www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issue 11 Nov. 2016, Page No. 19037-19044

A Data Owner Controlled Model for Co tenant Covert Channel Protection

S.Rama Krishna, B.Padmaja Rani

Abstract— The emerging technology cloud Computing has became as a popular choice for small and medium enterprises for their infrastructural needs. It has many services providing support like computing, storing etc. Virtualization is the key technology to provide features like scalability and elasticity using multi tenancy. In multi tenant cloud architecture malicious co tenants use covert channels like RAM, bus and other shared devices to hijack the sensitive data. Literature is available for identifying and preventing these kinds of attacks in matured cloud providers. Immature or lazy cloud service providers will suffer from these kinds of problems profoundly. Our novel model tries to suggest measures in protection of covert channel protection. It uses a simple technique of split and sharing data to store multiple regions in multiple places. We have taken factor called as missing data which is inaccessible data in the case of compromised co tenant covert channel attack. Experiments were done in the Amazon S3 cloud with different accounts in different regions and multiple buckets for storing split data and results were analyzed. We come across with best number of splits to maximize missing data factor with less impact over the time cost for upload and download operations from our result analysis satisfactorily.

Index Terms— cloud computing, Covert Channel Protection, Split Share, Virtualization security

1. Introduction

Simplicity, availability and ease of use are the features that make corporate companies to move towards cloud computing [5] [6]. Cloud computing is widely using technologies like Application clustering, Network technologies, virtualization and distributed file system for offering services to its wide variety of users. Distributed storage which is provided by the Cloud storage service provider makes the data available from anywhere and anytime. Cloud users use heterogeneous devices to access data from the cloud storage service [7][8]. Cloud service provider utilizes multi-tenancy to enable resource utilization maximization such as storage or computation etc.., Multiplexing is the technique used for sharing the virtualised physical resources to its cloud users. Multi tenancy will optimize usage of system physical resources it also support multiple virtual machines share one common physical space from different customers. In the same storage space these multiple user's data get stored. Scheduling and data storage responsibility is taken care by Cloud provider. This feature multi tenancy though it is helpful for maximizing usage of resource it may also cause several new security problems. Cloud security research is gaining more focus from researchers and numerous publications show their interest. This research is resulting in identification and mitigation of information leakage which is caused by feature co-tenancy. In multi tenant architecture of cloud computing virtual machine that share common physical space is said to be Co-tenant.

A malicious co tenant can trick to gain information using covert channels. RAM memory, hard disk, CPU cache, networking or i/o bus [9][10][11] can be acted as covert channels for these co-tenant attackers. This causes sensitive data leakage to the malicious VM that stays in same physical resources. Even some malicious VMs try to access the data that stored in same physical storage device which it is not privileged to access. A strict access policy is required because even a policy like Chinese wall security is proven to be vulnerable [12]. Channel hijacking can be prevented by storing the conflict files physically isolated in storage space. Cloud provider can easily implement this by including it in service level agreement. For avoiding channel hijacking problem cloud service provider has to assign isolated storage space with other specific malicious tenants. Cheap and lazy cloud service provider may not enforce this strategy due to negligence or cost cutting process [13].

This paper presents a novel framework structure that helps in the protection of data over cloud from un trusted malicious co tenant. The proposed framework is named as Co-tenancy Covert Channel Protection model. The organization of remaining of paper is organized as follows. Section 2 presented with literature related to security issues of co tenant covert channel attacks. Section 3 presents major problem of focus and design objectives of proposed model. Section 4 introduces Split share model for Co-tenant Covert channel protection. Section 5 brings implementation details of the model. Section 6 Analyses the results derived by model. Section 7 summarises and section 8 gives future directions.

