The Service Run Failed Successfully

zerosalarium.com/2026/02/Defense-Evasion-The-service-run-failed-successfully.html

Zero Salarium February 8, 2026

Defense Evasion: The Service Run Failed Successfully

I. LEAD-IN

In the process of red-teaming, what we often do during lateral movement is perform remote execution through other machines in the network. Depending on the context, we can leverage available functions like SMB, WMI, WinRM, or use exploits if you have some modern tools in your arsenal.

The method I want to discuss in this article is remote execution via SMB, akin to PSExec and Impacket SMB.

This article will focus on finding ways to execute remote services more stealthily, avoiding the traditional execution paths we usually rely on. You can also use this technique to create more effective persistence.

II. MAIN SECTION

1. Remote Code Execution through Services and Its Limitations

When attackers aim to execute payloads on remote machines, one of their go-to techniques is creating malicious services. They connect to the Service Control Manager (SCM) on a remote host using administrative credentials. They then establish a new service pointing to their payload (such as a malware executable or script).

Execution Flow:

- The attacker authenticates to the remote system.
- Creates a service with a malicious binary as the executable path.
- Starts the service, which runs the payload under SYSTEM privileges.
- Deletes or hides the service to minimize detection.

Common Detection Techniques:

- Correlate service creation with remote logons (Event ID 4624 + 7045/4697).
- Look for unusual service names (random strings, suspicious executables).
- Alert on PsExec service (PSEXESVC) creation.
- Monitor for services pointing to non-standard paths (e.g., C:\Users\Public\payload.exe)

Of course, for each detection or monitoring method mentioned, there are different evasion techniques. For instance, using DLL-Hijacking with files signed by Windows or employing Well-Known service names.

However, the biggest bottleneck is the **ImagePath** of the service you create or modify. No matter how well you conceal it, there will always be something to track in the **ImagePath**. For example, if you switch to using DLL-Hijacking, you'll have to load an image that lacks a digital signature. Utilizing **LOLBIN** provides few options and can easily leave traces.

This leads me to wonder if there’s a stable method to execute code without touching the **ImagePath** at all.

2. Windows Services Recovery

Windows services come with built-in recovery options that allow administrators to define actions when a service fails.

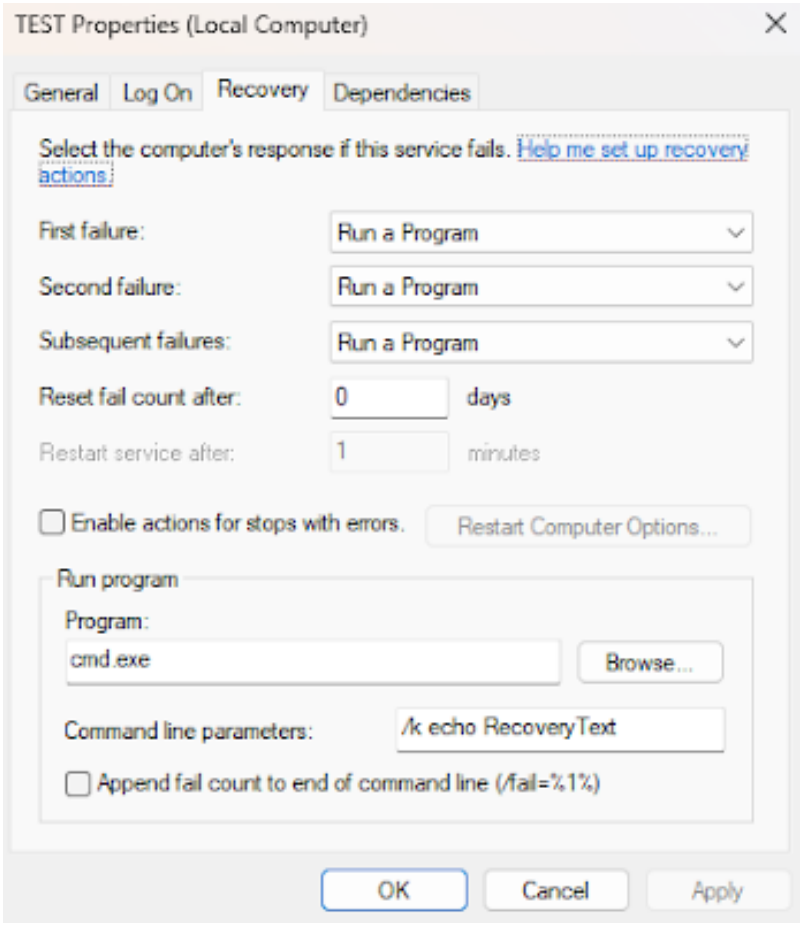
In the **Recovery** tab of the **Services** console, you can configure actions for:

