ISSN 2347 - 3983
Volume 8. No. 4, April 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter06842020.pdf
https://doi.org/10.30534/ijeter/2020/06842020

A Tutorial on PowerShell Pipeline and its Loopholes

Hitesh Mohapatra[1], Subarna Panda[2*], Amiya Kumar Rath[3], SA Edalatpanah[4], Ranjan Kumar[5]
[1] Assistant Professor, Jain Deemed to be University, Bengaluru, India
[2*] Assistant Professor, Jain Deemed to be University, India, subbarna.panda@gmail.com
[3] Professor at VSS University of Technology, Odisha, India
[4] Associate Professor, Department of Industrial Engineering, Ayandegan Institute of Higher Education, Iran
[5] Assistant Professor, Jain Deemed to be University, Bengaluru, India

ABSTRACT

Pipelines are undoubtedly the most blue-chip concept in Windows PowerShell (PS). It creates a significant difference between PowerShell and DOS. The working process of the pipeline is based on several inputs and single outputs. It means that the output of one sub-part becomes the input to the other. Pipeline successively refines the data through several stages for final filtered output. In this paper, we have studied the details of the pipeline and presented a tutorial by considering all the capacities of pipeline in the context of the shell.

Key words: Input, Output, Pipeline, PowerShell, Windows, Shell, Scripting.

1. INTRODUCTION

Information technology (IT) has become an integral part of our life to satisfy the need of society. In the current era, computer science is a major subject. It has many real-life applications such as cloud computing [1], artificial intelligence [2], virtualization environment [3], Internet of things [4,5,6,7,8,9,10,11], transportation problem [12,13], shortest path problem [14,15,16,17,18,19,20,21], internet security [22], uncertainty [23,24,25,26] and so on. IT is the mode by which users can store, fetch, communicate and utilize the information. So, all the organizations, industries and also every individual are using computer systems to preserve and share the information. Internet security appears in many real-life applications, e.g., home security, banking, defence system, education sector, railway and so on.

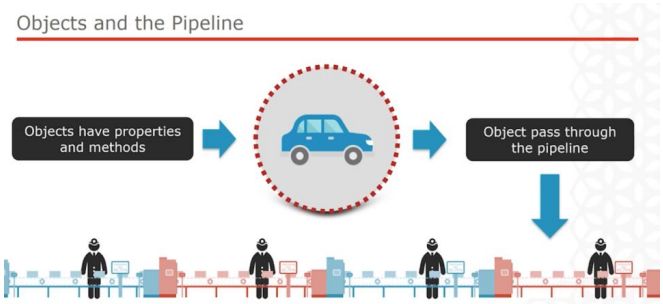

Figure 1: Pipeline Function

In this manuscript, we have discussed the PS concept which is normally used by the administrator to reduce the vulnerabilities and restricting access to the operating system (OS) [27]. Figure 1 illustrates a real-time demonstration of pipeline operations.

The Microsoft PowerShell is a by default installed application in every Windows version. The PS is the integration of command-line shell and scripting language. It works with the dot NET framework. It provides the user interface that allows programmers to interact with OS. The PS provides many benefits to the administrator like the dynamic generation of commands, execution from memory, encoding, and obscuring that helps to make the log files for forensic uses [27].

1.1 PowerShell

In the year of 2006, Microsoft introduced PS for automation and configuration management. The PS is a highly flexible scripting technology for the administrator end. This .NET based framework comprises of two components: one is command shell and the scripting language through which the complex functions of PS like windows management instrumentation (WMI) and component object models (COM) objects. The compilation of PS is happening through the .NET framework which allows working with DLL and assembly functions. These attributes give additional strengths to the PS like remote content downloading, execution of a command from memory, and handling of local registry keys [28].

The pipeline concept can be understood by assuming the in one end and the output comes at the other. This concept is a new thing to the shell environment but the major difference is that in the case of PS the whole object can be passed. The vertical bar (|) is used to denote the pipeline in the syntax environment. In this process, the user may not see the internal process in the pipeline but the user can give a direction of execution through | operator. Consider an example, to understand the functionalities of the pipeline [29,30]. Example 1, presents the *Get-Process* cmdlet. Figure 2 illustrates the actual processes of the system.

