Abstract: Each and every operating system has its own partition utility like parted for Unix based operating systems, disk management utility for Windows operating system and parted Util for ESX operating system. A partition utility for a particular operating system can able to detect only certain disk labels of other operating systems [1]. We need to develop a Utility by named partition table analyzer utility which has capability of analyzing a partition table if there is any partition table already exist on the disk. This utility is used to avoid data corruption in cloud environment.

Keywords: Cloud Operating System, Disk, Partition table, Partition Utility.

I. INTRODUCTION

To organize user’s data more effectively all operating systems allow users to divide a hard disk into multiple partitions in order, making one physical hard disk a defined storage space on a hard drive into several smaller logical hard disks. Each and every hard disk partition stores many file systems, in turn data is stored in each file system. All the information about partitions is used to store in the Partition table. The beginning sectors in a hard disk have a reserved area to store Partition table [2]. In creating, destroying, resizing, copying and checking partitions and file systems on them we use partition utility.

II. MOTIVATION

The Cloud operating system manages several storage resources and schedules the computing resources in a cluster of servers span across multiple data centers at geographically distributed locations [4][6][7]. When a system with particular operating system in cloud demands a storage resource for cloud, suppose cloud serves the storage resource which has been already a partition table with a disk label to the requested system [5]. If the requested system’s partition utility cannot recognize the already existing partition table, it may lead to data corruption.

So there is a need to develop a Utility by named partition table analyzer utility which has capability of analyzing any partition table, if it already exists on the disk, to avoid data corruption in cloud environment. To avoid data corruption we have to develop a partition analyzer utility to which we provide Disk with certain disk-label and partition table and results in displaying Disk-label and contents in partition table.
III. ARCHITECTURE FOR PARTITION TABLE GENERATOR UTILITY

![Diagram](https://via.placeholder.com/150)