- First failure

- Second failure
- Subsequent failures

Available Actions:

- Restart the service
- Run a program (such as a logging or alert script)
- Restart the computer (rarely used, mainly for critical services)



As mentioned above, if I can somehow create or modify a service with a valid **ImagePath** that crashes upon execution, it will trigger the recovery function to run a program or script. Thus, there's a high chance we will be listed as safe when the admin queries information about the service.

A new problem arises here: how can I make a service crash?

If I inject code into the service (remember, we are running a remote service), it’s likely that we’ll return to the original issue: needing to use a hijack module or a separate payload file.

If I can inject code, I might as well run the payload directly instead of going through the long route with service recovery.

A Windows service, when running, requires an execution environment similar to a normal process and the associated services. When these environmental factors change, there is a likelihood that the service will crash.

In this context, a crash occurs when the ServiceMain has been executed and registered with the Service Control Manager. The service process either crashes or exits with an error code.

Steps to find a service that will crash upon execution:

1. Enumerate all services on Windows.
2. Filter to find services with a status of stopped.
3. Sequentially test each service:
    1. Start the service.
    2. Wait about 1-3 seconds.
    3. Check if the service hasn't stopped, then stop the service.
4. After starting/stopping all services, parse Windows Event 7024 and 7031 (Windows Logs => System).

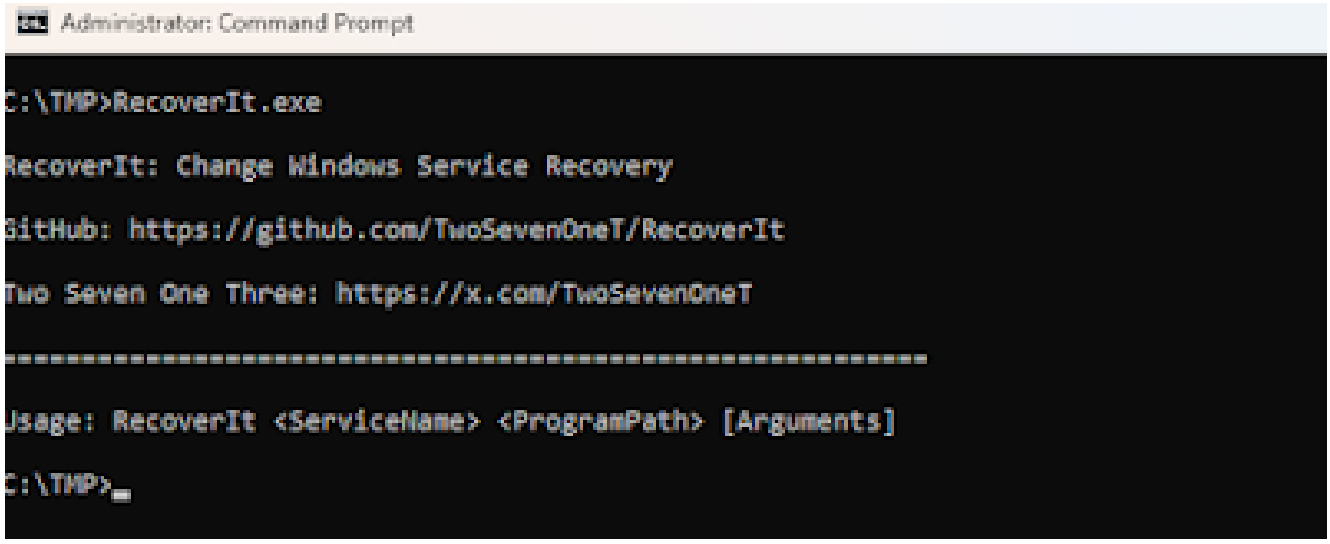
I will not release this public search tool. You can use AI to generate code or a script to perform these steps.