Example 1: *Get-Process | Where-Object WorkingSet -gt 500kb | Sort-Object -Descending Name*

These process objects contain several pieces of information like Id, process name, CPU usage, etc. The output of *Get-Process* cmdlets becomes an input to the *Where-Object* cmdlet [31]. The *Where-Object* cmdlet process this input data by applying the filter i.e. find the processes which consume more than 500 KB of memory (Ref: Figure 3). After that, it passes through *Sort-Object* cmdlet by which all the name field is sorted in a descending manner as shown in Figure 2. Example 1 clearly shows the passing of full-fidelity objects along the pipe-line, not their texts, through the pipe-line process. The admin can extract meaningful information from the output data. The overall objective of the pipelining method is to tie multiple instructions with a common thread.

```
PS C:\Windows\system32> Get-Process

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    639      66   140624     165904      32.23    1588   1 Acrobat
    433      33    11924      24956       0.67    2024   1 AcroCEF
    217      28    44992      52276       2.07    3076   1 AcroCEF
     60       8     1544       5496       0.02    2848   1 acrotray
    229      21     4256      10332       0.50    1384   0 AGMService
    121      12     2064       5996       0.05    1428   0 AGSService
     80       8     1380       3724       0.02    1360   0 armsvc
    134      10    18256      18436       0.12    5068   0 audiodg
    188      41    67224      89724       3.63     344   1 chrome
    280      34    86328     122944       2.82    1636   1 chrome
    283      34    89284     127120       3.12    2208   1 chrome
    140       7     2500       6060       0.03    2648   1 chrome
   1685      59   152312     218596      36.19    2904   1 chrome
```

Figure 2: Get-Process Execution

```
PS C:\Windows\system32> $h = dir | group -AsHash -AsString Length

PS C:\Windows\system32> $h

Name                           Value
----                           -----
2651136                        {wuaueng.dll}
976896                         {dui70.dll}
4636160                        {NlsData0816.dll}
89088                          {amstream.dll, iscsiwmi.dll, scext.dll}
866816                         {RASMM.dll}
691200                         {VAN.dll}
196642                         {C_936.NLS, C_949.NLS, C_950.NLS}
131072                         {IEAdvpack.dll, mmcshext.dll}
247808                         {msls31.dll}
3860992                        {UIRibbon.dll}
```

Figure 3: Group and Pivot Data by Name

The PowerShell also helps to group objects based on their properties and attributes (*Ref: Figure 3*). The command -AsHash parameter of the Group-Object cmdlet is an alternative for Where-Object cmdlet. This creates a mapping between the objects with property interested in. The primary limitation of this approach is, it is more efficient on the simple data set rather a complex one [32-33]. The all above examples clear the basic definitions of the pipeline as; it integrates several commands through pipeline operators where not only string can be passed but also the user can pass complex objects that are the strength of PowerShell cmdlets.

The rest of the paper is organized as follows; Section 2 contains the various usages of the pipeline concept, Section 3 explores the parallel existing technologies, Section 4 discusses the various criticality in using the pipeline, and Section 5 concludes the paper followed by references.

2. PIPELINE USES IN REAL-TIME

The pipeline is one of the cohesive concepts in remote server handling. In many real-time applications such as banking system, health care, etc. where a huge amount of data is involved. These applications need to be managed through a central server structure. In such circumstances pipeline helps to handle complex queries.

2.1 Object-Oriented PipeLine

The chaining of command lines is not a new concept, but on the contrary, the pipeline in PowerShell takes the object-oriented (OO) approach for real-time execution.

Example 2: *Dir | Sort-Object Length | Select-Object Name, Length | ConvertTo-Html | Out-File report.htm.\report.htm.*