Fig. 1 Partition Table Generator Utility Architecture

IV. GUID PARTITION TABLE (GTP) ANALYZER ALGORITHM

Algorithm for analyzing Guid Partition Table

**Input:** Disk-name with Guid Partition table  
**Output:** Analyzing Guid Partition table

Algorithm for analyzing Guid Partition table  
**Begin**

1. `Filedescriptor1 ↷ open(Disk-name, O_RDWR)`
2. `read(Filedescriptor1, buffer, 34 * 512)`
3. `sudoMbr ← (Partition_MBR *) (buffer + 446)`
4. `print sudoMbr[0].type`
5. `print sudoMbr[1].type`
6. `print sudoMbr[2].type`
7. `print sudoMbr[3].type`
8. `gptHeader ← (Partition_GptHeader *) (buffer + 512)`
9. `signature ← 0x5452415020494645`
10. `if gptHeader→signature equals signature`
11. `print valid Signature`
12. `else print invalid Signature`
13. `revision ← 0x00010000`
14. `if gptHeader→revision equals revision`
15. `print valid Revision`
16. `else print invalid Revision`
17. `headerSize ← 0x5C`
18. `if gptHeader→headerSize equals headerSize`
19. `print valid HeaderSize`
20. `else print invalid HeaderSize`
21. `reserved1 ← 0x0`
22. `if gptHeader→reserved1 equals reserved1`
23. `print valid Reserved1`
24. `else print invalid Reserved1`
25. `myLba ← 0x1`
26. `if gptHeader→myLba equals myLba`
27. `print valid MyLba Address`
28. `else print invalid MyLba Address`
29. `alternateLba ← ( gptHeader→lastUsableLba+33)`
30. `if gptHeader→alternateLba equals alternateLba`
31. `print valid AlternateLba Address`
32. `else print invalid AlternateLba Address`
33. `firstUsableLba ← 0x22`
34. `if gptHeader→firstUsableLba equals firstUsableLba`
35. `print valid FirstUsableLba Address`
36. `else print invalid FirstUsableLba Address`
37. `partitionEntryLba ← 0x2`
38. `if gptHeader→partitionEntryLba equals partitionEntryLba`
39. `print valid PartitionEntryLba Address`  
**End**
else
    print invalid PartitionEntryLba Address
numberOfPartitionEntries ← 0x80
if gptHeader→numberOfPartitionEntries equals numberOfPartitionEntries
    print valid Number Of Partition Entries
else
    print invalid Number Of Partition Entries
sizeofPartitionEntry ← 0x80
if gptHeader→sizeofPartitionEntry equals sizeofPartitionEntry
    print valid Size of Partition Entry
else
    print invalid Size of Partition Entry
temp ← gptHeader→headerCrc32
gptHeader→headerCrc32 = 0x0
calcCrc ← EFI_crc32(gptHeader, gptHeader→headerSize)
if calcCrc equals temp
    print valid HeaderCrc32 in Primary Partition Table
else
    print invalid HeaderCrc32 in Primary Partition Table
header
Filedescriptor1 ← close(Filedescriptor1)
Filedescriptor2 ← open(Disk-name, O_RDWR)
lseek(Filedescriptor2, 512*gptHeader→alternateLba, SEEK_SET)
read(Filedescriptor2, buffer1, 512)
gptHeader1 ← (Secondary_Partition_GptHeader *)(buffer1)
signature ← 0x5452415020494645
if gptHeader1→signature equals signature
    print valid Signature
else
    print invalid Signature
revision ← 0x00010000
if gptHeader1→revision equals revision
    print valid Revision
else
    print invalid Revision
headerSize ← 0x5C
if gptHeader1→headerSize equals headerSize
    print valid HeaderSize
else
    print invalid HeaderSize
reserved1 ← 0x0
if gptHeader1→reserved1 equals reserved1
    print valid Reserved1
else
    print invalid Reserved1
temp ← gptHeader→lastUsableLba
if gptHeader1→lastUsableLba equals temp
    print valid LastUsableLba Address in
    Secondary Partition header
else
    print invalid LastUsableLba Address in
    Secondary Partition header
alternateLba ← 0x1
if gptHeader1→alternateLba equals alternateLba
    print valid AlternateLba Address in
    Secondary Partition header
else
    print invalid AlternateLba Address in
    Secondary Partition header
myLba ← (gptHeader1→lastUsableLba+33)
if gptHeader1→myLba equals myLba
   print valid MyLba Address in Secondary Partition header
else
   print invalid MyLba Address in Secondary Partition header
firstUsableLba ←0x22
if gptHeader1→firstUsableLba equals firstUsableLba
   print valid FirstUsableLba Address
else
   print invalid FirstUsableLba Address
partitionEntryLba ←0x2
if gptHeader1→partitionEntryLba equals partitionEntryLba
   print valid PartitionEntryLba Address
else
   print invalid PartitionEntryLba Address
numberOfPartitionEntries ←0x80
if gptHeader1→numberOfPartitionEntries equals numberOfPartitionEntries
   print valid Number Of Partition Entries
else
   print invalid Number Of Partition Entries
sizeofPartitionEntry ←0x80
if gptHeader1→sizeofPartitionEntry equals sizeofPartitionEntry
   print valid Size of Partition Entry
else
   print invalid Size of Partition Entry
temp ←gptHeader1→headerCrc32
gptHeader1→headerCrc32=0x0
calcCrc ←efi_crc32(gptHeader1, gptHeader1→headerSize)
if calcCrc equals temp
   print valid HeaderCrc32 in Secondary Partition header
else
   print invalid HeaderCrc32 in Secondary Partition header
close(Filedescriptor2)
End

Fig 2: Screen shot for creating GUID Partition Table
TABLE I: GUID Partition Table Contents

