x86matthew - CallRemoteAPI - Call functions in remote processes

We can currently use CreateRemoteThread to call a function in a remote process, but this is limited to functions of the following format:

```
DWORD __stdcall ThreadProc(LPVOID lpParameter);
```

I have developed a generic function that allows any API to be called remotely. Firstly, a code stub is automatically generated and written to the target process:

```
push param_1
push param_2
push param_3
...
mov eax, TargetFunction
call eax
push eax
mov eax, RtlExitUserThread
call eax
```

This code calls the target function, and then calls RtlExitUserThread with the return value from the original function.

We can use CreateRemoteThread to execute the code above within the target process, wait for the thread to exit using WaitForSingleObject, and retrieve the return value using GetExitCodeThread.

Full code below:

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

DWORD CallRemoteAPI(HANDLE hProcess, DWORD dwAddr, DWORD *pdwParamList,
DWORD dwParamCount, DWORD *pdwReturnValue)
{
DWORD dwThreadID = 0;
HANDLE hThread = NULL;
DWORD dwExitCode = 0;
```

```
DWORD dwRtlExitUserThread = 0;
DWORD dwExecFunctionCodeLength = 0;
BYTE bPushParamCode[] = { 0x68, 0x00, 0x00, 0x00, 0x00 };
BYTE bExecFunctionCode[] = { 0xB8, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xD0, 0x50, 0xB8,
0x00, 0x00, 0x00, 0x00, 0xFF, 0xD0 };
BYTE *pRemoteExecFunctionCode = NULL;

// get RtlExitUserThread address
dwRtlExitUserThread = (DWORD)GetProcAddress(GetModuleHandle("ntdll.dll"),
"RtlExitUserThread");
if(dwRtlExitUserThread == 0)
{
return 1;
}

// calculate code length:
// push param_1
// push param_2
// push param_3
// ...
// mov eax, TargetFunction
// call eax
// push eax
// mov eax, RtlExitUserThread
// call eax
dwExecFunctionCodeLength = (dwParamCount * sizeof(bPushParamCode)) +
sizeof(bExecFunctionCode);

// allocate memory in remote process
pRemoteExecFunctionCode = (BYTE*)VirtualAllocEx(hProcess, NULL,
dwExecFunctionCodeLength, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
if(pRemoteExecFunctionCode == NULL)
{
return 1;
}

// write function parameter values
for(DWORD i = 0; i < dwParamCount; i++)
{
// set current param value
*(DWORD*)&bPushParamCode;[1] = pdwParamList[dwParamCount - 1 - i];
if(WriteProcessMemory(hProcess, (BYTE*)(pRemoteExecFunctionCode + (i *
sizeof(bPushParamCode))), (void*)bPushParamCode, sizeof(bPushParamCode), NULL)
== 0)
```

```
{
// error
VirtualFreeEx(hProcess, pRemoteExecFunctionCode, 0, MEM_RELEASE);

return 1;
}
}

// set target function address
*(DWORD*)&bExecFunctionCode;[1] = dwAddr;

// set RtlExitUserThread function address
*(DWORD*)&bExecFunctionCode;[9] = dwRtlExitUserThread;

// write function execution code
if(WriteProcessMemory(hProcess, (BYTE*)(pRemoteExecFunctionCode +
(dwParamCount * sizeof(bPushParamCode))), (void*)bExecFunctionCode,
sizeof(bExecFunctionCode), NULL) == 0)
{
// error
VirtualFreeEx(hProcess, pRemoteExecFunctionCode, 0, MEM_RELEASE);

return 1;
}

// create a thread in the remote process
hThread = CreateRemoteThread(hProcess, NULL, (128 * 1024),
(LPTHREAD_START_ROUTINE)pRemoteExecFunctionCode, NULL, 0, &dwThreadID;);
if(hThread == NULL)
{
// error
VirtualFreeEx(hProcess, pRemoteExecFunctionCode, 0, MEM_RELEASE);

return 1;
}

// wait for thread to complete
WaitForSingleObject(hThread, INFINITE);

// get thread exit code (api return value)
if(GetExitCodeThread(hThread, &dwExitCode;) == 0)
{
// error
CloseHandle(hThread);
VirtualFreeEx(hProcess, pRemoteExecFunctionCode, 0, MEM_RELEASE);
```

```
return 1;
}

// close thread handle
CloseHandle(hThread);

// free remote code block
VirtualFreeEx(hProcess, pRemoteExecFunctionCode, 0, MEM_RELEASE);

// store return value
*pdwReturnValue = dwExitCode;

return 0;
}
```

If we need to allocate further data in the remote process (eg string parameters), we can use the following functions:

```
BYTE *AllocateRemoteData(HANDLE hProcess, BYTE *pData, DWORD dwDataLength)
{
BYTE *pRemoteDataAddr = NULL;

