A Disk-Based Covert Channel Implementation

Srishti Bhargava
Information Security Institute,
Johns Hopkins University,
Maryland, USA
sbharga2@jhu.edu

Moriyike Mejabi
Information Security Institute,
Johns Hopkins University,
Maryland, USA
mmejabi1@jhu.edu

ABSTRACT
Cloud providers run multiple machines owned by different cloud users on the same physical host. This is done to make use of optimization of resources, and reduce the cost of running physical infrastructure for resources provisioned on the cloud. However, it also makes virtual machines (VMs) residing on the system co-resident i.e. provides a unintended channel for communication between virtual machines. This paper explores one such covert channel, which exists due to the possibility of creating hard disk contention on the shared host. Conniving VMs exchange messages by measuring the time taken to read from a specific disk memory location, while the other is writing heavy files to several memory locations on the shared hard disk. This paper documents the design and implementation of this covert channel, analyzing the results obtained and discussing modifications made to improve the covert channel. Arguments are made for the effectiveness of this implementation, and why this experimentation is as good as one carried out on Cloud Lab. We then suggest further work and improvements for this project.

CCS Concepts
• Security and privacy → Virtualization and security

Keywords
Covert channels, Cloud computing, Virtual machine security

1. INTRODUCTION
Many individuals and organizations look to the cloud as a key player in their business continuity solutions and scalability issues. However, several papers have been written to point out some security issues that people should be aware of as regards the "cloud". This paper explores only one of these vulnerabilities- the possibility of covert communication channels between co-resident virtual machines (VMs).

There are a number of ways that co-resident VMs can communicate, including the measurement of cache usage, load based covert channels, keystroke timing and disk contention. In this paper, we only explore the covert channel that exists via disk contention between co-resident VMs. The design and implementation of this attack is discussed, and the results are analyzed. Discussions are made about implementation on Cloud Lab, and possible future work on this project are suggested.

1.1 RELATED WORKS
A lot of work related to covert channels has been done. Some of these studies are mentioned during the course of this paper. Dr. Ristenpart and a group of researchers explored various information leakage mediums on Amazon EC2 [1]. The Black Hat paper by Sophia D'Antoine, a Computer Security student in Rensselaer Polytechnic Institute goes deeper into covert channels made possible via shared physical memory [2]. Another group of researchers from the IBM Research Lab and Barr Iian University explore side-channels which are present due to data de-duplication [3].

2. ATTACK VECTOR
2.1 Co-Residence of Virtual Machines
Cloud providers situate multiple VMs on the same physical machine. This is done to save cost and resources that are utilized for providing cloud infrastructure for their customers. The problem with this is that there is the danger that malicious users exploit resources owned by other individuals. Worse still, if carefully targeted, malicious users can try to detect which VMs share the same physical machines with theirs [1].

2.2 Disk Contention
Hard disk contention occurs in a system when several applications or processes compete for I/O resources, causing a reduction in performance. Since virtual machines which are co-resident share the same physical hard disk, if a VM tries to gain access to a memory location on the disk while another VM is writing a very heavy file (using up most of the host machine's resources), a contention occurs. This is a means of detecting the presence of other virtual machines [2]. This channel can be used to send and receive messages between VMs.

3. THE COVERT CHANNEL
The covert channel that is implemented and tested in this paper is based on disk contention (as explained previously). A VM (the sender) accesses several memory locations by writing very heavy files to disk. This uses up more CPU resources than allocated to the VM, such that when another VM (receiver) reads from a certain location from the disk, it takes more time than normal (this is measured by carrying out tests).

A 1-bit is sent by writing large files, and a 0-bit is sent by staying idle during the specified time frame. Figure 1 is a diagram that represents the covert channel construction.

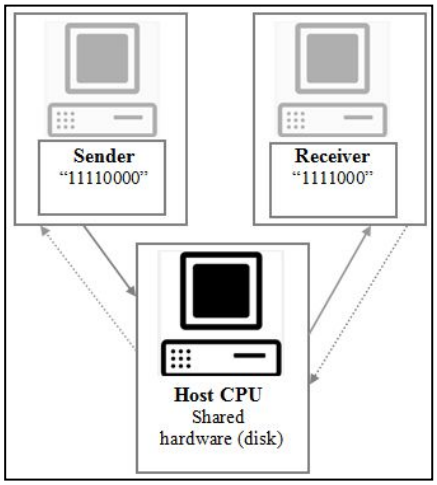

Figure 1. Two conniving VMs can communicate in synchronized time frames via hard disk competition.

