# Info

| Name | Language | Arch | Difficulty | Platform | Downloads |
|---|---|---|---|---|---|
| Password(Very Easy) | Assembler | x86-64 | 1.0 | Windows | 1376 |

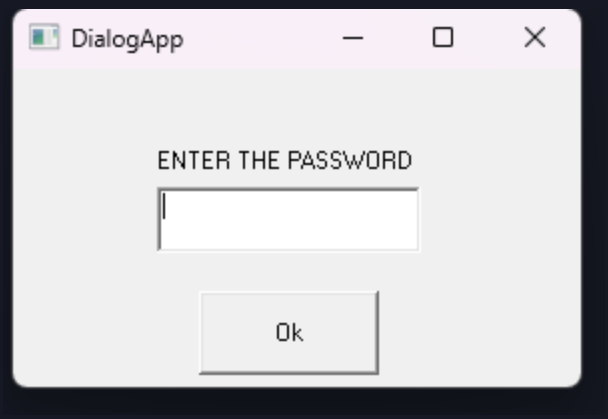
Password for folder is `crackmes.one`

---

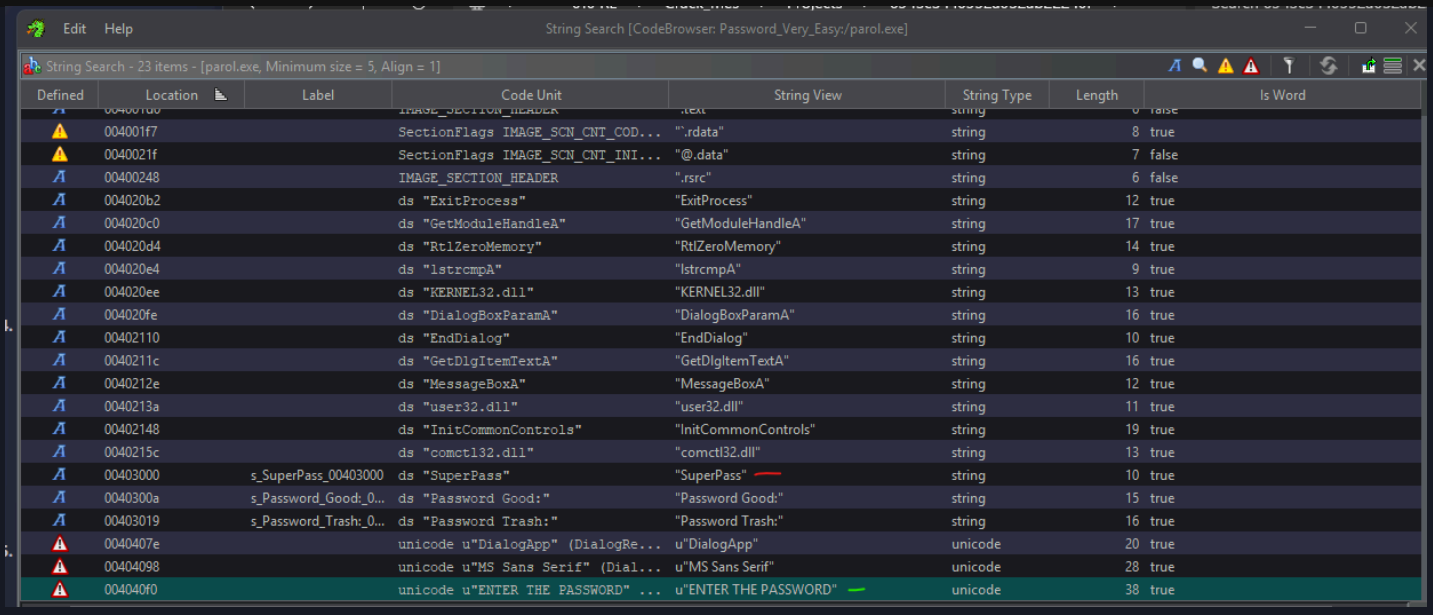
## Static Method (Ghidra)

Load program.

This appears




Ghidra -> search -> for strings




Found `ENTER THE PASSWORD` at green.