// allocate memory in remote process
pRemoteDataAddr = (BYTE*)VirtualAllocEx(hProcess, NULL, dwDataLength,
MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
if(pRemoteDataAddr == NULL)
{
return NULL;
}

if(pData != NULL)
{
// write data to remote process
if(WriteProcessMemory(hProcess, pRemoteDataAddr, (void*)pData, dwDataLength,
NULL) == 0)
{
// error
VirtualFreeEx(hProcess, pRemoteDataAddr, 0, MEM_RELEASE);

return NULL;
}
}
```

```
return pRemoteDataAddr;
}

BYTE *AllocateRemoteString(HANDLE hProcess, char *pData)
{
// allocate string data
return AllocateRemoteData(hProcess, (BYTE*)pData, strlen(pData) + 1);
}
```

Example #1 - Calling MessageBoxA in a remote process

```
int main(int argc, char *argv[])
{
DWORD dwPID = 0;
char *pMsgTextParam = NULL;
DWORD dwReturnValue = 0;
DWORD dwParamList[4];
HANDLE hProcess = NULL;
void *pMsgTitle = NULL;
void *pMsgText = NULL;

if(argc != 3)
{
printf("Usage: %s [target_pid] [msg]\n\n", argv[0]);

return 1;
}

// get command-line param values
dwPID = atoi(argv[1]);
pMsgTextParam = argv[2];

printf("Opening process: %u...\n", dwPID);

// open target process
hProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, dwPID);
if(hProcess == NULL)
{
// error
printf("Failed to open process: %u\n", dwPID);

return 1;
```

```
}

printf("Allocating strings in remote process...\n\n");

// allocate string in remote process
pMsgTitle = AllocateRemoteString(hProcess, "CallRemoteAPI");
if(pMsgTitle == NULL)
{
printf("Failed to allocate string\n");

// error
CloseHandle(hProcess);

return 1;
}

// allocate string in remote process
pMsgText = AllocateRemoteString(hProcess, pMsgTextParam);
if(pMsgText == NULL)
{
printf("Failed to allocate string\n");

// error
VirtualFreeEx(hProcess, pMsgTitle, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}

// set MessageBox parameters
dwParamList[0] = 0;
dwParamList[1] = (DWORD)pMsgText;
dwParamList[2] = (DWORD)pMsgTitle;
dwParamList[3] = MB_OK;

printf("Calling MessageBoxA in remote process...\n");

// call MessageBox in target process
if(CallRemoteAPI(hProcess, (DWORD)MessageBoxA, dwParamList, 4,
&dwReturnValue;) != 0)
{
printf("Failed to call remote API\n");

// error
VirtualFreeEx(hProcess, pMsgTitle, 0, MEM_RELEASE);
```

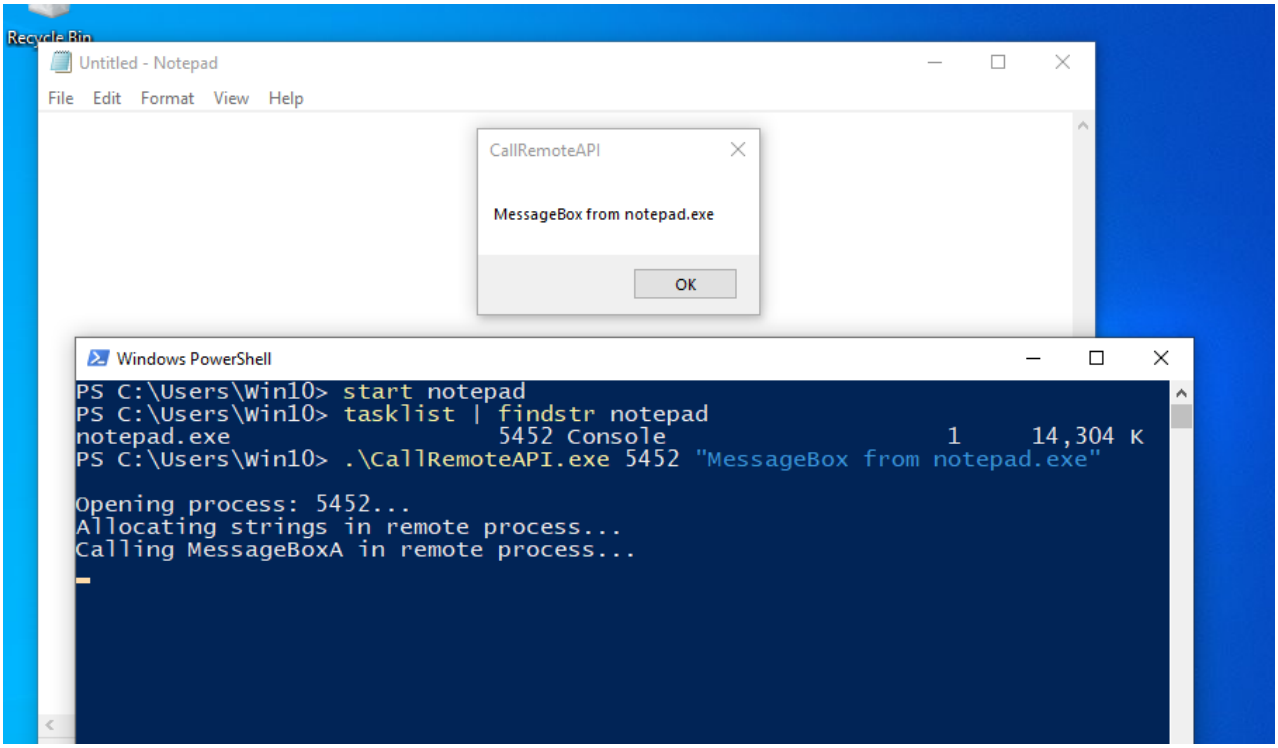
```
VirtualFreeEx(hProcess, pMsgText, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}

printf("MessageBoxA returned: %u\n", dwReturnValue);