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Byte Offset</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>End</td>
<td>Last</td>
<td>Identifies EFI-compatible partition table header. This value must contain the string “EFI PART”, 0x5452415020496465.</td>
</tr>
<tr>
<td>Revision</td>
<td>8</td>
<td>4</td>
<td>The specification revision number that this header complies to. For version 1.0 of the specification the correct value is 0x00010000.</td>
</tr>
<tr>
<td>HeaderSize</td>
<td>12</td>
<td>4</td>
<td>Size in bytes of the GUID Partition Table Header.</td>
</tr>
<tr>
<td>HeaderCRC32</td>
<td>16</td>
<td>4</td>
<td>CRC32 checksum for the GUID Partition Table Header structure. The ranged defined by HeaderSize is “check-summed”.</td>
</tr>
<tr>
<td>Reserved</td>
<td>20</td>
<td>4</td>
<td>Must be Zero.</td>
</tr>
<tr>
<td>MyLBA</td>
<td>24</td>
<td>8</td>
<td>The LBA that contains this data structure.</td>
</tr>
<tr>
<td>AlternateLBA</td>
<td>32</td>
<td>8</td>
<td>LBA address of the alternate GUID Partition Table Header.</td>
</tr>
<tr>
<td>FirstUsableLBA</td>
<td>40</td>
<td>8</td>
<td>The first usable logical block that may be contained in a GUID Partition Entry.</td>
</tr>
<tr>
<td>LastUsableLBA</td>
<td>48</td>
<td>8</td>
<td>The last usable logical block that may be contained in a GUID Partition Entry.</td>
</tr>
<tr>
<td>DiskGUID</td>
<td>56</td>
<td>16</td>
<td>GUID that can be used to uniquely identify the disk.</td>
</tr>
<tr>
<td>PartitionEntryLBA</td>
<td>72</td>
<td>8</td>
<td>The starting LBA of the GUID Partition Entry array.</td>
</tr>
<tr>
<td>NumberOfPartitionEntries</td>
<td>80</td>
<td>4</td>
<td>The number of Partition Entries in the GUID Partition Entry array.</td>
</tr>
<tr>
<td>SizeOfPartitionEntry</td>
<td>84</td>
<td>4</td>
<td>The size, in bytes, of each the GUID Partition Entry structures in the GUID Partition Entry array. Must be a multiple of 8.</td>
</tr>
<tr>
<td>PartitionEntryArrayCRC32</td>
<td>88</td>
<td>4</td>
<td>The CRC32 of the GUID Partition Entry array. Starts at Partition Entry LBA and is (NumberOfPartitionEntries)* (SizeOfPartitionEntry in byte length.)</td>
</tr>
<tr>
<td>Reserved</td>
<td>92</td>
<td>92</td>
<td>The rest of the block is reserved by EFI and must be zero.</td>
</tr>
</tbody>
</table>
V. MASTER BOOT RECORD PARTITION (MBR) ANALYZER ALGORITHM

Algorithm for analyzing Master Boot Record Partition Table

Input: Disk-name with MBR Partition table
Output: Analyzing MBR Partition table

Algorithm for analyzing MBR Partition table
Begin
    Char buffer[512]
    Filedescriptor1 ← open(Disk-name, O_RDWR)
    MSDOS_LABEL_MAGIC ← 0xAA55
    label ← (unsigned short *)(buffer + 446)
    if *label equals MSDOS_LABEL_MAGIC
        print MSDOS label found
    else
        print MSDOS label not found
    SCSI_PTABLE_SECTOR_OFFSET
    p ← (sector) ((Partition_MBR *)((uint8 *)(sector) + SCSI_PTABLE_SECTOR_OFFSET))
    for i=0 to 4
        if(p→boot_ind==0x80) then
            print STATUS IS BOOTABLE(ACTIVE)
        elseif(p→boot_ind==0x00) then
            print STATUS IS NON-BOOTABLE(INACTIVE)
        else
            print STATUS IS INVALID
    end if then
End
if(p\rightarrow type==0x0f) 
  print PARTITION TYPE IS EXTENDED
  PARTITION WITH LBA
elsif(p\rightarrow type==0x05) 
  print PARTITION TYPE IS EXTENDED
  PARTITION WITH CHS ADDRESS
elsif(p\rightarrow type==0x00) 
  print EMPTY PARTITION
else 
  print PARTITION TYPE IS PRIMARY PARTITION
end if then

FirstSector\leftarrow p\rightarrow firstSector
NumSectors\leftarrow p\rightarrow numSectors
StartingLba\leftarrow p\rightarrow firstSector
EndingLba\leftarrow (p\rightarrow firstSector+ p\rightarrow numSectors)
print FirstSector, NumSectors, StartingLba, EndingLba
p++
end i-loop

end

close(Filedescriptor1)

End

Fig 5: A Basic DOS disk with two partitions and one MBR

Fig 6: A DOS disk with three primary file system partitions and one primary secondary partition.

