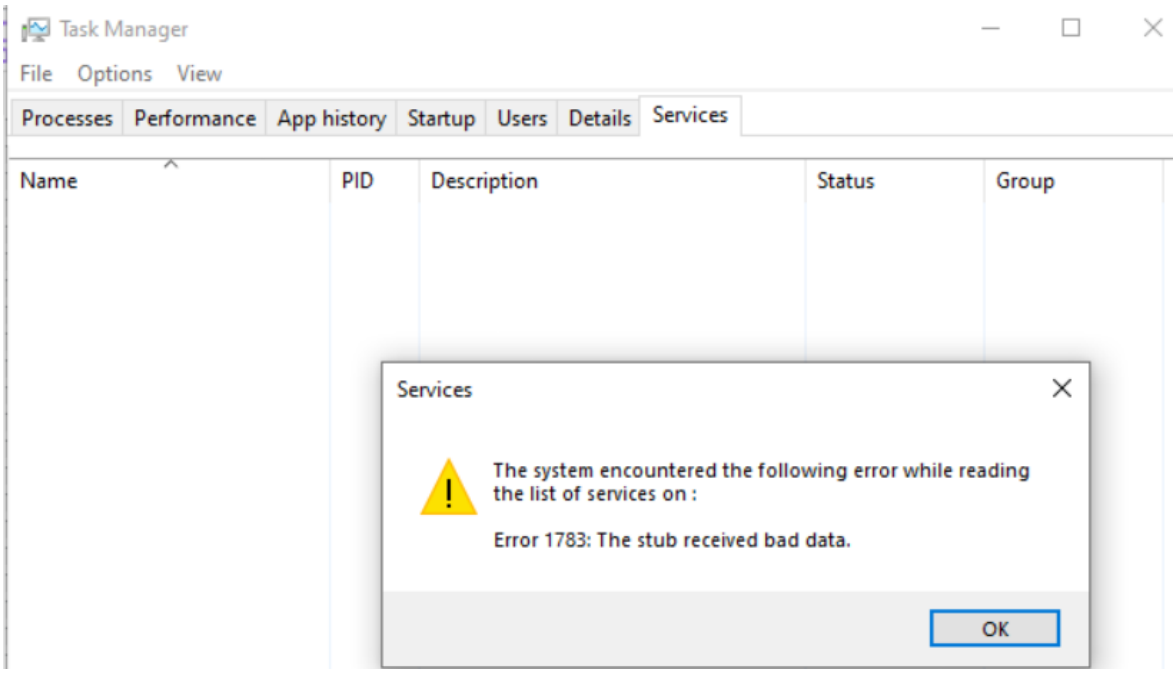
TEAM TEXAS
INCIDENT RESPONSE:
PAEXEC REPLICATION

Summary:

The incident outlined in this report involves the self-replication of our employee's logon services on our Windows 10 workstation (mesquite) on our auto domain, which would overload the system's resources and cause it to crash. Mesquite was a workstation that belonged to the auto domain, a domain belonging to an organization focusing on autonomous vehicles which required the IT team to gain control of after the employees refused to hand over credentials. The source of the incident was traced back to PAexec, a core component for handling remote logons into the system for our employees to copy/execute files/commands. Every logon would create this service as an open process, but would not close when users log out, which had caused the overload of the system. It is unknown if this incident was intentional or unintentional but has since been temporarily solved by creating a script to clear the replicated processes on a 30-minute timer, with the IT team looking into methods to ensure each service process shuts down.

Initial Reconnaissance:

On April 8th, it was discovered that our mesquite machine had crashed with a windows error screen after too much ram was being used on the system. Restarting the system was found to have freed 2 Gigabytes of storage, however certain processes were starting to fail such as task manager receiving bad data and later crashing the windows service manager. Checking system logs and Graylog determined nothing out of the ordinary around the time of the shutdown.



On April 10th, the IT team managed to fix the services on Mesquite. The error for task manager and service manager was "Error 1783". Clicking the following link will direct you to Microsoft's official website goes into detail about the error. To summarize the article, the error is caused when the number of services installed exceeds the limit of the number of services the system can control at a single time, with the solution being to search for and uninstall any unneeded services within the system. This led to the discovery of small executables labeled by "nagios" to have been flooding the system seen in the photo below:

PAExec-2951852-nagios.classex.tu.exe	4/13/2024 2:42 PM	Application	0 KB
PAExec-2955205-nagios.classex.tu.exe	4/13/2024 2:44 PM	Application	0 KB
PAExec-2958549-nagios.classex.tu.exe	4/13/2024 2:46 PM	Application	0 KB
PAExec-2961967-nagios.classex.tu.exe	4/13/2024 2:48 PM	Application	0 KB
PAExec-2965303-nagios.classex.tu.exe	4/13/2024 2:50 PM	Application	0 KB
PAExec-2968640-nagios.classex.tu.exe	4/13/2024 2:52 PM	Application	0 KB
PAExec-2971990-nagios.classex.tu.exe	4/13/2024 2:54 PM	Application	0 KB
PAExec-2975385-nagios.classex.tu.exe	4/13/2024 2:56 PM	Application	0 KB
PAExec-2978740-nagios.classex.tu.exe	4/13/2024 2:58 PM	Application	0 KB
PAExec-2982068-nagios.classex.tu.exe	4/13/2024 3:00 PM	Application	0 KB
PAExec-2985431-nagios.classex.tu.exe	4/13/2024 3:02 PM	Application	0 KB
PAExec-2988772-nagios.classex.tu.exe	4/13/2024 3:04 PM	Application	0 KB
PAExec-2992148-nagios.classex.tu.exe	4/13/2024 3:06 PM	Application	0 KB
PAExec-2995479-nagios.classex.tu.exe	4/13/2024 3:08 PM	Application	0 KB
PAExec-2998906-nagios.classex.tu.exe	4/13/2024 3:10 PM	Application	0 KB
PAExec-3002314-nagios.classex.tu.exe	4/13/2024 3:12 PM	Application	0 KB

The time duration between these attacks lined up with times on graylog of users logging into our system, leading us to believe that the issue occurring was caused by a misconfiguration within PAExec.

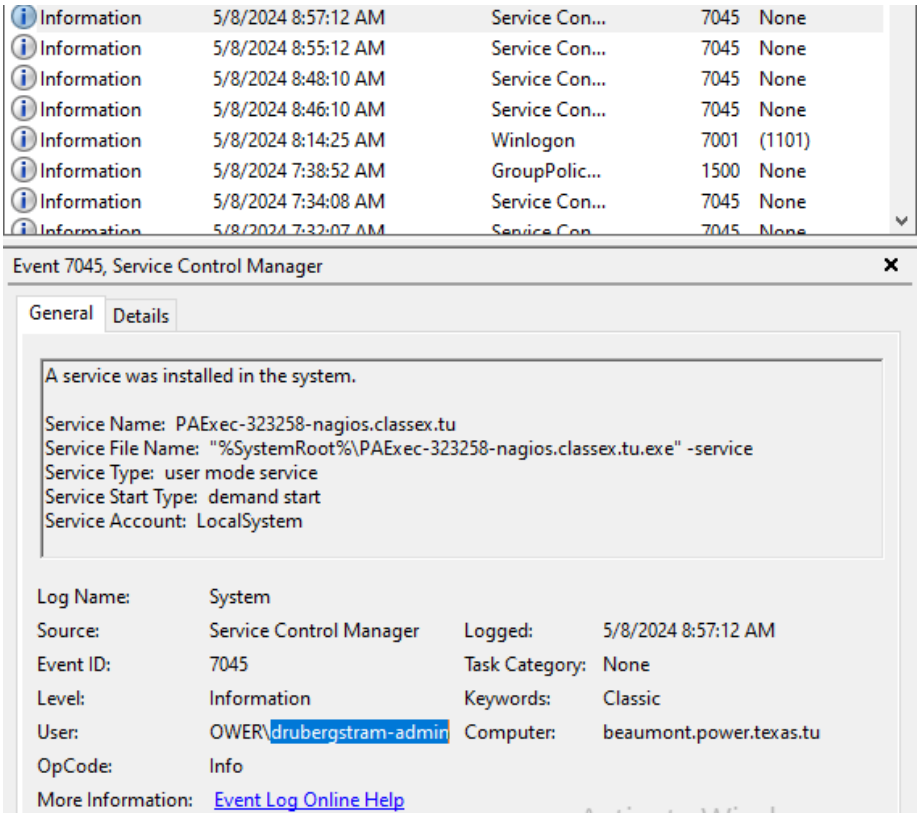
Initial Access:

Mesquite had belonged to the auto domain, which was bought out by the company from a smaller AI organization. After disgruntlement from the previous organization's IT team, systems were left pre-configured with no access credentials, requiring our IT team to regain control of the systems. After regaining control, our IT team discovered that PAExec, a tool used for remote system administration, was pre-installed on the device. PAExec is typically used for legitimate purposes, such as remote execution of processes, and was configured to handle the log in of our own organizational users. However, it was observed that PAExec was configured to auto-replicate (either intentionally or by mistake) and did not delete itself when users logged out. This behavior is atypical and suggests a potential misconfiguration or a deliberate attempt to maintain persistent access to the system.

Breakdown of PAExec:

PAExec is a remote execution tool designed for Windows systems to allow for the administration and automation of tasks across multiple systems. At a high level, this tool allows users to execute programs, commands, and scripts on remote systems by taking the desired script to run and copying onto the machine, simulating the process as if it was run locally. For our organization, it allows administrators to deploy updates, or perform routine tasks without physically needing to access the machine.