3. Exploiting Windows service failure recovery functions

I discovered a service named "**UevAgentService**" that crashes each time it runs. Based on the Operational Event Log, this service crashes because the UE-V service is disabled.

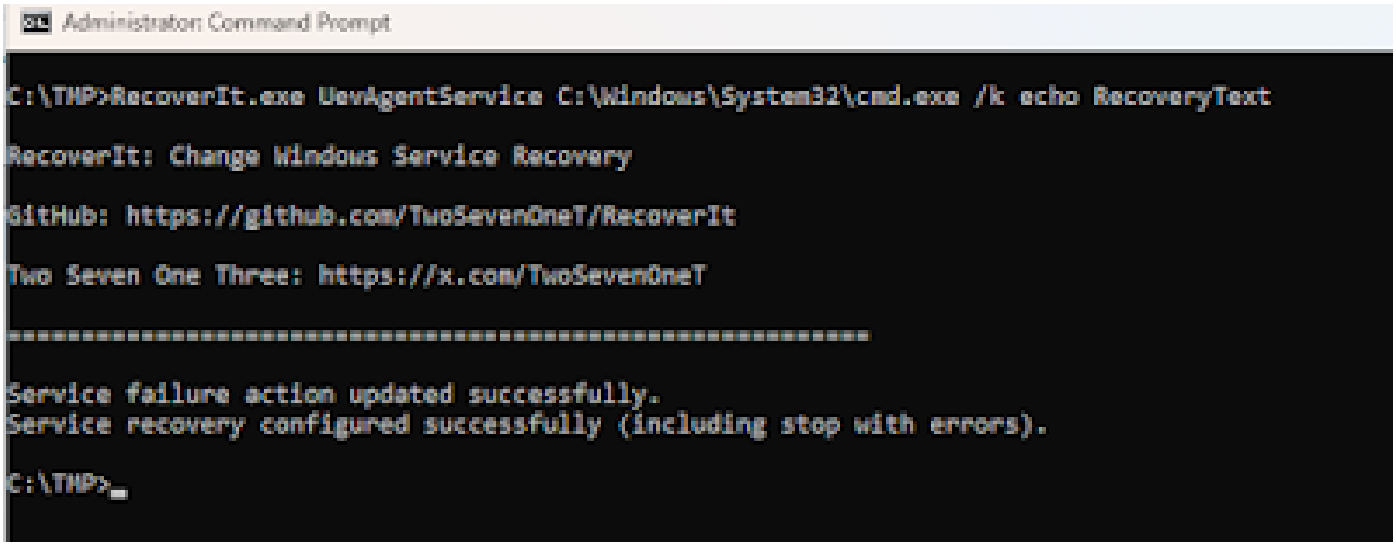
To exploit the recovery function of Windows services, I developed a tool to change the configuration of a service. In reality, we seldom have a user interface for this, right? You can download the **RecoverIt** tool from the link below.