What is red? `SuperPass`

Red: 403000 -> SuperPass

```
                    s_SuperPass_00403000                            XREF[2]:     0040022c(*), 0040108b(*)
00403000 53 75 70        ds         "SuperPass"
         65 72 50
         61 73 73 00
```

`XREF[2]`:

40022c -> Link to str

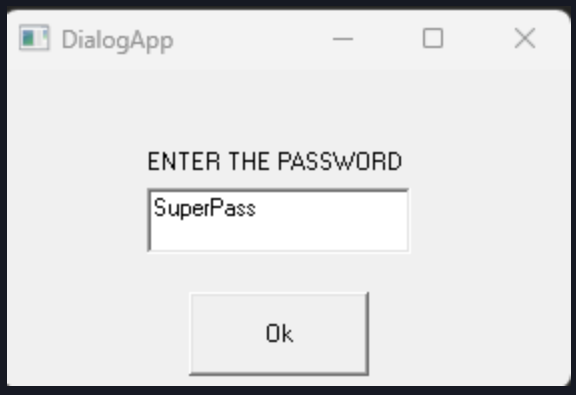
40108b -> Link to function

```
undefined4 UndefinedFunction_0040102e(HWND param_1,int param_2,uint param_3)

{
  int iVar1;
  CHAR aCStack_108 [260];

  if (param_2 == 0x110) {
    RtlZeroMemory(aCStack_108,0x104);
  }
  else if (param_2 == 0x111) {
    if ((param_3 >> 0x10 == 0) && ((param_3 & 0xffff) == 0x3eb)) {
      GetDlgItemTextA(param_1,0x3e9,aCStack_108,0x104);
      iVar1 = lstrcmpA(aCStack_108,s_SuperPass_00403000);
      if (iVar1 == 0) {
        MessageBoxA((HWND)0x0,aCStack_108,s_Password_Good:_0040300a,0);
      }
      else {
        MessageBoxA((HWND)0x0,aCStack_108,s_Password_Trash:_00403019,0);
      }
    }
  }
  else {
    if (param_2 != 0x10) {
      return 0;
    }
    EndDialog(param_1,0);
  }
  return 1;
}
```

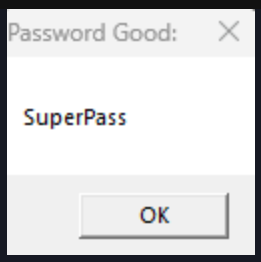
```
        00401086 e8 8b 00        CALL       USER32.DLL::GetDlgItemTextA
UINT GetDlgItemTextA(HWND hDlg,
                 00 00
        0040108b 68 00 30        PUSH       s_SuperPass_00403000
= "SuperPass"
                 40 00
        00401090 8d 85 fc        LEA        EAX,[EBP + 0xfffffefc]
                 fe ff ff
        00401096 50              PUSH       EAX
        **00401097** e8 68 00        CALL       KERNEL32.DLL::lstrcmpA
int lstrcmpA(LPCSTR lpString1, L
                 00 00
        0040109c 0b c0           OR         EAX,EAX
        **0040109e** 75 17           JNZ        LAB_004010b7
        004010a0 6a 00           PUSH       0x0
        004010a2 68 0a 30        PUSH       s_Password_Good:_0040300a
= "Password Good:"
                 40 00
        004010a7 8d 85 fc        LEA        EAX,[EBP + 0xfffffefc]
                 fe ff ff
        004010ad 50              PUSH       EAX
        004010ae 6a 00           PUSH       0x0
        004010b0 e8 67 00        CALL       USER32.DLL::MessageBoxA
int MessageBoxA(HWND hWnd, LPCST
                 00 00
        004010b5 eb 15           JMP        LAB_004010cc
                             LAB_004010b7
XREF[1]:     0040109e(j)
        004010b7 6a 00           PUSH       0x0
        004010b9 68 19 30        PUSH       s_Password_Trash:_00403019
= "Password Trash:"
                 40 00
        004010be 8d 85 fc        LEA        EAX,[EBP + 0xfffffefc]
                 fe ff ff
```

String comparison for SuperPass

Attempt:




Success!




---

# Dynamic Method (x64dbg)

Load x32dbg

Attach to parol.exe

Attach breakpoint to entrypoint

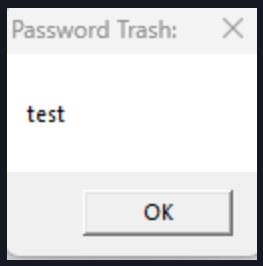
```
mov edx,dword ptr ss:[ebp+10]          [ebp+10]:EntryPoint
```

Interesting setup...

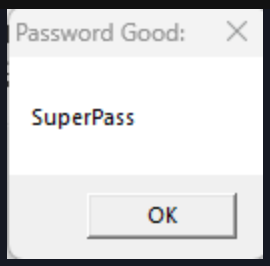
```
75403103    8BEC            mov ebp,esp
75403105    8B55 0C         mov edx,dword ptr ss:[ebp+C]    [ebp+0C]:"SuperPass"
75403108    8B4D 08         mov ecx,dword ptr ss:[ebp+8]    [ebp+08]:"test"
7540310B    E8 18470300     call kernel32.75437828
```

It looks like it is setting up a comparison

```
75437840    FF15 D80E4575   call dword ptr ds:[<CompareStringA>]
75437846    85C0            test eax,eax                    eax:"test"
75437848    75 56           jne kernel32.754378A0
7543784A    6A FF           push FFFFFFFF
7543784C    56              push esi                        esi:"SuperPass"
7543784D    6A FF           push FFFFFFFF
7543784F    57              push edi                        edi:"test"
75437850    68 00000040     push 40000000
75437855    68 00080000     push 800
7543785A    FF15 D80E4575   call dword ptr ds:[<CompareStringA>]
```




Lets try with SuperPass




---

## Frida

Things to note from previous findings!