2. Literature on Covert-channel attacks and defences

Covert channels are utilized by the co resident VM to obtain unprivileged information from another virtual machine working in same workspace. These covert channels can give chances to get access to unauthorized data. Memory, network, CPU cache, and power consumption can be used as covert channels for extracting others data. Literature [1] introduced a method for tracking software events by observing behavioural changes in hardware. The CPU cache

DOI: 10.18535/ijecs/v5i11.57

response time is used to verify whether the target VM is co occupant or not [4]. Assailant uses load predictor and cubic spline are used to process the cache utilization data linear regression to analyze the behaviour of the cache. An attacker VM will possess the significant portion of cpu cache to target its co-tenant. Subsequently sends a data request to the co-tenant. Attacker machine ascertains cache access time using a program to measure loads produced for this transaction. In attacker machine it executes load measuring program to calculate cache access time. Literature [4] presumes that higher access time means more co-tenant activities. Same experiment also extended and tested with three VM. Attackers VM not only analyses cache access times but also capture the target VM data through the covert channels. Literature [2] specified the role of malware addition by attacker VM over target machine for hijacking the information. Attacker infuses malware into software of target machine to gain indirect access. Attack acquires information from target VM and also removes all the traces used for the attack without leaving any evidence. For this attack memory bus is used as a covert channel. Attacker VM sends 1 to memory bus because atomic CPU instruction will be issued. This atomic instruction will increase latency in memory usage. When the latency is reduced than transfers 0 bit and releases memory bus. Literature [2] also referred to other covert channels like exploiting contention of cache. Target VM bandwidth is calculated using overlapped time in execution by the attacker VM. Literature [2] referred a technique to mitigate this problem by making changes to the scheduler of hypervisor. For the successful defence to bandwidth analysis using cache contention, overlapping time of execution is minimized for two virtual machines. Proper care should be taken to minimize overlapping time without reducing system performance. One more bottle neck to protect bandwidth analysis attack is that it is very difficult for scheduler to identify malicious VM. Limiting the frequency of switching VMs will minimize overlapping execution time but reduces system performance [2]. Therefore, as another measure to counteract noise pumping technique can be used in protection of bandwidth analysis attack. This generated noise defends attack against memory bus. Xenpump is a proposed model mentioned in literature [1][3] to protect covert channel attacks. Random latency is generated by Xenpump to limit the bandwidth of timing channel. Attacker VM is confused by generated latency, but this will reduce effectiveness of timing channel. Imposing unpredictability through generated latency will defend bandwidth analysis attack but reduces system performance. Literature [1] discussed another type of attack called as Prime trigger probing that uses cache as the covert channel attack. This probing is performed by the attacker machine by occupying more cache lines. Attacker VM try to access more records to occupy many cache lines. Subsequently obtains encoded message of target VM by accessing parts of cache. Attacker VM will start accessing cache parts after target VM completes its job. As attacker has greater access time compared to the baseline cache failure it is caused by each line used to access the cache.

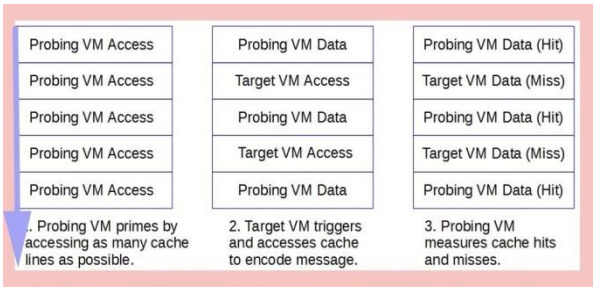

Fig. 1 A diagram is shown to describe about Prime trigger probing.

To safeguard Probing attack flushing the data in between the switching of VMS can be done so that information in cache will not be accessible to the attacker. From the exploratory results it was observed that 15% more overhead is created by using flushing technique mentioned in literature [1].

3. Major Problem of focus

3.1. Co-tenant Covert Channel attack

Shared resources such as RAM, CPU cache or hard disk storage can be used as covert channels to leak valuable information while services of two VMs which try to access conflict files of interest through those shared resources. Literature [5] [6] [7] presents several such major issues. This study was not sufficiently done over cloud storage.