https://github.com/TwoSevenOneT/RecoverIt

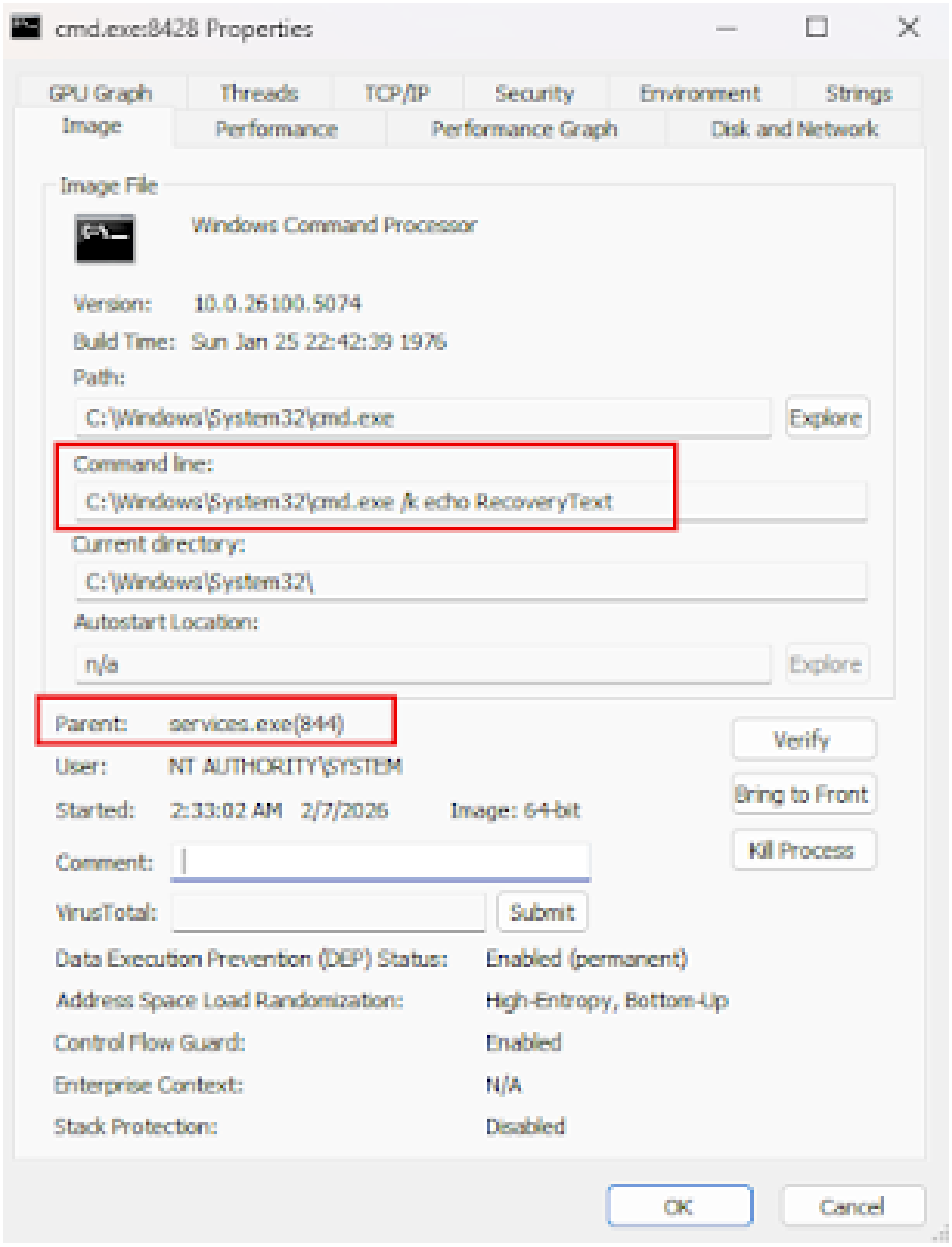


The RecoverIt tool requires three parameters: the name of the service whose configuration needs to be changed, the program to run when a crash occurs, and the parameters for this program.