TABLE III: MBR Partition Table Contents [3]

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Byte Offset</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BootCode</td>
<td>0</td>
<td>440</td>
<td>Code used on legacy Intel architecture system to select a partition record and load the first block (sector) of the partition pointed to by the partition record. This code is not executed on EFI systems.</td>
</tr>
<tr>
<td>uniqueMBR Signature</td>
<td>440</td>
<td>4</td>
<td>Unique Disk Signature, this is an optional feature and not on all hard drives. This value is always written by the OS and is never written by EFI firmware.</td>
</tr>
<tr>
<td>Unknown</td>
<td>444</td>
<td>2</td>
<td>Unknown</td>
</tr>
<tr>
<td>Partition Record</td>
<td>446</td>
<td>16*4</td>
<td>Array of four MBR partition records.</td>
</tr>
<tr>
<td>Signature</td>
<td>510</td>
<td>2</td>
<td>Must be 0xaa55.</td>
</tr>
</tbody>
</table>
Fig 7: Screen shot for creating MBR Partition Table

Fig 8: Nesting of Legacy MBR Partition Table [3]
Algorithm for analyzing MAC Partition Table

Input: Disk-name with MAC Partition table
Output: Analyzing MAC Partition table

Algorithm for Analyzing MAC Partition table
Begin
Char buffer[4096]
Filedescriptor1 ← open(Disk-name, O_RDWR)
q ← (MacRawDisk *) (buffer + 0)
print q → signature, q → block_size, q → block_count
for i = 512 to 8 * 512
  p ← (MacRawPartition *) (buffer + i)
if (p → signature == 0x4d50) then
  print p → signature, p → res1, p → map_count
  for i = 0 to 31
    print p → name[i], p → type[i]
  print p → data_start, p → data_count
  if (p → status == 0x00000001)
    print ‘Entry is valid’
  elseif (p → status == 0x00000002)
    print ‘Entry is allocated’
  elseif (p → status == 0x00000004)
    print ‘Entry in use’
  elseif (p → status == 0x00000008)
    print ‘Entry contains boot information’
  elseif (p → status == 0x00000010)
    print ‘Partition is readable’
  elseif (p → status == 0x00000020)
    print ‘Partition is writable’
  elseif (p → status == 0x00000040)
    print ‘Boot code is position independent’
  elseif (p → status == 0x00000100)
    print ‘Partition contains chain-compatible driver’
  elseif (p → status == 0x00000200)
    print ‘Partition contains a real driver’
  elseif (p → status == 0x00000400)
    print ‘Partition contains a chain driver’
  elseif (p → status == 0x40000000)
    print ‘Automatically mount at startup’
else (p → status == 0x80000000)
  print ‘The startup partition’
else
  print ‘Reserved’
print p → boot_start,
  p → boot_count, p → boot_load, boot_load2
print p → boot_entry, p → boot_entry2, p → boot_cksum
for i = 0 to 16
  print p → processor[i]
print p → driver_sig
else
  print ‘Analysis is finished’
i ← i + 512
end i-loop
close(Filedescriptor1)
End
<table>
<thead>
<tr>
<th>Byte Range</th>
<th>Description</th>
<th>Essential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Signature Value (0x504D)</td>
<td>NO</td>
</tr>
<tr>
<td>2-3</td>
<td>Reserved</td>
<td>NO</td>
</tr>
<tr>
<td>4-7</td>
<td>Total Number of partitions</td>
<td>YES</td>
</tr>
<tr>
<td>8-11</td>
<td>Starting sector of partition</td>
<td>YES</td>
</tr>
<tr>
<td>12-15</td>
<td>Size of partition in sectors</td>
<td>YES</td>
</tr>
<tr>
<td>16-47</td>
<td>Name of partition in ASCII</td>
<td>NO</td>
</tr>
<tr>
<td>48-79</td>
<td>Type of partition in ASCII</td>
<td>NO</td>
</tr>
<tr>
<td>80-83</td>
<td>Starting sector of data area in partition</td>
<td>NO</td>
</tr>
<tr>
<td>84-87</td>
<td>Size of data area in sectors</td>
<td>NO</td>
</tr>
<tr>
<td>88-91</td>
<td>Status of partition</td>
<td>NO</td>
</tr>
<tr>
<td>92-95</td>
<td>Starting sector of boot code</td>
<td>NO</td>
</tr>
<tr>
<td>96-99</td>
<td>Size of boot code in sectors</td>
<td>NO</td>
</tr>
<tr>
<td>100-103</td>
<td>Address of boot loader code</td>
<td>NO</td>
</tr>
<tr>
<td>104-107</td>
<td>Reserved</td>
<td>NO</td>
</tr>
<tr>
<td>108-111</td>
<td>Boot code entry point</td>
<td>NO</td>
</tr>
</tbody>
</table>
Fig 9: Screen shot for creating MAC Partition Table