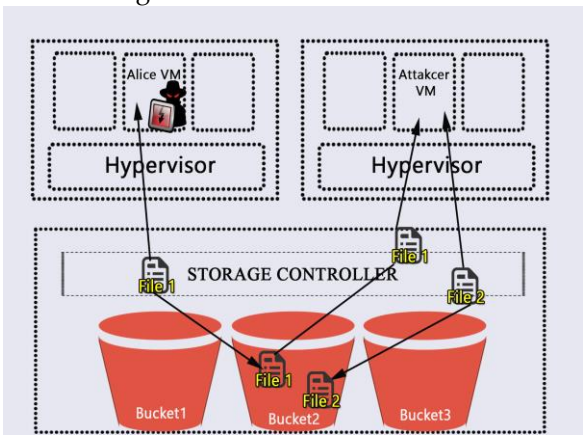

Fig. 2. Information leakage due Co tenant Covert channel attack in shared storage space.

Leakage in storage of hard disk can be explained using figure 4.1 as an example. Alice VM and Attacker VM are two virtual machines hosted in two different physical machines. File 1 and file 2 are two files with interest of conflict those are stored in same disk storage space. Alice is having access permission for file 1 while the attacker has access privileges for file2.Hypervisor or

storage controller will be protected by enforcing strict access policy like Chinese wall security [8]. Security policy restricts Alice VM access to File 2 or Attacker VM access to File 1.

When Alice VM compromised by malware sent by attacker VM, then Alice VM may be guided to access File 1 with certain pattern so that contents of File 1 can be leaked to Attacker VM through covert channel like shared bus or cache. Attacker VM uses this pattern to decode file1 using access time measurement of co located file1. This will cause leakage even when we have powerful access control policy like Chinese wall policy.

3.2. **Design objectives of model**

1. Model has to reduce the attack surface from malicious Co-tenant to protect even in compromised covert channel attacks.
2. Identifying optimized split criteria (no of splits) that maximize the missing data and minimizes split time and upload times.
3. Model has to support customizable storage policy that supports multiple objects, multiple buckets, multiple regions, and multiple account support.
4. Model has to provide support for Integrity verification

4. Split Share Model

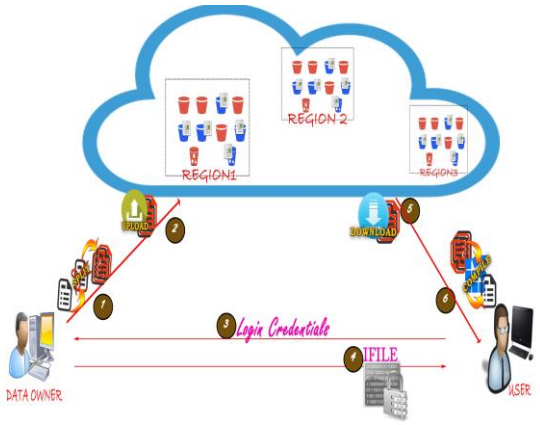

Fig 3 Split – Share Model for Co-tenant Covert Channel Protection

Split –Share Model contains five modules. Named to be

1. File split module
2. Data upload module
3. Data share module
4. Data download module
5. Compile module

In this method of Co-tenant Covert Channel protection we focus more on splitting data and storing it in the distributed manner unlike conventional storage, for making data unavailable even in the case of co tenant using covert channel to access un privileged sensitive information as afore mentioned in section 3.1. Objective of Split –Share method is to reduce data availability to the malicious co-tenants by splitting data in to different buckets, different regions and even for different clouds. But in this paper our topic of discussion limited only for the single cloud.

By splitting the data to different buckets or in different regions even bucket is compromised or even the region is compromised only part of data will be available for malicious co-tenant. This method may give best covert channel protection in the case of immature or lazy cloud storage service providers. As these service providers don't try to isolate data which is conflict of interest (i.e., data which is tried to hijack by malicious co-tenants through covert channels) files because of negligence or cost cutting process. In this method we focused on a factor called missing data for comparing our method with existing methods. Remaining of this section discusses in detail about different modules involved in the basic construction of Split-Share method and how it protects from malicious Co-tenant covert channel protection.