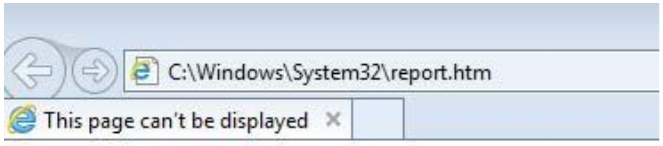

Figure 4: Group and Pivot Data by Name

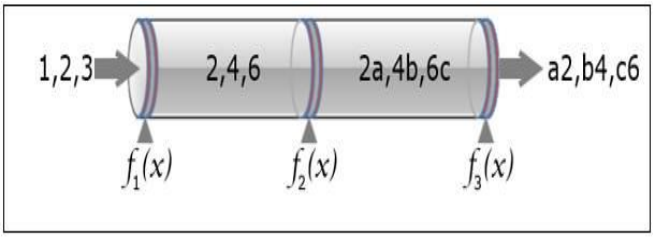

Figure 5: Pipeline Working

The command of Example 2 returns an HTML page on the Windows directory with sorted contents. The command begins with Dir, which passes its results to Sort-Object. These properties constrained the sorted object coverts the objects into an HTML by using *ConvertTo-Html* cmdlets.

This example demonstrates the multiple uses of pipelines in the OO-environment. In the said example the intermediate results of pipelines are very rich whereas the final output is reduced one. Figure 4 illustrates the object-oriented execution [34]. Figure 5 illustrates the internal working mechanism of the pipeline. Figure.6 illustrates the basic flow of the

object-oriented approach where the standard output of a command is standard input to the other command. Figure 7 illustrates the cmdlets environment of pipeline operations. Figure 8 illustrates the object transmission over the pipeline in PowerShell.

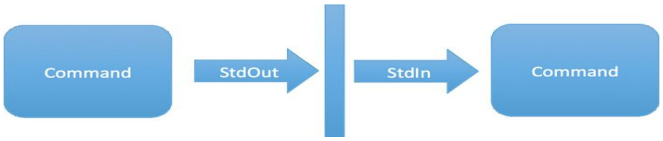

Figure 6: Standout and Standard in Over Pipeline

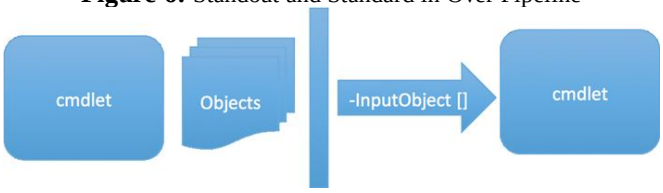

Figure 7: Working of Pipeline

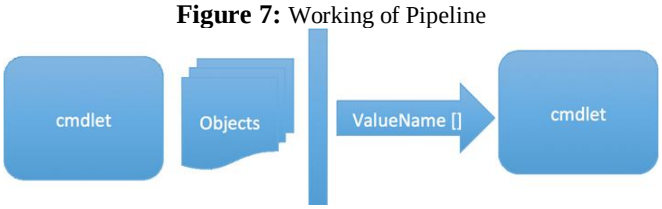

Figure 8: Working of Pipeline

2.2 Example

The use of pipeline brings a notable advantage like conserve memory. In the process of memory conservation, the pipeline concept helps to break the whole huge file into memory which helps to read the file concurrently. Because without the pipeline concept it becomes difficult to have such a huge memory to read the total file [35]. The second advantage of the pipeline is the concurrent execution of several tasks through a single processor. It means when one process working upon reading operation with a huge file, the other process of pipeline helps to do another module of the same work. The third advantage of the pipeline is to enhance the end-user experience. It means that if the task is about to take 1 minute then in traditional execution the end-user has to wait for 1 minute to get the result whereas, in under pipelining execution the user may start getting output after a few second intervals. This helps to enhance the user experience. Figure 8 clearly shows that the functions or the filters transform the fed input into various intermediary results. The performed operations are deliberately simple and nonsensical. The following commands presents:

```
# double the input
Function f1($x) {$x * 2}

# concatenate the nth letter to the input where n is half
the input values
Function f2($x) {“$x" + [char]([byte][char] "A" + $x/2 - 1)}

# reverse the 2-character string
Function f3($x) {$x [1...0] -join '}
```

3. SCENARIOS OF PIPELINE EXECUTION

This section presents various scenarios of pipeline execution. The scenarios are

3.1 Pipeline with Individual Output

In this case, we just loop the pipeline data, calculating the output, and displaying the output. Even if the data flows smoothly through the pipeline, it dries up the pipeline completely then only it forwards the data to the next pipeline. Example 3 clearly illustrates the execution of the pipeline through completely dries up. In this Example 3, it dries up the entire line input until the pipeline is empty.