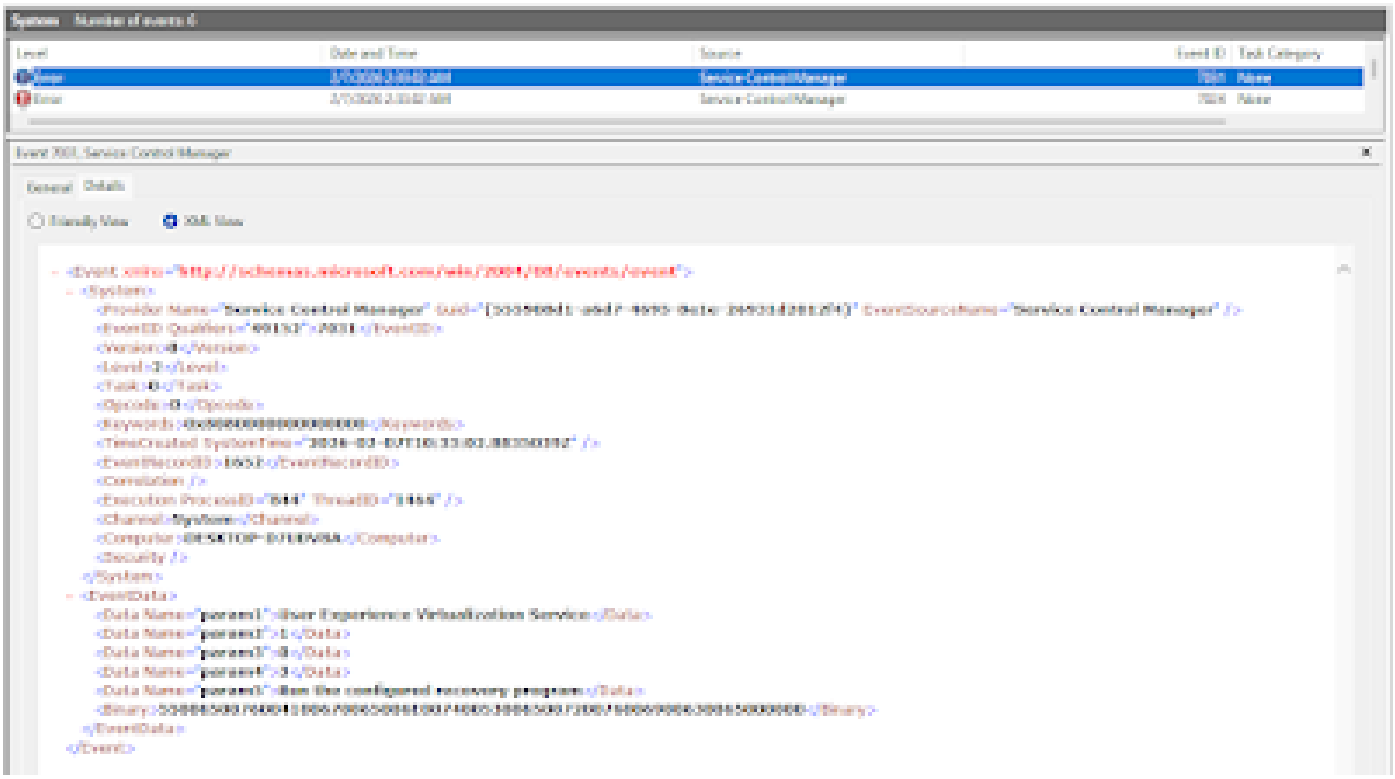
I will run RecoverIt with the UevAgentService:



With the configuration above, each time I start the **UevAgentService**, when a crash occurs, Services.exe will execute the **CMD** program with the corresponding parameters.



The information recorded in the Windows Event:



As you can see above, the program executed during the crash will not be present in this event.

This failure recovery execution function will provide red teamers with a more stealthy way to establish persistence or lateral movement since SysAdmins tend to focus more on the ImagePath of services.

Demo video: https://youtu.be/zu3lchEOrxE

III. SUMMARY

Lateral movement techniques are often closely monitored by SysAdmins and EDR solutions. This is because these techniques are frequently utilized during pentesting and red-teaming.

Remote execution or establishing persistence through Windows Services is a commonly used technique due to its stability and simplicity. Since the **ImagePath** of a service is consistently monitored and analyzed in detail, many evasion methods have been developed for this record. However, their effectiveness has been decreasing while the difficulty of evading detection has been increasing.

Using the service recovery function to trigger payload execution avoids the primary limitation of needing to modify the **ImagePath** or the executables found in the original **ImagePath**. However, it introduces a different challenge: **determining how to trigger the service crash on the remote machine**.

To detect the usage of this technique, we can pay closer attention to the **FailureCommand** and **FailureActions** of the service.

Find me on X to get the latest pentest and red team tricks that I've been researching: [Two Seven One Three (@TwoSevenOneT) / X]