4.1 File split (file, n):

Inputs

File – Data owner File information
n- Number of Splits

Output

IFile- information File
Splitfilelist[n]

Begin

Fsize← getSize(file)
Blocksize← Fsize/n
If(Fsize % n!=0)
N=n+1
End if
For i =1 to n do
Create newfile
Readbuffer ← Read file(blocksize)
newFile. write(Readbuffer)
SplitFileList[i]← newFile
IFile.write(SplitFileList[i])
End for

End

Description:

File and no of splits are two inputs given to File split module. Size of file is divided by n to find out each split file size. After dividing if any data remaining that will be added to n+1 split. If the size n increases security increase but proportionally computation overhead also increases. So while splitting the file n should be picked heuristically. Same module may be re written for supporting variable file sizes. In this model we implemented and tested only equal size partition. In IFile split file information to be incorporated for sharing that to the user who data owner intend to share. For error detection in the split file we can maintain CRC/ Message Digest each split information along with CRC updated in IFile.

4.2.Uploadfile(UploadPolicy,Mode,CloudAccessInfo, SplitFilesList):

Inputs:
SplitFilesList-FileArrayconsiting list of files
UploadPoilicy-policy defined for distribution of split files
Mode- distribution mode
CloudAccessInfo –cloud Account information
Output:
IFile- information File
Begin
For i← n
Account ← getCloud(CloudAccessInfo,UploadPolicy)
Region ← setRegion(uploadPolicy)
Bucket ← createBucket(mode)
Object ← readFile(SplitFileList[i])
uploadcloud(Cloud,Region,Bucket,Object)
update(IFile)
End for
End

Description:

Upload file module takes four inputs IFile → IFile consist of information about files after splitting such as no of splits, file size, timestamp, message digest value.
UploadPolicy→ it is having values ranging from 1 to 4

1. Normal file upload
2. Split single Bucket in single region
3. Split Multiple Bucket in single region
4. Split Multiple buckets in multiple regions

Mode→ it has 3 values

1. Random distribution mode
2. Circular distribution mode
3. Sequential distribution mode

CloudAccessInfo →this contains cloud account information such as account name, access parameters.

Uploading process is done based on the upload policy and mode by picking proper account information from cloud access info data structure. Once uploading is done details of uploading account information and time stamp will be updated to the IFile .

Each split file is taken from split file array properly uploaded by picking the proper cloud, region, and bucket with different modes of storage and naming conventions. If the files were not properly distributed we may not achieve our desired parameter (missing data) to the extent of expectation.

Here our experimental implementation only limited to single cloud but in our method we made a generalized framing of algorithm so that even in future extension same model can be used for multiple cloud infrastructures also.

4.3. Data sharing (Login credentials**)**

Inputs:
LoginCredentials -user login credentials
Output:
IFile – information File
Begin
If(login successful)
Transfer IFile
Else
Return error message
End if
End

Description:

When the user wants to get information from the data owner he has to subscribe at data owner for credentials. Once he receive credentials at the time of authentication user has to produce these credentials to the data owner in secure channel. Data owner will verify these credentials and if the credentials satisfy the access policy of the information then will share the file information to the user in secret form either mail or encrypted format.
Data owner may also reduce burden of being online for authentication or credentials verification by deploying a trust server at cloud. Proper secrecy must be ensured at trust server so that if it is compromised entire structure will be compromised. Model should support for secure IFile storage and transformation to prevent data leakage.

4.4. Download(IFile**)**

Inputs:
IFile – information File
Output:
DownloadedFilesList[n]
Begin
For i = 1 to n
Account ← getCloud(IFile)
Region ← getRegion(IFile)
Bucket ← getBucket(IFile)
Object ← readFile(IFile)
DownloadfileList[i]←downloadcloud(Cloud,Region,Bucket,Object)
Chksum←IFile.getChecksum(i)
If(chksum!= Checksum(DownloadedFilesList[i]))
Return error
End if
End for
End

Description:

User of the data who obtained IFile can easily download the list of files from the cloud as mentioned in IFile by providing proper security credentials at cloud. Process repeated for n splits and digest verification is done here.
In this module we can also keep the additional feature of error checking by calculating CRC value for each downloaded file. This CRC will be checked by comparing the generated CRC with one that is there in IFile. If verification produces any errors same fragment can be re downloaded and file re-construction can be done again.

Anyhow error detection is not major concern or scope of the discussion so we limit the discussion here.