Fig 10: An Apple disk with one partition map partition and three file system partitions.
Algorithm for analyzing SUN Partition Table

Input: Disk-name with SUN Partition table
Output: Analyzing SUN Partition table

Algorithm for analyzing SUN Partition table
Begin
Char buffer[512]
Filedescriptor1 ← open(Disk-name, O_RDWR)
nheads ← (unsigned short *)(buf+437)
lseek(Filedescriptor1,437,SEEK_SET)
print *nheads

ntracks ← (unsigned short *)(buf+439)
lseek(Filedescriptor1,439,SEEK_SET)
print *ntracks

label1 ← (unsigned short *)(buf+508)
lseek(Filedescriptor1,508,SEEK_SET)
print *label1

disk_speed ← (unsigned short *)(buf+420)
lseek(Filedescriptor1,420,SEEK_SET)
print *disk_speed

phy_cylinders ← (unsigned short *)(buf+422)
lseek(Filedescriptor1,422,SEEK_SET)
print *phy_cylinders

version ← (unsigned short *)(buf+431)
lseek(Filedescriptor1,431,SEEK_SET)
print *version

phy_cylinders1 ← (unsigned short *)(buf+432)
lseek(Filedescriptor1,432,SEEK_SET)
print *phy_cylinders1

disk_size ← (unsigned long *)(buf+464)
lseek(Filedescriptor1,464,SEEK_SET)
print *disk_size

checksum ← (unsigned short *)(buf+510)
lseek(Filedescriptor1,510,SEEK_SET)
print *checksum

close(Filedescriptor1)
End

TABLE V: Data Structure for the SUN SPARC Disk label[2]

<table>
<thead>
<tr>
<th>Byte Range</th>
<th>Description</th>
<th>Essential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-127</td>
<td>ASCII Label</td>
<td>NO</td>
</tr>
<tr>
<td>128-261</td>
<td>Sparc VTOC</td>
<td>YES</td>
</tr>
<tr>
<td>262-263</td>
<td>Sectors to skip, writing</td>
<td>NO</td>
</tr>
<tr>
<td>264-265</td>
<td>Sectors to skip, reading</td>
<td>NO</td>
</tr>
<tr>
<td>266-419</td>
<td>Size of partition in sectors</td>
<td>NO</td>
</tr>
<tr>
<td>420-421</td>
<td>Disk speed</td>
<td>NO</td>
</tr>
<tr>
<td>422-423</td>
<td>Number of physical cylinders</td>
<td>NO</td>
</tr>
<tr>
<td>424-425</td>
<td>Alternates per cylinder</td>
<td>NO</td>
</tr>
<tr>
<td>426-429</td>
<td>Reserved</td>
<td>NO</td>
</tr>
<tr>
<td>430-431</td>
<td>Interleave</td>
<td>NO</td>
</tr>
<tr>
<td>432-433</td>
<td>Number of data cylinders</td>
<td>NO</td>
</tr>
<tr>
<td>434-435</td>
<td>Number of alternate cylinders</td>
<td>NO</td>
</tr>
<tr>
<td>436-437</td>
<td>Number of heads</td>
<td>YES</td>
</tr>
</tbody>
</table>
TABLE VI: Data Structure for the VTOC in SUN SPARC Disk label [2].

