack for the ***lpdwFlags*** parameter.
- ***call ds:InternetGetConnectedState***: Calls the ***InternetGetConnectedState*** function from the dynamic segment to check if the system is connected to the Internet. The result is stored in the ***eax*** register.

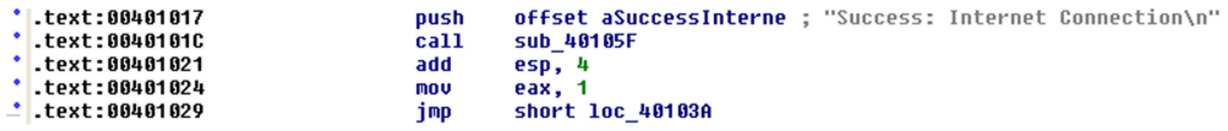
3. **Check Internet Connection State**



```
.text:0040100E                 mov     [ebp+var_4], eax
.text:00401011                 cmp     [ebp+var_4], 0
.text:00401015                 jz      short loc_40102B
```

- ***`mov [ebp+var_4], eax`***: Moves the value in ***`eax`*** (result of ***`InternetGetConnectedState`***) into a local variable ***`var_4`***.
- ***`cmp [ebp+var_4], 0`***: Compares the value of ***`var_4`*** with ***`0`***.
- ***`jz short loc_40102B`***: Jumps to ***`loc_40102B`*** if the value of ***`var_4`*** is ***`0`***, indicating no Internet connection.

4. **Log or Display Success Message**



```
.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
```

- ***push offset aSuccessInterne***: Pushes the address of the success message string onto the stack.
- ***call sub_40105F***: Calls a subroutine, likely responsible for logging or displaying the success message.
- ***add esp, 4***: Adjusts the stack pointer to clean up the argument pushed earlier.
- ***mov eax, 1***: Sets ***eax*** to ***1***, indicating success.
- ***jmp short loc_40103A***: Jumps to ***loc_40103A*** to conclude the function.

5. **Handle No Connection**

```
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
```

- ***loc_40102B***: Label marking the location for handling no Internet connection.
- ***jmp short loc_40103A***: Jumps to ***loc_40103A*** to conclude the function without any further action.

6. **End of Function**
   - ***loc_40103A***: Label marking the end of the function. Both cases (with or without Internet) converge here.

Conclusion:

This assembly code snippet is a part of a program, potentially malware, which checks for an active Internet connection using the ***InternetGetConnectedState*** function. If a connection is detected, it logs or displays a success message. The code effectively handles both scenarios (with and without Internet connection) and ensures appropriate action based on the result of the connectivity check.