4. EXPERIMENTS

The experimentation for this covert channel analysis was carried out locally on a host machine and then on Cloud Lab. The experiment comprised of using 2 virtual machines on the same host OS. This covert channel exploits contention caused in the hard-disk due to heavy synchronized writes and reads from multiple VMs- we call one VM the "sender", and the other "receiver". The experiment design, implementation, results, and discussions are included in this section.

4.1 Experiment Design and Implementation

Below are the specifications of the host OS, and 2 virtual machines used locally:

Host CPU

Memory on hard disk: 931GB

OS : Windows 7 Enterprise SP1 (64-Bit)

RAM size: 16GB

Processor: Intel Core i7-4770

Virtual Machine 1 (sender):

Memory on virtual disk: 40GB

RAM size: 4096

Number of processors: 1

OS: Ubuntu 64-Bit

Virtual Machine 2 (receiver):

Memory on virtual disk: 40GB

RAM size: 4096

Number of processors: 1

OS: Ubuntu 64-Bit

4.1.1 Communication Algorithm

The algorithm used for implementing this covert channel for sender and receiver are as follows:

Sender:

BEGIN

DELAY for 1 second

INITIALIZE elapsedTime, iterator

SET messageBits, timeLimit

FOR each bit in message (messageBits)

    IF messageBit ==1

        WRITE largeFile within timeLimit

    ELSE

        DO some simple operation until timeLimit

END of FOR-LOOP

DELETE largeFiles

END

Receiver:

BEGIN

INITIALIZE elapsedTime, message, iterator

SET timeLimit,threshold

FOR each bit (messages)

    READ largeFile until timeLimit

    COMPUTE readTime

    IF readTime> threshold

        messageBit = 1

    ELSE

        messageBit = 0

    END of FOR-LOOP

PRINT message

END

4.1.2 Establishing Thresholds

4.1.2.1 File-Size Thresholds

The file size thresholds for both the sender and receiver had to be computed. For the sender, this was done by repeated testing based on the size of the virtual disk assigned to the sender-VM, to determine what size of files would be large enough to cause some contention without crashing the OS. The test was done by writing 2GB files for each bit of the message that was 1- this had no significant effect. This was then increased to 3GB, which made a slight impact, however when 5GB files were written, the results were more accurate.

For the receiver, the file size threshold was established after several runs, using varied file sizes. The test was done with a 2GB file, which when read by the receiver (when in sync with the sender's write operations) did not impact the performance or the receiver's read operation. It was then changed to a 3GB file- this did the job. In trying to obtain an even greater level of precision in the readings, the file size was changed to 4GB, which continuously crashed the VM.

4.1.2.2 Time Thresholds

The time-frame for the synchronized write and read operations by the sender and receiver was set after several test-runs on the sender. Once the time range for the execution of each write was determined, the time pocket was set accordingly. Execution of a write is terminated when it exceeds the time limit allocated to it; accordingly, if the write execution completes early, it has to wait for the time limit to be reached before moving to the next bit.

In the receiver-VM, the threshold for calculating a 1-bit or 0-bit was set after analyzing the time taken to read the file by the receiver, when the sender was not running. Table 1 shows the read times for 3 test runs without the sender,and figure 2 represents the threshold determination.

