Zombie Processes

What is zombie process?

Zombie process are created when we abort a job manually from the scheduler inside RED, what happens in the background is all transactions are killed in the database and repository but the PowerShell script which was started initially for the job is not killed from RED abort action. Thus, this creates a zombie PowerShell process being run in the background. This could cause a delay and use massive space in the long run as we continue to abort jobs manually daily.

How to get rid of Zombie Process?

We need to run PowerShell script below to get sequence ID's and run select queries on metadata and get zombie PS function in variable.

Then we need to execute the minus query on PS script with Sequence ID and active sequence.

Then we will get the minus result in a variable and use those variables' values to kill the zombie jobs.

This is the PowerShell script used to get all WS process: -

```
Function getPID($seqNumber){ $process = "powershell.exe" $pidn ="" $allCommands = Get-WmiObject
Win32_Process -Filter "name = '$process'" | Select-Object CommandLine, ProcessId $seqNumber=
"wsl"+$seqNumber + "j" foreach ($cmdline in $allCommands){ if ($cmdline.CommandLine -match
$seqNumber){ $pidn = $cmdline.ProcessId } } return $pidn } $pidnew=getPID( "315") $pidnew
```

-- Continue by Sushant on next page

By Sushant – 18-July-23

In **Zombie Processes.docx**, they have just provided the code for getting process_id for any sequence number. But not provided entire code to get zombie powershell processes and to kill them.
I have come up with below steps to determine zombie powershell processes and to kill them from the desription in the document.

--**Steps**:

1. Get all sequence numbers

metadata tables ws_wrk_job_run to check for active (running/failed) job sequences.

2. Get all sequence numbers from metadata tables ws_wrk_job_log and wx_wrk_audit_archive to check for non-active (completed/failed-aborted) job sequences.
3. Execute minus query to get only non-active job sequences which are older than 7 days.
4. Get process_ids for all non-active job sequences.
5. Kill those zombie powershell processes.

The function **getPID()** from the document was only giving one process_id for each job sequence. But we can have multiple powershell processes running for single job sequence. So modified the function as below to get comma separated list of all powershell process ids running for that particular job sequence.

```
Function getPID($seqNumber)
{    $process = "powershell.exe"
   $pidn =""
```

```
   $allCommands = Get-WmiObject Win32_Process -Filter "name =
'$process'" | Select-Object CommandLine, ProcessId
   $seqNumber= "wsl"+$seqNumber + "j"
   $counter = 0

   foreach ($cmdline in $allCommands)
   { if ($cmdline.CommandLine -match $seqNumber)
      #{ $pidn = $cmdline.ProcessId }
      { if ($counter -ne 0)
         {
            $pidn += ","
         }


        $pidn += $cmdline.ProcessId
        $counter = $counter + 1
      }
   }
   return $pidn
}
 ----------
 $pidnew=getPID( "271")
 ----------
 $pidnew
 O/P: 18956,10360,18680
```

Script::

```
$Server= "DESKTOP-S8MM9F6"#Enter your Server here

$DB="SF_REPO"#Enter your Repository DB name

$SEQ = @(Invoke-Sqlcmd -ServerInstance $Server -Database $DB -Query "SELECT
wjr_sequence as id FROM ws_wrk_job_run where wjr_started = getdate()-7

union all

SELECT wjl_sequence as id FROM ws_wrk_job_log where wjl_started = getdate()-7

union all

SELECT distinct wa_sequence as id from wx_wrk_audit_archive as id where
wa_time_stamp = getdate()-7

union all

SELECT distinct wd_sequence as id from wx_wrk_error_archive as id where
wd_time_stamp = getdate()-7;")



Function getPID($seqNumber) {

$process = "powershell.exe"

$pidn = ""

$allCommands = Get-WmiObject Win32_Process -Filter "name = '$process'" | Select-
Object CommandLine, ProcessId

$seqNumber= "wsl" + $seqNumber + "j"

$counter = 0



foreach ($cmdline in $allCommands) {

if ($cmdline.CommandLine -match $seqNumber) {

if ($counter -ne 0) {

$pidn += ","

}

$pidn += $cmdline.ProcessId

$counter = $counter + 1
```

```
}
}
return $pidn
}


# File path to save the output
$filePath = "E:\ZombieProcess\ZP.txt" #Add your own directory
Write-Output "Zombie Process ID's: " | Out-File -FilePath $filePath


$zombieProcessFound = $false


foreach ($seqNumber in $SEQ) {
$pidnew = getPID($seqNumber.id)
if (![string]::IsNullOrEmpty($pidnew)) {
$zombieProcessFound = $true
Write-Output "We are in loop" | Out-File -FilePath $filePath -Append
$pidArray = $pidnew -split ',' # Split the PID string into an array of PIDs
foreach ($pidToKill in $pidArray) {
$pidToKill | Out-File -FilePath $filePath -Append
# Kill each PID one by one using taskkill and suppress the output
Start-Process -FilePath "taskkill" -ArgumentList "/PID", $pidToKill, "/F" -Wait -
PassThru | Out-Null


# Introduce a delay of 1 second before proceeding to the next PID
#Start-Sleep -Seconds 1
```

```
$killResult = Get-Process -Id $pidToKill -ErrorAction SilentlyContinue

if ($killResult -eq $null) {

Write-Output "Zombie process with PID $pidToKill killed successfully." | Out-File -FilePath $filePath -Append

} else {

Write-Output "Failed to kill zombie process with PID $pidToKill." | Out-File -FilePath $filePath -Append

}

}

}

}



if (-not $zombieProcessFound) {

Write-Output "No Zombie processes found to kill." | Out-File -FilePath $filePath -Append

}
```