Example: 3

```
function showInputs
{
   $output = @ ()
   foreach ($data in $input) {$output += $data}
   Write-Host $output
   Write-Output $output
}
function f1Runner
{
   $output = @ ()
   foreach ($data in $input) {$output += f1 $data}
Write-Host $output
   Write-Output $output
}

function f2Runner
{
   $output = @ ()
   foreach ($data in $input) {$output += f2 $data}
   Write-Host $output
   Write-Output $output
}
function f3Runner
{
   $output = @ ()
   foreach ($data in $input) {$output += f3 $data}
   #Write-Host $output
   Write-Output $output
}}
```

3.1 Pipeline with Individual Output

The previous Example.3 can be fixed by emitting each value as soon as it gets calculated. In the Example.4, emitting value at a time. (Ref: Figure 9)

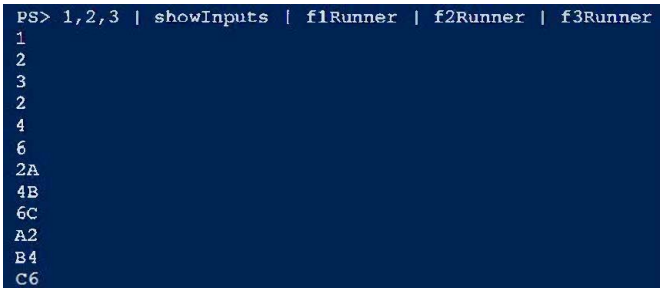

Figure 9: Output without Pipeline

Example: 4

```
Function showInputs
{
   $input | ForEach-Object {
      $output = $_
      Write-Host $output
      Write-Output $output
   }
}
Function f1Runner
{
   $input | ForEach-Object {
      $output = f1 $_
      Write-Host $output
      Write-Output $output
   }
}
Function f2Runner
{
   $input | ForEach-Object {
      $output = f2 $_
      Write-Host $output
      Write-Output $output
   }
}
Function f3Runner
{
   $input | ForEach-Object {
      $output = f3 $_
      # Write-Host $output
      Write-Output $output
   }
}
```

In the given output (*Ref: Figure 9*) the output is not being pipelined. Here the declared functions emitting results. The example explicitly shows that it is not aggravating the results.

3.1 Process Inputs Promptly After Receiving

In this said Ex. 5, the process block runs for each input without loop. It does not mean that the execution happens without loop but that loop management is done by the PowerShell itself. *$_* is used to access the current value. (Ref: Figure 10).

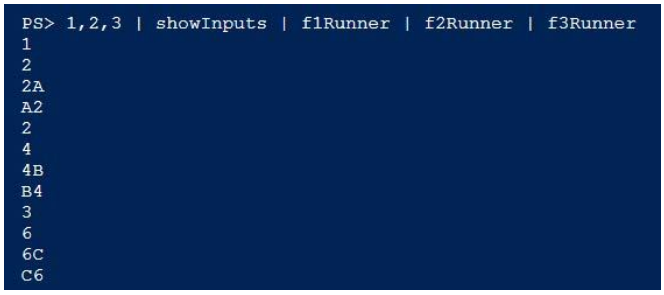

Figure 10: Output with Pipeline

Example: 5

```
Function show Inputs
{
   process
   {
      $output = $_
      Write-Host $output
      Write-Output $output
   }
}
Function f1Runner
{
   process
   {
      $output = f1 $_
      Write-Host $output
      Write-Output $output
   }
}
Function f2Runner
{
   process
   {
      $output = f2 $_
      Write-Host $output
      Write-Output $output
   }
}
Function f3Runner
{
   process
   {
      $output = f3 $_
      # Write-Host $output
      Write-Output $output
```

Figure 11 demonstrates the output which is obtained through by using the pipeline concept of PowerShell. These previously given instances explain the reality of pipeline uses, it means that user expectation typically behaves in two ways, either it waits to get all the inputs before processing or it processes all given inputs before sending output.