<table>
<thead>
<tr>
<th>Byte Range</th>
<th>Description</th>
<th>Essential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>Version (0x01)</td>
<td>NO</td>
</tr>
<tr>
<td>4-11</td>
<td>Volume Name</td>
<td>NO</td>
</tr>
<tr>
<td>12-13</td>
<td>Number of Partitions</td>
<td>YES</td>
</tr>
<tr>
<td>14-15</td>
<td>Partition #1 type</td>
<td>NO</td>
</tr>
<tr>
<td>16-17</td>
<td>Partition #1 flags</td>
<td>NO</td>
</tr>
<tr>
<td>18-19</td>
<td>Partition #2 type</td>
<td>NO</td>
</tr>
<tr>
<td>20-21</td>
<td>Partition #2 flags</td>
<td>NO</td>
</tr>
<tr>
<td>22-23</td>
<td>Partition #3 type</td>
<td>NO</td>
</tr>
<tr>
<td>24-25</td>
<td>Partition #3 flags</td>
<td>NO</td>
</tr>
<tr>
<td>26-27</td>
<td>Partition #4 type</td>
<td>NO</td>
</tr>
<tr>
<td>28-29</td>
<td>Partition #4 flags</td>
<td>NO</td>
</tr>
<tr>
<td>30-31</td>
<td>Partition #5 type</td>
<td>NO</td>
</tr>
<tr>
<td>32-33</td>
<td>Partition #5 flags</td>
<td>NO</td>
</tr>
<tr>
<td>34-35</td>
<td>Partition #6 type</td>
<td>NO</td>
</tr>
<tr>
<td>36-37</td>
<td>Partition #6 flags</td>
<td>NO</td>
</tr>
<tr>
<td>38-39</td>
<td>Partition #7 type</td>
<td>NO</td>
</tr>
<tr>
<td>40-41</td>
<td>Partition #7 flags</td>
<td>NO</td>
</tr>
<tr>
<td>42-43</td>
<td>Partition #8 type</td>
<td>NO</td>
</tr>
<tr>
<td>44-45</td>
<td>Partition #8 flags</td>
<td>NO</td>
</tr>
<tr>
<td>46-47</td>
<td>Boot info</td>
<td>NO</td>
</tr>
<tr>
<td>48-49</td>
<td>Signature Value (0xDABE)</td>
<td>NO</td>
</tr>
<tr>
<td>50-51</td>
<td>Checksum</td>
<td>NO</td>
</tr>
<tr>
<td>52-53</td>
<td>Boot info</td>
<td>NO</td>
</tr>
<tr>
<td>54-55</td>
<td>Signature Value (0xDABE)</td>
<td>NO</td>
</tr>
<tr>
<td>56-57</td>
<td>Checksum</td>
<td>NO</td>
</tr>
<tr>
<td>58-59</td>
<td>Reserved</td>
<td>NO</td>
</tr>
</tbody>
</table>

Fig 11: An SUN disk with three dos partitions, the final one contains a disk label and three SUN partitions.
TABLE VII: Data Structure for the SUN SPARC Disk label Disk Map [2]

<table>
<thead>
<tr>
<th>Byte Range</th>
<th>Description</th>
<th>Essential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>Starting Cylinder</td>
<td>YES</td>
</tr>
<tr>
<td>4-7</td>
<td>Size</td>
<td>YES</td>
</tr>
</tbody>
</table>

VIII. AMIGA PARTITION TABLE (AMIGA) ANALYZER ALGORITHM

Algorithm for analyzing AMIGA Partition Table

Input: Disk-name with AMIGA Partition table
Output: Analyzing AMIGA Partition table

Algorithm for analyzing AMIGA Partition table
Begin
Char buffer[4096]
Filedescriptor1 ← open(Disk-name, O_RDWR)
p ← (RigidDiskBlock *)(buffer +1024)
if(p → rdb_ID==0x4b534452)
    print 'Amiga Label is 0x4b534452'
else
    print ‘Amiga Label is not Found’
print p → rdb_SummedLongs, p → rdb_ChkSum,
print p → rdb_HostID, p → rdb_BlockBytes
print p → rdb_FLAGS
close(Filedescriptor1)
End

Fig 12: Screen shot for creating AMIGA Partition Table
IX. BSD PARTITION TABLE (BSD) ANALYZER ALGORITHM