`0x00401097` : This is the function call to compare strings. We give them 2 args. This will check those args

```
// frida -l hook.js -f parol.exe

setTimeout(() => {
    const callSite = ptr("0x00401097");

    Interceptor.attach(callSite, {
        onEnter(args) {
            const esp = this.context.esp;
            const input    = esp.add(0).readPointer().readAnsiString();
            const expected = esp.add(4).readPointer().readAnsiString();
            console.log("\ninput:   ", input);
            console.log("expected:", expected);
        }
    });
}, 1000);
```

`0x0040109e` : This is a jump if not equal function. noping them will make the jump not happen and make it so that we get a yes no matter what

```
// frida -l hook.js -f parol.exe

setTimeout(() => {
    const jnz = ptr("0x0040109e");
    Memory.protect(jnz, 4096, 'rwx');
    jnz.writeByteArray([0x90, 0x90]);
    Memory.patchCode(jnz, 0, () => {});
    console.log("bytes:", jnz.readByteArray(2));
}, 1000);
```

---

# C++ DLLs

Observation

```
#include <windows.h>
#include <iostream>
#include <cstdio>

// ---------------------------------------------------------------------
// Inline detour -- 5-byte relative JMP patch
// ---------------------------------------------------------------------

using lstrcmpA_t = int(WINAPI*)(LPCSTR, LPCSTR);

static lstrcmpA_t oLstrcmpA    = nullptr;
static BYTE       origBytes[5] = {};

static int WINAPI hkLstrcmpA(LPCSTR s1, LPCSTR s2) {
    std::cout << "\n[lstrcmpA]"
              << "\n  input:    " << (s1 ? s1 : "(null)")
              << "\n  expected: " << (s2 ? s2 : "(null)")
              << "\n";
    return oLstrcmpA(s1, s2);
}

static void WriteJmp(uintptr_t src, uintptr_t dst) {
    DWORD old;
    VirtualProtect((LPVOID)src, 5, PAGE_EXECUTE_READWRITE, &old);
    *(BYTE*)src        = 0xE9;
    *(DWORD*)(src + 1) = (DWORD)(dst - src - 5);
    VirtualProtect((LPVOID)src, 5, old, &old);
    FlushInstructionCache(GetCurrentProcess(), (LPVOID)src, 5);
}

static bool Hook() {
    // KERNELBASE is where lstrcmpA actually lives on modern Windows
    HMODULE hMod = GetModuleHandleA("KERNELBASE.dll");
    if (!hMod) hMod = GetModuleHandleA("kernel32.dll");
    if (!hMod) { std::cout << "[-] kernel module not found\n"; return false; }

    auto target = (uintptr_t)GetProcAddress(hMod, "lstrcmpA");
    if (!target) { std::cout << "[-] lstrcmpA not found\n"; return false; }
    std::cout << "[+] lstrcmpA at 0x" << std::hex << target << "\n";

    // Save original 5 bytes
    memcpy(origBytes, (void*)target, 5);

    // Trampoline: original 5 bytes + jmp back to target+5
    auto tramp = (uintptr_t)VirtualAlloc(nullptr, 32, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);
    if (!tramp) { std::cout << "[-] VirtualAlloc failed\n"; return false; }
    memcpy((void*)tramp, origBytes, 5);
    WriteJmp(tramp + 5, target + 5);
    oLstrcmpA = (lstrcmpA_t)tramp;

    // Patch target -> hook
    WriteJmp(target, (uintptr_t)hkLstrcmpA);
```

```
    std::cout << "[+] Hook installed\n";
    return true;
}

static void Unhook() {
    HMODULE hMod = GetModuleHandleA("KERNELBASE.dll");
    if (!hMod) hMod = GetModuleHandleA("kernel32.dll");
    if (!hMod) return;

    auto target = (uintptr_t)GetProcAddress(hMod, "lstrcmpA");
    if (!target) return;

    DWORD old;
    VirtualProtect((LPVOID)target, 5, PAGE_EXECUTE_READWRITE, &old);
    memcpy((void*)target, origBytes, 5);
    VirtualProtect((LPVOID)target, 5, old, &old);
    FlushInstructionCache(GetCurrentProcess(), (LPVOID)target, 5);
    std::cout << "[+] Hook removed\n";
}

// ---------------------------------------------------------------------
// Thread
// ---------------------------------------------------------------------

DWORD WINAPI HackThread(HMODULE hModule) {
    AllocConsole();
    FILE* f;
    freopen_s(&f, "CONOUT$", "w", stdout);

    std::cout << "[observe.dll] Loaded -- press END to unload\n";
    Sleep(1000);

    if (!Hook()) goto cleanup;

    std::cout << "[*] Enter a password and click OK\n";

    while (!(GetAsyncKeyState(VK_END) & 1))
        Sleep(100);

    Unhook();

cleanup:
    fclose(f);
    FreeConsole();
    FreeLibraryAndExitThread(hModule, 0);
    return 0;
}

// ---------------------------------------------------------------------
// DllMain
// ---------------------------------------------------------------------

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID
lpReserved) {
```

```
    switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
        CloseHandle(CreateThread(nullptr, 0, (LPTHREAD_START_ROUTINE)HackThread,
hModule, 0, nullptr));
        break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

Nop Slide