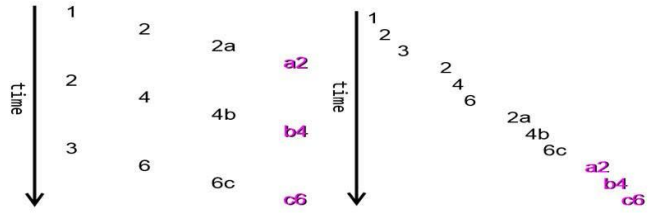

Figure 11: Pipeline Process

```
5, 6, 7, 2, 3, 0, 9 |% -begin {$v = @ ()} {$v += $_} -end {[array]: Sort($v); $v}
```

So, to avoid this waiting state from the end-user end, the pipelining in PowerShell is the key to that. For example, sorting is the best example of pipelining operations [36].

4. LOOPHOLES WITH POWERSHELL

The PS provides easy accessibility to the major components of the work station like access to the OS, server, etc. The PS is vulnerable to the attacks for the following reasons:

- All Windows by default installed with PS.
- The payloads can be executed directly from memory.
- PS generates traces which creates difficulties to track attacks.
- It has remote access capabilities.
- The script part of this creates obfuscation and traditional security tools are fails to catch it.
- Defenders often overlook it during hardening their computer.
- It has the capacity to bypass application-whistling tools.
- Gateways are not enough capable to handle script-based malware.
- The ready availability of scripts.
- Permission from the administrator end for working with PS malware.

These causes make the PS most vulnerable. In the next sub-section, we have presented a few attacks on PS.

4.1 Powershell Sensor Issues

According to the [37] report, the PS mostly used by the IT people to restrict access and to reduce the vulnerabilities. The major weakness with PS is, it invokes external commands and modules for dynamic defined environment variables. This short of execution does not produce visibility for detecting attacks on PS. One such instance of PS attack is "IEX \env:iu7Gt" environmental behaviour. This malicious script invokes a log that shows the command before its dynamic interpretation [13]. In PS 5.0 the Microsoft improved the logging capabilities by introducing Anti-Malware Scan Interface (AMSI) [14] These malicious activities can be handled through many ways one of those is deep neural network-based attack handling. According to the report published by the [37] in 2016, 49,127 PowerShell scripts were submitted to the Symantec. The deep analysis of those scripts concludes that, out 100%, 95.4% of scripts are malware. Through the manual investigation of these scripts by the Symantec, brings out 4,782 recent distinct samples of malware. This malware belongs to 111 malware families. The deep study also says that the malware "W97M" is the most prevalent malware of 9.4%, "Kotver" came second with 4.5% and the "JS. Downloader" came third with 4%. Figure 12, illustrates the malicious PS script of the year 2016.

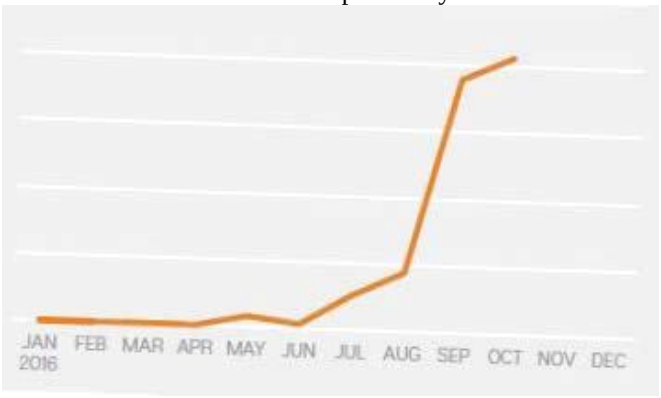

Figure 12: Malware Based Scripts in PS in the Year 2016

4.2 Execution Policy

In this sub-section, we have put light on the various phases of the PS attack and its consequences. Microsoft has set a few policies for PS script execution. By default, in every window there are five options are available such as *Restricted, AllSigned, Bypass, RemoteSigned, Unrestricted.*

These policies are not providing any security to the system rather it prevents the user from the accidental running of scripts. The default policy that is set is *Restricted* except Windows Server 2012 R2 where it is *RemoteSigned*. Organizations select these policies based on several scopes like Machine Policy, User Policy, Process, Current User or Local Machine. Despite all these policies, the attackers have several methods for attacks. The few notes methods are:

Pipe the script into the standard-in of powershell.exe, such as with the echo or type command.

```
TYPE myScript.ps1 | PowerShell.exe -noprofile -
```