Algorithm for analyzing BSD Partition Table

Input: Disk-name with BSD Partition table
Output: Analyzing BSD Partition table

Algorithm for analyzing BSD Partition table
Begin
Char buffer[512]
Filedescriptor1 ← open(Disk-name, O_RDWR)
p ← (BSDRawLabel *)(buffer +64)
if(p->d_magic== BSD_DISKMAGIC)
    print ‘BSD label is Found that is : 0x82564557’
else
    print ‘BSD label is not Found’
if(p->d_type==0x1)
    print ‘Drive type is :BSD_DTYPE_SMD that is SMD<XSMMD’
elseif(p->d_type==0x2)
    print ‘Drive type is :BSD_DTYPE_MSCP’
elseif(p->d_type==0x3)
    print ‘Drive type is :BSD_DTYPE_DEC that is other DEC (rk,rl)
elseif(p->d_type==0x4)
    print ‘Drive type is :BSD_DTYPE_ESDI that is ESDI interface’
elseif(p->d_type==0x5)
    print ‘Drive type is :BSD_DTYPE_DEC that is DEC DTYPE_ESDI
elseif(p->d_type==0x6)
    print ‘Drive type is :BSD_DTYPE_ST506’
elseif(p->d_type==0x7)
    print ‘Drive type is :BSD_DTYPE_HPIB that is CS/80 on HP-IB’
elseif(p->d_type==0x8)
    print ‘Drive type is :BSD_DTYPE_HPFL that is HP Fiber-link’
else
    print ‘bsd dtype is not found.’
print p->d_subtype, p->d_secsize, p->d_nsectors, p->d_ntracks
print p->d_ncylinders, p->d_seccyl, p->d_seccylunit
print p->d_sparecylinders, p->d_acylinders,
p->d_rpm, p->d_interleave
print p->d_trackskew, p->d_cylskew,
p->d_headswitch, p->d_trkseek
print p->d_flags, p->d_sparespertrack
for i=0 to 5
    print p->d_driverdata[j]
print p->d_magic2, p->d_checksum, p->d_npartitions
print p->d_bbsize, p->d_sbsize
close(Filedescriptor1)
End
We clearly stated algorithms of how to detect and analyze different major partition tables with disk labels. We have performed analysis on all major partition tables and succeeded in detecting and analyzing them. In future enhancement, the analysis will be extended on remaining partition tables and make this utility strong enough of detecting any specified partition table. The conclusion of the paper is to avoid data corruption in cloud environment by using partition analyzer utility.

References


Fig 13: Screen shot for creating BSD Partition Table
**Author(s) Profile**

**Perla Ravi Theja**, received the Master of Technology in Computer Science and Engineering from School of Computer Science and Engineering (SCSE), VIT University, Vellore in 2012. He is currently working as a Programmer Analyst for the Cognizant Technology Solutions and pursuing PhD degree in Computer Science and Engineering. During his Masters, he spent ten months as an intern at VMware, Bangalore. His research interests include cloud, storage area networks, partition tables, iscsi, fiber channel, cloud security and virtualization. He has published several papers in international journals and conferences related to his research topics.

Mr. Perla Ravi Theja is a member of the IEEE Computer Society. He has awarded best student award in his school during his masters. He has published “Partition Table Generator for Cloud OS”, in International Journal of Computer Applications (IJCA), 41(16):1-15, March 2012. http://research.ijcaonline.org/volume41/number16/pxc3877750.pdf.

**Varun Manchikalapudi**, received Master of Technology in Information Technology, Networking Specialization from School of Information Technology and Engineering (SITE), VIT University, Vellore in 2010. He is working as Assistant Professor in Department of Information Technology, V R Siddhartha Engineering College, Vijayawada since 2010. He has an industry experience, Worked as Software Intern in Honeywell Technology Solutions Lab, Bangalore. He has published two research papers on wireless networks in IEEEExplore digital library. He has published a paper on image processing in an International Journal (IJCST) in 2011.

**Dr. SK. Khadar Babu**, working as Senior Assistant Professor in VIT University, Vellore. He is working under statistics and operations research division in school of Advanced Sciences. His area of interests include computer networks, cloud computing, Digital image processing, ARIMA models, Statistical quality control, Fuzzy programming techniques.