```
#include <windows.h>
#include <iostream>
#include <cstdio>

// JNZ instruction at 0x0040109e (75 17) -- jumps to "Password Trash" block
// NOP patch (90 90) makes it fall through to "Password Good" always
constexpr uintptr_t JNZ_ADDR   = 0x0040109e;
constexpr BYTE      NOP_PATCH[] = { 0x90, 0x90 };
constexpr SIZE_T    PATCH_SIZE  = sizeof(NOP_PATCH);

static BYTE origBytes[PATCH_SIZE] = {};

static bool Patch() {
    auto target = (LPVOID)JNZ_ADDR;

    // Verify bytes match expected JNZ before patching
    BYTE current[PATCH_SIZE];
    memcpy(current, target, PATCH_SIZE);
    std::cout << "[*] Bytes at 0x" << std::hex << JNZ_ADDR << ": ";
    for (auto b : current) std::cout << std::hex << (int)b << " ";
    std::cout << "\n";

    if (current[0] != 0x75) {
        std::cout << "[-] Expected JNZ (0x75), got 0x" << std::hex <<
(int)current[0]
                  << " -- wrong address or already patched\n";
        return false;
    }

    // Save original bytes for restore
    memcpy(origBytes, target, PATCH_SIZE);

    // Make page writable, write NOPs, restore protection, flush cache
    DWORD old;
    if (!VirtualProtect(target, PATCH_SIZE, PAGE_EXECUTE_READWRITE, &old)) {
        std::cout << "[-] VirtualProtect failed: " << GetLastError() << "\n";
        return false;
    }
    memcpy(target, NOP_PATCH, PATCH_SIZE);
    VirtualProtect(target, PATCH_SIZE, old, &old);
    FlushInstructionCache(GetCurrentProcess(), target, PATCH_SIZE);

    // Verify write succeeded
    BYTE after[PATCH_SIZE];
    memcpy(after, target, PATCH_SIZE);
    std::cout << "[*] Bytes after patch:  ";
    for (auto b : after) std::cout << std::hex << (int)b << " ";
    std::cout << "\n";

    if (after[0] != 0x90) {
        std::cout << "[-] Patch did not stick\n";
        return false;
    }
```

```
    std::cout << "[+] JNZ patched to NOP -- any password accepted\n";
    return true;
}

static void Restore() {
    auto target = (LPVOID)JNZ_ADDR;
    DWORD old;
    VirtualProtect(target, PATCH_SIZE, PAGE_EXECUTE_READWRITE, &old);
    memcpy(target, origBytes, PATCH_SIZE);
    VirtualProtect(target, PATCH_SIZE, old, &old);
    FlushInstructionCache(GetCurrentProcess(), target, PATCH_SIZE);
    std::cout << "[+] JNZ restored\n";
}

// ---------------------------------------------------------------------
// Thread
// ---------------------------------------------------------------------

DWORD WINAPI HackThread(HMODULE hModule) {
    AllocConsole();
    FILE* f;
    freopen_s(&f, "CONOUT$", "w", stdout);

    std::cout << "[noppatch.dll] Loaded -- press END to restore and unload\n";
    Sleep(1000);

    if (!Patch()) goto cleanup;

    std::cout << "[*] Enter any password and click OK\n";

    while (!(GetAsyncKeyState(VK_END) & 1))
        Sleep(100);

    Restore();

cleanup:
    fclose(f);
    FreeConsole();
    FreeLibraryAndExitThread(hModule, 0);
    return 0;
}

// ---------------------------------------------------------------------
// DllMain
// ---------------------------------------------------------------------

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID
lpReserved) {
    switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
        CloseHandle(CreateThread(nullptr, 0, (LPTHREAD_START_ROUTINE)HackThread,
hModule, 0, nullptr));
        break;
```

```
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

---