Table 1. The receiver's read times after 3 runs without the sender

	Time taken to read file by receiver (milliseconds)		
Iteration	**1st run**	**2nd run**	**3rd run**
1	1754.157	1772.065	1863.352
2	1755.605	1755.583	1818.79
3	1751.708	1808.917	1795.594
4	1646.824	1675.628	1758.423
5	1787.865	1763.829	1747.729
6	1942.789	1759.954	1939.81
7	1737.623	1750.646	1741.12
8	1771.026	1794.984	1767.538
Mean:	1768.45	1760.20	1804.04
Max:	1942.789	1808.917	1939.81

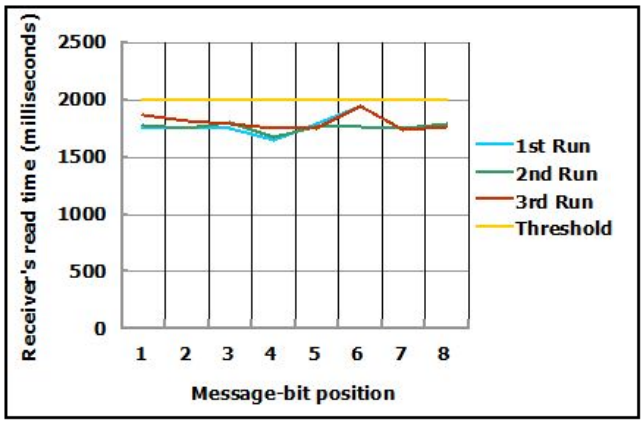

Figure 2. A graph used to determine the threshold for the read time of the receiver-VM

4.2 Results

Based on the experiment design explained in the previous section, several tests were carried out. The results are given subsequently, and based on these numbers, adjustments were made to the programs. Problems which are not solved are discussed later in this paper, giving reasons for such inaccuracies.

4.2.1 Initial Testing

The covert channel was first tested with a 11111111 message being sent by the first VM; figure 3 shows the results obtained.

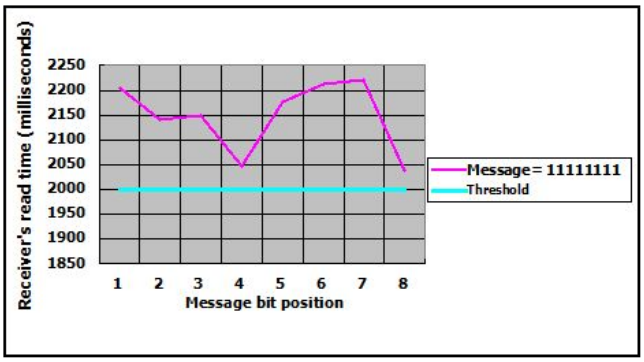

Figure 3. A graph showing the results of the receiver's read times when the message is 11111111. The message is decoded with 100% accuracy.

The message was then varied after the initial test, and the message sent was 11101111. The results as shown in figure 4 were inaccurate. The read time dipped at the 0-bit position, but did not go below the threshold.

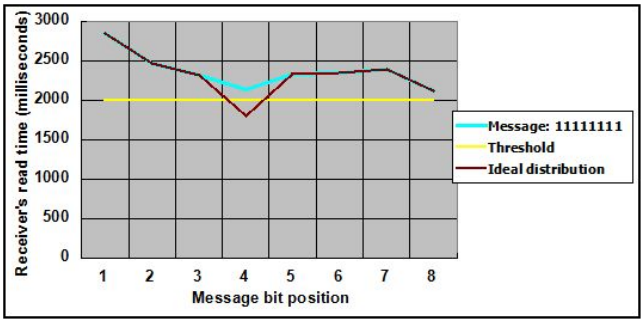

Figure 4. This graph shows the result of the receiver's read times when the message is 11101111. The message is decoded with 87.5% accuracy.

The next step was to increase the number of 0's to 3, to observe if the receiver would respond as wanted (i.e. correctly decoding the message). There was still a lot of noise in the results obtained.

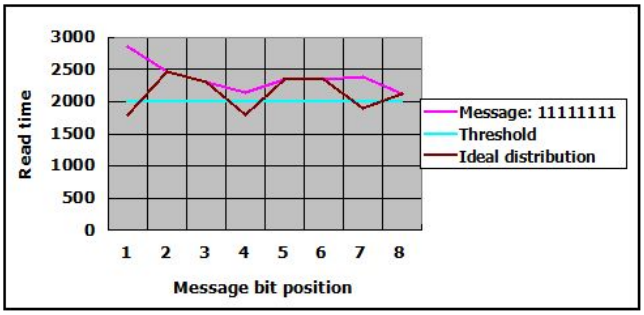

Figure 5. Graph showing results when the message is 01101101. The message is decoded with 62.5% accuracy.