Use the command argument to execute a single command

```
powershell.exe -command "iex(New-Object Net.WebClient).DownloadString('http://[REMOVED]/myScript.ps1')"
```

Use the Encoded Command argument to execute a single Base64-encoded command.

```
powershell.exe -enc [ENCODED COMMAND]
```

Use the execution policy directive and pass either "bypass" or "unrestricted" as argument.

```
powershell.exe -ExecutionPolicy bypass -File myScript.ps1
```

These problems with PS may invite the interferences of attackers to modify the interactive PS sessions where the attackers can run the codes on a compromised computer.

4.3 Script Execution

Normally, in the PS environment, the Restricted mode does not allow the attacker to execute the malicious code whereas, the attackers use the following methods to evade detection and to bypass the local restrictions.

1 -NoP / -NoProfile (ignore the commands in the profile file)

2 -Enc / -EncodedCommand (run a Base64-encoded command)

3 -W Hidden / -WindowStyle Hidden (hide the command window)

4 -Exec bypass / -ExecutionPolicy Bypass

5 -NonI / -NonInteractive (do not run an interactive shell)

6 -C / -Command (run a single command)

7 -F / -File (run commands from a specified file)

In malicious PS script, there are few commands which are frequently used such as;

1 (New-Object System.Net.Webclient).DownloadString()

2 (New-Object System.Net.Webclient).DownloadFile()

3 -IEX / -Invoke-Expression

4 -Start-Process

4.3 Email Vector

Email is one of the primary commuters for deliver vector for PS. The attacker sends the malicious codes in the *.zip* formats. These malicious files have following extensions:

.lnk, .wsf (Windows Script file), .hta, .mhtml, .html, .doc, .docm, .xls, .xlsm, .ppt, .pptm, .chm (compiled HTML help file), .vbs (Visual Basic script), .js (JavaScript), .bat, .pif, .pdf, .jar

Upon receiving such files, the injected malicious PS script runs aromatically. The report produced by [38] says that during their investigation they are blocking around 466,028 malicious emails per day. These malicious JavaScript-based attachments mostly have the extensions of .html, .vbs and .doc type.

5. CONCLUSION

In this paper, we have exploited the PowerShell concept in Windows environment. The first part of the paper has explored the advantages and limitations of the pipeline concept. In the second part, we have concentrated on the loopholes of PS based on possible attacks. It focuses on assault bunches that utilizes PowerShell. There are immense networking contents for PowerShell entrance analyzers. It can be predicted that cybercriminals may begin to utilize the PowerShell in the later stages. Additionally, it is conceivable to have full secondary passage Trojans or ransomware coded in PowerShell. This paper can be extended by analyzing several real-time attacks through PS in advanced environments.

This paper in prospect to future advancement would concentrate upon an open and ubiquitous cross-platform of PS in core support and adaptation in the cloud, REST or any first-party scenarios. It would also enable in implantation of large and trusted artefacts such as Azure compliance requirements, Azure-based power shell services pertinent to the hybrid cloud. Availability of ConvertFrom-String cmdlet would be a boost to the administrator to convert the output into objects in Linux, macOS commands for the second-class citizens in the PowerShell world.