4.5. Compile (DownloadedFilesList**)**

Inputs:

IFile

DownloadedFilesList[n]

Output:

Final

Begin

For i=1 to n do

Readbuffer ← Readfile(DownloadedFilesList[i])

Final.append(Readbuffer)

End for

End

Description:

Compile module is to re formulate original file at to make it ready for user. All the downloaded files are appended to the original file to re construct. This module pools all piles of information according to IFile sequence number and compiles a complete file.

5. Implementation

The Split share model was implemented using java. Amazon S3 (Simple Storage Service) accounts were created in Amazon Web Services public cloud. These accounts were configured to create buckets and store objects in different regions like US-EAST, US-WEST2 and so on. AWS java API is used for accessing Amazon Web Services S3 (Simple Storage Service). Eclipse Mars IDE is used to execute developed program. Data owner and User Machines we use Intel(R) Core(TM) i5-4210U CPU @ 1.70 GHz to 2.40 GHz With 8 GB RAM. Data Owner Machine get Connected to the cloud with basic internet speeds 256 KBPS to 2 MBPS. All the experiments were done at various times of day in a week and results were averaged and normalized using Min –Max Normalization technique.

Split share model discussed above tested with 4 different upload policies. Such as

1. Normal file upload
2. Split single Bucket in single region
3. Split Multiple Bucket in single region
4. Split Multiple buckets in multiple regions

Setup 2:

To identify co-residency in hard disk contention data is stored in storage service offered by Amazon-S3 Cloud provider. Data in Amazon S3 is stored as objects. Buckets are the containers of objects. Each bucket has to be given with unique name so that it is accessible to everyone. Each object is associated with a key value. This key value in the bucket is unique. For testing we utilize two upload policies mentioned underneath.

Multiple bucket upload policy: Reading two files from one bucket with other files with n buckets same region.

Single bucket upload policy: Reading two files from one bucket.

For the verification of disk contention we perform download operation with two t2.large EC2 instances in US-East region. The two instances has same configuration. These instances are utilized to download files at the same time from S3. These files stored in S3 were uploaded using two upload policies specified previously. Test was rehashed for different document sizes. Correlation coefficient of downloading times was tabulated and figured as appeared in Fig. 11. We conduct the experiment at different time of a day and repeated for two weeks. We can observe that the download time correlation coefficient of multiple bucket upload policy is having higher order than the single bucket upload policy.

6. Result analysis

6.1 Best number of splits

For analyzing best no of splits we conducted experiments by uploading different files starting with 100 KB to 1.5 GB sizes. We record values of missing data and split size with no of splits ranging from 1 to 100.

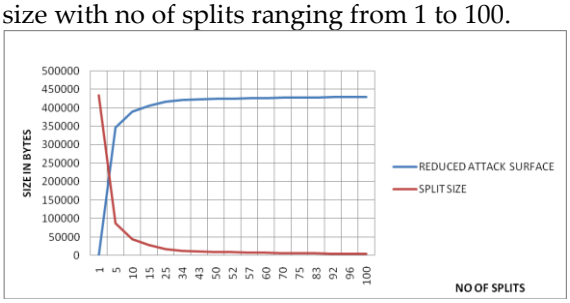

Figure 4 Split Size Vs Mising Data

Figure 4 on the X Axis we denote no of Splits which variate from 1 to 100 , Y Axis we had taken File size in bytes. Graph was drawn by taking a file size of 43356 bytes. There were two curves to represent split size and missing data in the bucket as reducing surface area in the graph. This graph is drawn to find out the relation between reduced surface area and split size with respect to no of splits. From the graph it was observed as the number of splits increases split size is decreasing and proportionally there is further reduction in data available in same bucket.

But from the graph it was observed that variation to number of splits to missing data is considerably high in the range of no of split in between 8-15. Even number of splits increase further beyond that missing data is not increasing significantly.

So we can conclude that best number of splits for any file will be in the range of 8 to 15.