The next messages tested were 00001111 and 00000011. The results are shown in figures 6 and 7.

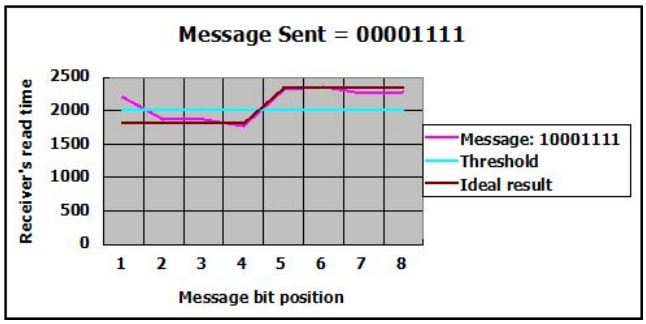

Figure 6. Graph showing results when the message is 00001111. The message is decoded with 87.5% accuracy.

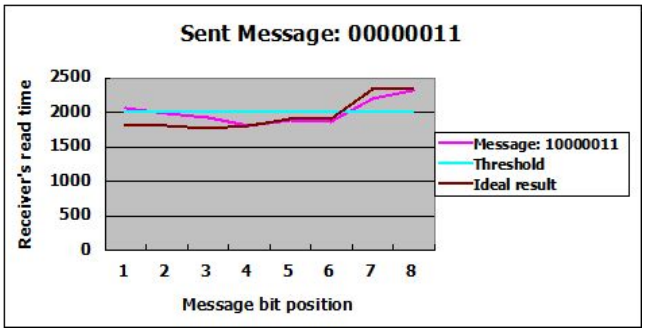

Figure 7. Graph showing results when the message is 00000011. The message is decoded with 87.5% accuracy.

4.2.2 Modifications based on Initial Testing

4.2.2.1 Cache Clearing

We observed that the sender and receiver program times differed significantly from expected results, as we ran our covert channel programs on sender and receiver for the first time in each cycle after switching off the machine or resuming implementation after few hours.

```
sync; /usr/bin/echo 3 > /proc/sys/vm/drop_caches
```

By executing the above command, the kernel is issued an instruction to clear the dirty and clean cache, but it must be noted that this way of cache clearing is not necessarily a means of controlling growth of kernel caches, as they are automatically taken by the system again when memory is needed somewhere else in the memory [4]. By deleting and cleaning page cache, dentries and inodes related cache data from memory, significant RAM memory is freed which makes rest of the system work faster.

```
sender@sender-VirtualBox:~$ free -m
             total       used       free     shared    buffers     cached
Mem:          3952       1284       2668         12         33        478
-/+ buffers/cache:        772       3180
Swap:         4093         54       4039
root@sender-VirtualBox:/home/sender# sync; echo 3 > /proc/sys/vm/drop_caches
root@sender-VirtualBox:/home/sender# free -m
             total       used       free     shared    buffers     cached
Mem:          3952        912       3039         12          0        153
-/+ buffers/cache:        759       3193
Swap:         4093         54       4039
root@sender-VirtualBox:/home/sender# exit
exit
```

Figure 8: This shows free cache memory on the VM before and after the clearing of cache

Our presumption to clear caches before implementing covert channel programs on sender and receiver VMs was that it would speed up access to disk files- and as expected, the results differed significantly.

Table 2. The receiver's read times before and after clearing cache

Message[i]	Before clearing cache	After clearing cache
i	elapsed time (sec)	elapsed time (sec)
0	2205.278	1772.065
1	2139.761	1755.583
2	2148.454	1808.917
3	2047.406	1675.628
4	2176.151	1763.829
5	2213.3	1759.954
6	2220.432	1750.646
7	2038.137	1794.984

4.2.2.2 Sleep()

While the sender is performing certain I/O operations in a given time frame (e.g. x seconds), sender is keeping I/O resources busy, resulting in a time lag which causes other VMs on the same machine to observe latency in performing their usual I/O operations (e.g. accessing a disk file). However, it is important that both sender and receiver are functioning in the same time frames i.e. they stay in sync with each other. We observed that each time we ran our sender/receiver programs, the first bit decoded by the receiver was always greater than the set threshold.