REFERENCES

1. BWK Malubaya and G Wang, **Real-time parking information system with cloud computing open architecture approach**, *International Journal of Emerging Trends in Engineering Research*, vol. 8, pp. 18-22, 2020 https://doi.org/10.30534/ijeter/2020/04812020
2. DA Petrosov, RA Vashchenko, AA Stepovoi and NV Petrosova, **Application of artificial neural networks in genetic algorithm control problems**, International Journal of Emerging Trends in Engineering Research, vol. 8, pp. 177-181, 2020. https://doi.org/10.30534/ijeter/2020/24812020
3. H Mohapatra, "**HCR using neural network**," Biju Patnaik University of Technology, PhD dissertation 2009.
4. H Mohapatra and AK Rath, "**Detection and avoidance of water loss through municipality taps in India by using smart tap and ICT**, *IET Wireless Sensor Systems*, vol. 9, pp. 447-457, 2019 https://doi.org/10.1049/iet-wss.2019.0081
5. H Mohapatra and AK Rath, **Fault tolerance in WSN through PE-LEACH protocol**," *IET Wireless Sensor Systems*, vol. 9, pp. 358-365, 2019
6. H Mohapatra, S Debnath, and AK Rath, **Energy management in wireless sensor network through EB-LEACH**, *International Journal of Research and Analytical Reviews (IJRAR)*, pp. 56-61, 2019.
7. VN. Nirgude, H Mahapatra, and SA. Shivarkar **Face recognition system using principal component analysis & linear discriminant analysis method simultaneously with 3d morphable model and neural network BPNN method**, *Global Journal of Advanced Engineering Technologies and Sciences*, vol. 4, pp. 1-6, 2017.
8. M Panda, P Pradhan, H Mohapatra, and NK Barpanda, **Fault-tolerant routing in a heterogeneous environment**, *International Journal of Scientific & Technology Research*, vol. 8, pp. 1009-1013, 2019.
9. H Mohapatra, and AK Rath, **Fault-tolerant mechanism for wireless sensor network**, *IET Wireless Sensor Systems*, vol. 10, pp. 23-30, 2020. https://doi.org/10.1049/iet-wss.2019.0106

10. D. Swain, G. Ramkrishna, H. Mahapatra, P. Patra, and PM. Dhandrao, **A novel sorting technique to sort elements in ascending order**, *International Journal of Engineering and Advanced Technology*, vol. 3, pp. 212-126, 2013.
11. R Kumar, SA Edalatpanah, S Jha, and R Singh, **A Pythagorean fuzzy approach to the transportation problem**, *Complex and Intelligent System*, vol. 5, pp. 255-263, 2019. https://doi.org/10.1007/s40747-019-0108-1
12. J Pratihar, R Kumar, A Dey, and S Broumi, **Transportation problem in neutrosophic environment**, in *Neutrosophic Graph Theory and Algorithms*, F. Smarandache and S. Broumi, Eds.: IGI-Global, 2019, ch. 7, pp. 176-208.
13. S Broumi, A Dey, M Talea, A Bakali, F Smarandache, D Nagarajan, M Lathamaheswari and R Kumar, **Shortest path problem using Bellman algorithm under neutrosophic environment**, *Complex & Intelligent Systems*, vol. 5, pp. 409-414, 2019.
14. R Kumar, SA Edalatpanah, S. Jha, S Broumi, R Singh, and A Dey, **A multi-objective programming approach to solve integer-valued neutrosophic shortest path problems**, *Neutrosophic Sets and Systems*, vol. 24, pp. 134-149, 2019.
15. R Kumar, A Dey, F Smarandache, and S Broumi, "**A study of neutrosophic shortest path problem**," in *Neutrosophic Graph Theory and Algorithms*, F. Smarandache and S. Broumi, Eds.: IGI-Global, 2019, ch. 6, pp. 144-175. https://doi.org/10.4018/978-1-7998-1313-2
16. R Kumar, SA Edalatpanah, S Jha, and R Singh, **A novel approach to solve gaussian valued neutrosophic shortest path problems**, *Int J Eng Adv Technol*, vol. 8, pp. 347-353, 2019b.
17. R Kumar, SA Edalatpanah, S Jha, S Gayen, and R Singh, **Shortest path problems using fuzzy weighted arc length**, *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, pp. 724-731, 2019.
18. R Kumar, SA Edaltpanah, S Jha, S Broumi, and A Dey, **Neutrosophic shortest path problem**, *Neutrosophic Sets and Systems*, vol. 23, pp. 5-15, 2018.
19. R Kumar, S Jha, and R Singh, **A different approach for solving the shortest path problem under mixed fuzzy environment**, *International Journal of fuzzy system Applications*, vol. 9, pp. pp.132-161, 2020.
20. R Kumar, S Jha, and R Singh, **Shortest path problem in network with type-2 triangular fuzzy arc length**, *Journal of Applied Research on Industrial Engineering*, vol. 4, pp. 1-7, 2017.
21. J Sakhnini, H Karimipour, A Dehghantanha, RM Parizi, and G Srivastava, **Security aspects of Internet of Things aided smart grids: A bibliometric survey**, *Internet of Things*, pp. 100-111, 2019. http://www.sciencedirect.com/science/article/pii/S2542660519302148 https://doi.org/10.1016/j.iot.2019.100111
22. S Gayen, F Smarandache, S Jha, and R Kumar, **Interval-valued neutrosophic subgroup based on interval-valued triple t-norm**, in *Neutrosophic Sets in Decision Analysis and Operations Research*, M. Abdel-Basset and F. Smarandache, Eds.: IGI-Global, Dec. 2019c, ch. 10, p. 300. https://doi.org/10.4018/978-1-7998-2555-5.ch010
23. S Gayen, F Smarandache, S Jha, MK Singh, S Broumi and R Kumar, **Introduction to Plithogenic Subgroup**, in *Neutrosophic Graph Theory and Algorithm*, F Smarandache and S Broumi, Eds.: IGI-Global, 2020, ch. 8, pp. 209-233. https://doi.org/10.4018/978-1-7998-1313-2.ch008
24. S Gayen, S Jha, M Singh, and R Kumar, **On a generalized notion of anti-fuzzy subgroup and some characterizations**, *International Journal of Engineering and Advanced Technology*, vol. 8, pp. 385-390, 2019.
25. V Lyashenko, SK Mustafa, I Tvoroshenko and MA Ahmad, **Methods of using fuzzy interval logic during processing of space states of complex biophysical objects**, *International Journal of Emerging Trends in Engineering, vol. 8, pp. 372-377, 2020* . https://doi.org/10.30534/ijeter/2020/22822020
26. D Hendler, S Kels, and A Rubin **Detecting malicious powershell commands using deep neural networks**, in *Proceedings of 2018 on Asia Conference on Computer and Communications Security*, New York, NY, USA, 2018, pp. 187–197. https://doi.org/10.1145/3196494.3196511
27. D Ugarte, D Maiorca, F Cara, and G Giacinto, **PowerDrive: accurate deobfuscation and analysis of powershell malware**, in *Detection of Intrusions and Malware, and Vulnerability Assessment*, Cham, 2019, pp. 240-259.
28. A Rubin, S Kels, and D Hendler, **AMSI-Based detection of malicious powershell code using contextual embeddings**, pp.1-17, May 2019., https:// arxiv.org/ pdf/1905.09538.pdf
29. A Rousseau, **Hijacking.NET to defend Powershell**, Sep. 2017.
30. G Rusak, A Al-Dujaili, and UM O'Reilly **AST-Based deep learning for detecting malicious Powershell**, Oct. 2018. https://doi.org/10.1145/3243734.3278496
31. X Sun, J Lin, and B Bischl, **ReinBo: machine learning pipeline search and configuration with bayesian optimization embedded reinforcement learning**, pp. 1-17, Apr. 2019. https://arxiv.org/abs/ 1904.05381
32. A Hamidinekoo, E Denton, and R Zwiggelaar, **Automated mammogram analysis with a deep**