DOI: 10.18535/ijecs/v5i11.57

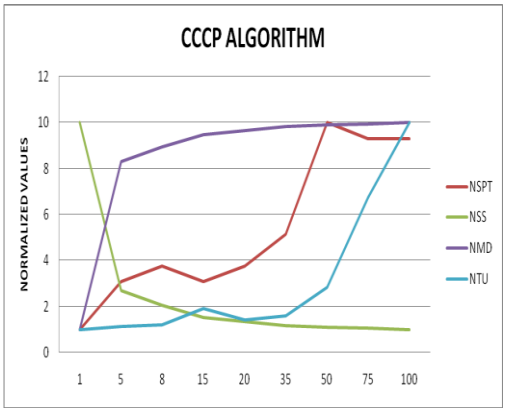

Figure 5 Normalized values of Split time, Split Size, Missing Data , Total Upload Time change with no of split

Figure 5 is the Graph which also includes Normalized Split Time and Normalized Total Upload Times along with Normalized Split Size and Missing Data. Here we use Min –Max Normalization process to bring the values in the range of 1 to 10 as Split size and missing data sizes are dominating the split time and upload times. This figure presents variation in split time, split size total upload time and missing data with respect to no of splits which variate starting from 1 to 100 depicted on X-Axis. Y –Axis Normalized values were represented.

These curves also define best no of splits as at best number of splits split size , split time and total upload time should be minimum where as missing data should be high. In such case observations also proves that in the range of 8-15 no of splits that it is the best range of no of splits .

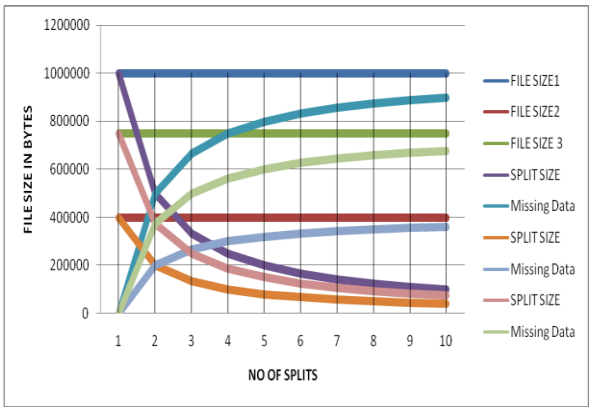

Figure 6 Split size and Missing data variation Vs no of splits

Figure 6 the depicts graph of curves for 3 different file sizes . It was observed split size and missing data variation for 3 different file sizes 47 MB, 57 MB, 100 MB. Here we took no of splits taken are 1 to 10 over X axis and on the Y- axis file sizes.

6.2 Performance variation for three upload policies

Next experiments were done to find the overhead cost incurred among various models. For these experiments we choose different file sizes and applied 3 models and compared the generated results with normal file uploading.

It was observed that as the file sizes increase costs increased proportionally. From the results discussed in previous part of this section we may conclude that among three upload models split multiple buckets and split multiple buckets in multiple regions will have high value of missing data.

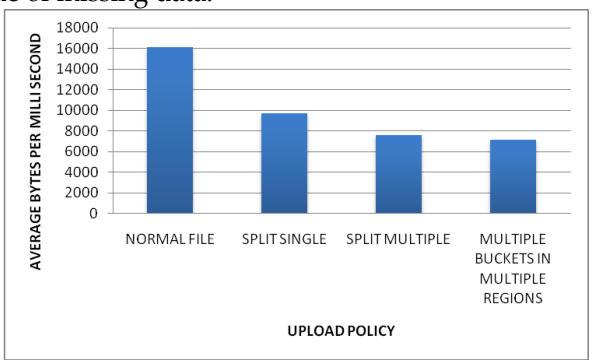

Figure 7 . Average bytes per Mille second transferred considering the total cost involved for four models.

Split Single bucket and Normal file upload policies does not able to increase the missing data size. In this experiment we try to observe the impact of additional costs over file transfer rate incurred in four upload policies. This experiment was conducted by uploading different files with various sizes ranging from 34 KB to 700 MB. It was tested by sending files from various regions in the cloud. All the results were averaged and they were rounded up for drawing the graphs.