Our receiver program usually lagged by the time it took us to switch to the receiver program after the sender program was made to run. As a result, their times could not be synced because two virtual machines in Oracle's Virtual Box could not be started at the same time. Hence, we used sleep() in our sender program so that receiver and sender would be in sync, after estimating the time lag to be around 1 second.

By putting sleep() in sender program, we made sender wait for receiver program to start running, and implementing sleep() along with cache clearing, helped us filter noise in the first message bit predicted by receiver. Table 3 shows the elapsed times in both cases i.e. Before and after introduced sleep. Figure 7 and 8 show the contrast observed, while figures 9 and 10 show the accuracy of first bits after this modification.

Table 3. The receiver's read times before and after introducing sleep in Sender VM

Message[i]	Before sleep(1s)	After sleep(1s)
i	elapsed time (sec)	elapsed time (sec)
0	2054.539	1967.456
1	1973.424	1905.383
2	1914.991	1877.815
3	1810.499	1762.033
4	1878.459	1912.584
5	1865.076	1956.376
6	2196.845	2125.495
7	2317.187	2203.36
Output:	**10000011**	**00000011**

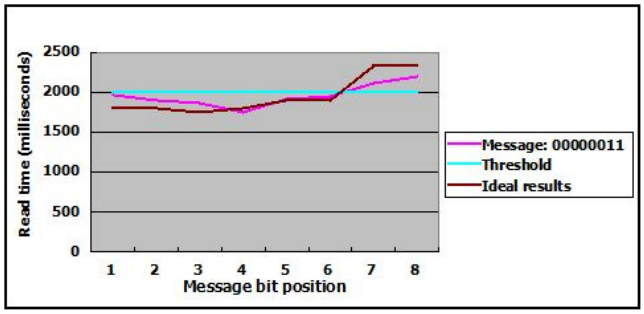

Figure 9. Graph showing results after introducing sleep() with message as 00000011. The message is decoded with 100% accuracy.

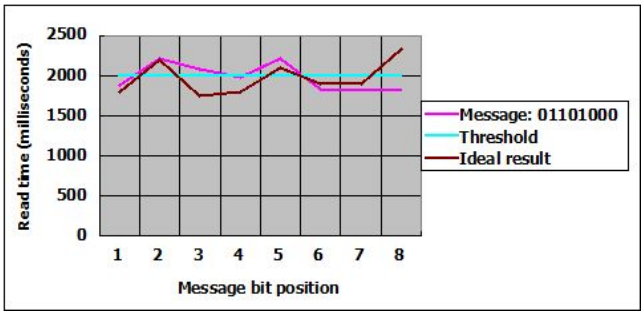

Figure 10. Graph showing results when the message is 01001001. There are 2 noisy bits.

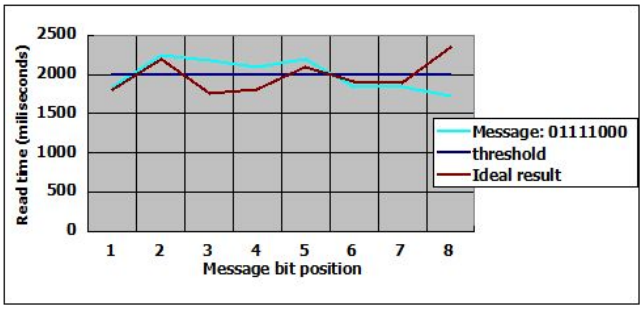

Figure 11. Graph showing results when the message is 01101000. There are also 2 noisy bits.

4.2.3 Interpreting Results

In general, we observed that our covert channel works best when number of one's in message are greater than 4. There were ample test cases in which are covert channel could exactly replicate the sender's message even when the number of 1's in the message were less than 3, but it does not serve as a – medium to confirm if co-residency exists. The reason being, our covert channel is not immune to noise, the message bits at receiver's end are easily changed by user activities being performed in background, by cache as well as some distortion of bits was also observed due to issues in time sync as described in section 4.2.2.2.