The incident would begin with employees accessing our mesquite system with PAEXEC with their desired command (in this case, an executable labeled "*-nagios.classex.tu.exe", a commonly known executable name used by the company). It is estimated that users log into our system every 3 minutes, with each login causing PAEXEC to copy the files over. The photo below shows an example of this in action on our Beaumont Power domain:



Execution/Persistence:

After successfully logging into the system, and the employees' script is successfully copied to the machine, the machine would then begin to execute the script (with the execution of the contents of the script being different depending on the user). Normally, after execution, PAExec handles the cleanup of executables after they are run on the remote system. However, in this scenerio, the process for handling the shutdown of the script fails, causing the service to continue running on the system, even after the user logs off, causing persistence on the system. Given that this has not been issued on other running windows machines, we believe this to be a misconfiguration either unintentionally or deliberately placed by the previously disgruntled employees.

Evasion:

While PAExec is designed to remove temporary executables after execution, misconfigurations had led to these files persisting on the system. The scripts executed by our IT team are small (less than 1 KB in size). Over time, these small remnants can accumulate within open services and

critical directories, gradually consuming system resources. This silent buildup did not immediately trigger any alerts, allowing it to slip through the radar for as long as it had.

Impact:

Once the number of services exceeded what the system can manage, the system began to shut down other services to make room. Initially, this had caused issues with our employees being able to run PSExec/and PAExec services, with nagios (service outage monitor) reporting frequent issues by employees. This would continue until the system crashed, preventing our users from logging into the system until it could be brought back online.

The IT team were successful in bringing the system back online along with saving any user data within the system before the crash. Because most of the PAexec services were temporary processes (only created after a user had logged in), restarting the had freed up a significant amount of space (Estimated 2 GB). Thankfully, after deleting any remnants of open PAExec services, the system ran normally without any issue. However, if left unchecked, the issue would simply persist again unless a method was incorporated to handle the shutdown of leftover services.

Clean Up/Response:

After learning about the issue, the IT team put together a powershell script to handle deleting unnecessary PAExec services on a 15 minutes timer. This was done by using windows Task Scheduler, a built in tool that allows users to automate the execution of tasks at specified times or in response to certain events on windows machines. The following is the command used to call our script using windows Task Scheduler:

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -WindowStyle hidden -NoProfile -ExecutionPolicy Bypass -File ("(thefilename).ps1")

The following is a breakdown of the function:

1. C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe:

   we first pass the location of powershell.exe, the process we want task scheduler to call that will handle running our script.

2. -WindowStyle hidden:

   This parameter runs the powershell script outside of a visible window, keeping it silent and in the background.

3. -NoProfile:

   This parameter starts powershell without loading a user profile, preventing any user customization from effecting the script.

4. -ExecutionPolicy Bypass:

   This parameter allows the script to run without being blocked by the systems execution policy, allowing the script to run without any restrictions or interruption.

5. -File "(thefilename).ps1:

   This parameter tells powershell the name of the file we want to run, with (thefilename) being replaced with the name of our file and ps1 being the powershell script extension.

This function allows us to call our clean up script described below:

```
stop-service -name "PAExec*"
Remove-Item -Path HKLM:\SYSTEM\CurrentControlSet\Services\PAExec* -Force -Verbose
Get-ChildItem -C:\Windows -recurse -Filter "PAExec*" | Remove-Item -Verbose
```

The following is a breakdown of each line of the script:

1. stop-service -name "PAExec*":

   This command handles stopping any service that contains PAExec as the beginning service name, followed by any combination of words or strings. Comparing it to the screenshot found within initial reconnisance, we can see that this will handle all PAExec files left open.

2. Remove-Item -Path HKLM:\SYSTEM\CurrentControlSet\Services\PAExec* -Force -Verbose:

This command deletes any registry entry related to PAExec from the system registry, with -Path handling the path of the location we want to delete the registries from, -Force forcing the removal of the registry (in the event the system may attempt to keep it alive), and -Verbose to provide a detailed output of the action being taken for debugging purposes (so we can check and ensure the registries were deleted.

3. Get-ChildItem -Path C:\Windows -Recurse -Filter "PAExec*" | Remove-Item -Verbose:

   This command deletes any files/directories related to PAExec within C:\Windows and its subdirectories. In the event the system gets flooded with PAExec services, it's not uncommon for the system to start to place the corresponding scripts within different directories. -Path ensures we start at the C:\Windows directory, -Recurese ensures we check all subdirectories of the path, -Filter ensures we only get the files we are filtering for, and -Verbose to debug the output.

As of 5/16/2024, this method has been sufficient in ensuring that rogue PAExec files get deleted. However, this is only a short-term temporary method so that users can continue to use the system/services without any errors. The IT team is looking into both PAExec and the scripts being run by our users to crack down on the source of what's causing the failing of closure of these services. Again, due to the of the incident, we believe this to be an incident stemming from the previous IT team of the auto domain. Whether deliberate or unintentional, this incident has caused our system to crash, being deemed as either an internal (unfixed misconfiguration from the current IT team) or external (deliberate misconfiguration from the previous IT team) attack, with our team also carefully looking more into system logs before and between the dates of receiving the systems and the incident to find evidence of deliberate tampering.