From the figure 7 on the X axis we have four different upload policies. Y axis we represent average no of bytes those were transferred per mille second. Split single bucket upload policy when compared to split multiple bucket upload policy more transfer rate but the difference is considerably less. The same case was observed with the split multiple buckets upload policy in multiple regions.

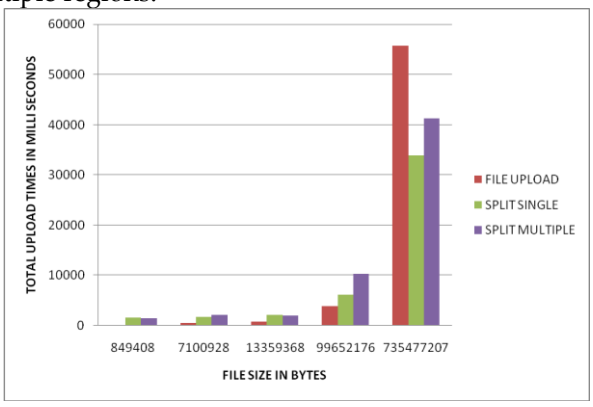

Figure 8 Total upload times for three models with various file sizes

It was observed that normal file is having high average no of bytes transferred per mille second compared to remaining three models. The remaining three models involve splitting time

DOI: 10.18535/ijecs/v5i11.57

additionally there is little overhead added to the model.

6.3 Performance evaluation of split multiple bucket upload policy.

This experiment gives how different times were varying with respect to different file sizes in split multiple bucket upload policy in the single region. We this we created 10 buckets in US-WEST2 region. Uploaded split files to these 10 buckets sequentially. Experiment was conducted in such a way that network latency was reduced, by setting up ec2 instances and s3 buckets in the same region.

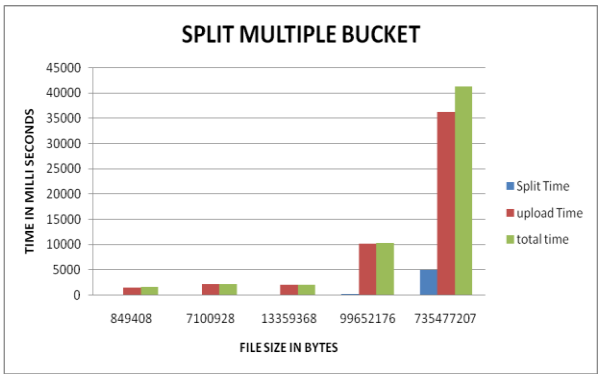

Figure 9 Split ,upload , total time comparison of various file sizes in the split multiple bucket model.

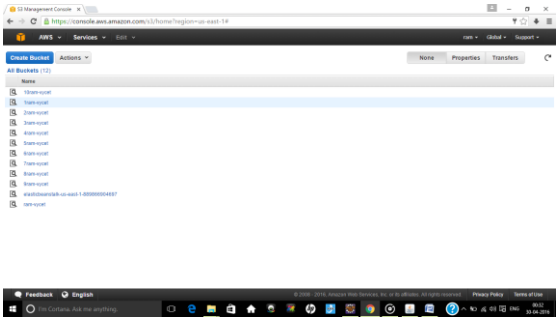

Figure 10 Screen shot of split file storage in multiple buckets in AWS S3 account

Figure 9 shows the results drawn by uploading five different files of size varying from 30 MB TO 700 mb. From the results it was observed that split time is less factor when it compared to upload time and it moves up with the file size that we transfer. Figure 10 gives a snapshot of files stored in multiple buckets in AWS cloud.

6.4 Hard disk contention checking between split single bucket upload policy and split multiple buckets upload policy.

This experiment was done using second setup as mentioned in implementation section 5 to check the hard disk contention when two users download two files at a time. From the results drawn we draw figure 11 and it was clear conclusion as we are isolating two files hard disk contention reduced significantly and very less correlation has observed while downloading files in split multiple bucket upload policy.