However, we believe that the results obtained from our covert channel implementation are quite in sync with what is expected. This is because co-residency test with shared disk in theory, as well as in practice can accurately be achieved when there is significant activity going on at sender's end i.e. shared disk is kept engaged by one VM to an extent that the other VMs on the machine can detect its presence by observing latency times to access resources like files on shared disk.

Consider, message at sender's end is 10010000; it indicates that sender is not performing significant activity in keeping shared disk engaged, and thus other VMs on the machine can also access resources shared disk normally, that is without observing any considerable time lag. In this context, it is also worth noting that in our experiment we have restricted size of files being written by the sender VM, so each 1 in the message corresponds to a fixed size binary file being written to shared disk. However, had the file size varied, it was also possible to overload shared disk with just two 1's in the message and monitor storage performance instead of time latency to detect co-residency.

Also, our covert channel implementation detects co-residency with minimum 75% accuracy where;

Number of 1's in message > 4,

Accuracy = no. of bits detected correctly at receiver's end / no. of bits in message = 6/8 = 75%

We calculate only calculate the minimum accuracy achieved with result set, the range however varied between [75 – 100]%.

The covert channel we implemented lacks proper time syncing, because of which the sender and receiver do not always functionally operate in the same time frames. We used sleep() to sync time between sender and receiver. However, the actual time the program spends sleeping, could also highly be dependent on the program structure with respect to iterations and conditional statements. Furthermore, sender and receiver never start and end at the same time, as we observed in our test cases that sender program always completes executing first. As a result of which, receiver is probably functioning for time equal to lag time in sender's previous time frame, while sender has actually moved ahead by the same lag time in its next time frame, which possibly is the reason in distortion of bits in message. It is also seen that distortion mostly occurs when sender is starting to sleep or write a new file, which favors our assumption that distortion of bits is occurring due to time lag.

4.2.4 Experiment on Cloud Lab

On running the receiver's program on Cloud Lab without Sender, values for thresholds of elapsed time and file size was found to be 6300 ms and 5GB respectively. However, when we attempted to run sender with increased thresholds, i.e. file size = 8 GB, sender VM often threw error in console stating that memory in disk space is less. We deduce that the VM ran out of memory in between the program execution, because our program deletes files written on shared disk after program finishes executing. We then tried launching a new instance of a VM with flavors x1.large and x1.vlarge, however, the instance creation was not successful as we repeatedly received error on OpenStack console that it was unable to allocate floating IP's to the VMs.

5. CONCLUSION

Contrary to the intuitive interpretation of the data collected during the first phase of our testing, the accuracy levels based on the number of 1's introduced into the message works in favor of this covert channel which is also in sync with D'Antoine's work[2]. We claim that our covert channel implementation on Oracle Virtual Box, would run with equal efficiency on Cloud Lab too, as the basic skeleton would remain valid regardless.

6. FUTURE WORK

The following are the potential areas of development in our covert channel implementation:

(i) Proper time syncing : sleep() method is not efficient means if syncing two VM processes in the same time frame.

(ii) Adopting a time syncing approach which can strictly keep sender and receiver programs function in same time frames can increase the efficiency of covert channel implementation by great extent.

(iii) File size restriction: Receiver - Even as the VM on Oracle Virtual Box was allocated 40 GB space, the receiver could not read files greater than 3GB. Due to the restriction on maximum file size, our test could not be performed on reading files with size greater than 3 GB which could also possibly have varied the accuracy of message bits being produced at receiver's end.

7. REFERENCES

[1] Ristenpart, Thomas, et al. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds." *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009.

[2] D'Antoine, Sophia M. *Exploiting processor side channels to enable cross VM malicious code execution*. Diss. Rensselaer Polytechnic Institute, 2015.

[3] Harnik, Danny, Benny Pinkas, and Alexandra Shulman-Peleg. "Side channels in cloud services: Deduplication in cloud storage." *Security & Privacy*, IEEE 8.6 (2010): 40-47.

[4] Maurice, Clémentine, et al. "C5: Cross-Cores Cache Covert Channel."

[5] Delete clean cache to free up memory on your slow Linux server, VPS. Retrieved on 10 December, 2015 from http://www.blackmoreops.com/2014/10/28/delete-clean-cache-to-free-up-memory-on-your-slow-linux-server-vps/