// free remote memory
VirtualFreeEx(hProcess, pMsgTitle, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pMsgText, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 0;
}
```



Example #2 - Creating a file in a remote process

```
int main(int argc, char *argv[])
{
DWORD dwPID = 0;
char *pFilePathParam = NULL;
char *pContentParam = NULL;
DWORD dwReturnValue = 0;
DWORD dwParamList[16];
```

```
HANDLE hProcess = NULL;
void *pFilePath = NULL;
void *pContent = NULL;
void *pBytesWritten = NULL;
HANDLE hFile = NULL;

printf("CallRemoteAPI - www.x86matthew.com\n\n");

if(argc != 4)
{
printf("Usage: %s [target_pid] [file_path] [content]\n\n", argv[0]);

return 1;
}

// get command-line param values
dwPID = atoi(argv[1]);
pFilePathParam = argv[2];
pContentParam = argv[3];

printf("Opening process: %u...\n", dwPID);

// open target process
hProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, dwPID);
if(hProcess == NULL)
{
// error
printf("Failed to open process: %u\n", dwPID);

return 1;
}

printf("Allocating data in remote process...\n\n");

// allocate string in remote process
pFilePath = AllocateRemoteString(hProcess, pFilePathParam);
if(pFilePath == NULL)
{
printf("Failed to allocate string\n");

// error
CloseHandle(hProcess);

return 1;
}
```

```
// allocate string in remote process
pContent = AllocateRemoteString(hProcess, pContentParam);
if(pContent == NULL)
{
printf("Failed to allocate string\n");

// error
VirtualFreeEx(hProcess, pFilePath, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}

// allocate dword value (BytesWritten) in remote process
pBytesWritten = AllocateRemoteData(hProcess, NULL, sizeof(DWORD));
if(pBytesWritten == NULL)
{
printf("Failed to allocate value\n");

// error
VirtualFreeEx(hProcess, pContent, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pFilePath, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}

// set CreateFile parameters
dwParamList[0] = (DWORD)pFilePath;
dwParamList[1] = GENERIC_WRITE;
dwParamList[2] = 0;
dwParamList[3] = 0;
dwParamList[4] = CREATE_ALWAYS;
dwParamList[5] = FILE_ATTRIBUTE_NORMAL;
dwParamList[6] = 0;

printf("Calling CreateFileA...\n");

// call CreateFileA
if(CallRemoteAPI(hProcess, (DWORD)CreateFileA, dwParamList, 7, &dwReturnValue;)
!= 0)
{
printf("Failed to call remote API\n");

// error
VirtualFreeEx(hProcess, pBytesWritten, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pContent, 0, MEM_RELEASE);
```

```
VirtualFreeEx(hProcess, pFilePath, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}

printf("CreateFileA returned 0x%08x\n\n", dwReturnValue);

// store the returned file handle
hFile = (HANDLE)dwReturnValue;

// check if CreateFile failed
if(hFile == INVALID_HANDLE_VALUE)
{
printf("CreateFileA failed\n");

// error
VirtualFreeEx(hProcess, pBytesWritten, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pContent, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pFilePath, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}

// set WriteFile parameters
dwParamList[0] = (DWORD)hFile;
dwParamList[1] = (DWORD)pContent;
dwParamList[2] = strlen(pContentParam);
dwParamList[3] = (DWORD)pBytesWritten;
dwParamList[4] = 0;

printf("Calling WriteFile...\n");

// call WriteFile
if(CallRemoteAPI(hProcess, (DWORD)WriteFile, dwParamList, 5, &dwReturnValue;) != 0)
{
printf("Failed to call remote API\n");

// error
VirtualFreeEx(hProcess, pBytesWritten, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pContent, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pFilePath, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}
```

```
printf("WriteFile returned 0x%08x\n\n", dwReturnValue);

// set CloseHandle parameters
dwParamList[0] = (DWORD)hFile;

printf("Calling CloseHandle...\n");

// call CloseHandle
if(CallRemoteAPI(hProcess, (DWORD)CloseHandle, dwParamList, 1, &dwReturnValue;)
!= 0)
{
printf("Failed to call remote API\n");

// error
VirtualFreeEx(hProcess, pBytesWritten, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pContent, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pFilePath, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 1;
}

printf("CloseHandle returned 0x%08x\n\n", dwReturnValue);

// clean up
VirtualFreeEx(hProcess, pBytesWritten, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pContent, 0, MEM_RELEASE);
VirtualFreeEx(hProcess, pFilePath, 0, MEM_RELEASE);
CloseHandle(hProcess);

return 0;
}
```