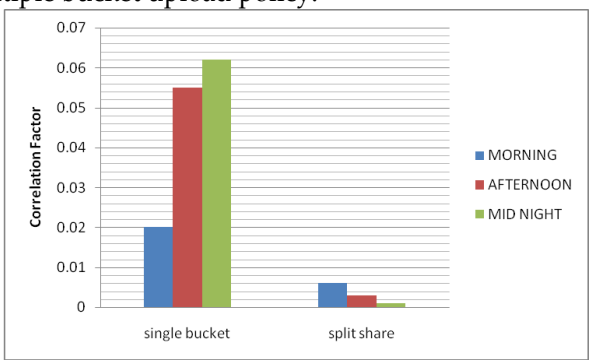

Figure 11 Correlation factor for single bucket and split share 2 files reading times.

7. Summary

This paper proposes a novel model split share for provisioning security to the cotenant covert channel attack. The proposed model implemented and the results were analyzed. We can summarize that split share model with multiple bucket upload policy at no of splits range from 8 to 15 gives best results. These results are satisfactory as we observe less overhead, more missing data factor and less hard disk contention correlation factor as per the design objectives. Error detection was also included in the model.

8. Future Scope

8.1 Encryption Function

For making malicious co-tenants job tough in the covert channel attack we need to make even the available small piece of information difficult to access. This can be possible by incorporating encryption function to the model. So that the data we place in the cloud would be much harder to crack for the malicious cotenants. Working in this direction may give valuable conclusions. Security should not become unbearable overload computing. So in this model we propose a basic model.

8.2 Multi Cloud Support

Since the implementation differentiation amongst multiple clouds our proposed model was not tested under the multi cloud environment. Research in that direction may make even tough for malicious co-tenants to gain access as well as difficult to leak data. Future research may address this direction.

8.3 Sharing and Storage of IFile

IFile consist of all the Meta data related to cloud access as well as storage information and check sum. So Storage and sharing of this IFile information in a flexible and secure manner is expected. Research in this direction may also draw some important conclusions.

9. References

[1] M. Godfrey and M. Zulkernine, "A Server-Side Solution to Cache-Based Side-Channel Attacks in the Cloud," Proc. Of 6th IEEE International Conference on Cloud Computing, 2013, pp. 163–170.
[2] F. Liu, L. Ren, and H. Bai, "Mitigating Cross-VM Side Channel Attack on Multiple Tenants Cloud Platform," Journal of Computers, 9(4), 2014, pp. 1005–1013.
[3] J. Wu, L. Ding, Y. Lin, N. Min-Allah, and Y. Wang, "xenpump: A New Method to Mitigate Timing Channel in Cloud Computing," Proc. Of 5th IEEE International Conference On Cloud Computing, 2012, pp. 678–685.
[4] S. Yu, X. Gui, J. Lin, X. Zhang, and J. Wang, "Detecting vms Co-residency in the Cloud: Using Cache-based Side Channel Attacks," Elektronika Ir Elektrotechnika, 19(5), 2013, pp. 73– 78.
[5] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *ACM Conference on Computer and Communications Security*, 2009, pp. 199–212.
[6] Y. Xu, M. Bailey, F. Jahanian, K. R. Joshi, M. A. Hiltunen, and R. D. Schlichting, "An exploration of l2 cache covert channels in virtualized environments," in *CCSW*, 2011, pp. 29–40.
[7] J. C. Wray, "An analysis of covert timing channels," in *IEEE Symposium*
on Security and Privacy, 1991, pp. 2–7.
[8] D. F. C. Brewer and M. J. Nash, "The chinese wall security policy," in *IEEE Symposium on Security and Privacy*, 1989, pp. 206–214.
[9] Dan@AWS, "Best Practices for Using Amazon S3," 2009. [Online]. Available: http://aws.amazon.com/articles/1904
[10] Amazon Web Services. [Online]. Available: aws.amazon.com
[11] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Encryption policies for regulating access to outsourced
data," *ACM Trans. Database Syst.*, vol. 35, no. 2, 2010.
[12] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing" in *INFOCOM*, 2010, pp. 534–542.
[13] T. Tsai, Y. Chen, H. Huang, P. Huang, and K. Chou, "A practical Chinese wall security model in cloud computing," in *APNOMS*, 2011, pp. 1–4.