learning pipeline, July 2019.

33. J Gao, Z Yang, and R Nevatia, **Cascaded boundary regression for temporal action detection**, May 2017. https://doi.org/10.5244/C.31.52
34. K Papaioannou, M Theobald, and M Böhlen, **Generalized lineage-aware temporal windows: supporting outer and anti joins in temporal-probabilistic databases**, Feb. 2019.
35. W Fuhl, T Santini, G Kasneci, and E Kasneci, **PupilNet: convolutional neural networks for robust pupil detection**, Jan. 2016.
36. H Deshev, PowerShell Community Extensions," in ***Pro Windows PowerShell*. Berkeley, CA: Apress**, 2008, pp. 417-448. https://doi.org/10.1007/978-1-4302-0546-3_21
37. S Talaat, **Getting started with azure powershell**, in *Pro PowerShell for Microsoft Azure*. Berkeley, CA: Apress, 2015, pp. 9-17. https://doi.org/10.1007/978-1-4842-0665-2_2
38. X Xu, **From cloud computing to cloud manufacturing**, *Robotics and Computer-Integrated Manufacturing*, vol. 28, pp. 75-86, 2012. http://www.sciencedirect.com /science/article/pii/ S0736584511000949 https://doi.org/10.1016/j.rcim.2011.07.002