

Hard-disk based covert channel and Load-based co-residence detection

Anil Kumar Konasale Krishna
akumarkk@cs.utah.edu

School of Computing, University of Utah

January 4, 2015

1 Introduction

Covert channel is a hidden communication in which two Virtual Machine(VM) instances cooperate to send messages via shared resources. Covert channel communication is used to extract information from the target VM running on the same physical machine. This can also be used to verify the correctness of network-based co-residence checks as demonstrated in the work [3].

VMs on the same physical machine share the computing resources such as CPU, disks and memory. Hypervisor arbitrates the VM resource contention and provides sequential access to the shared resources. A VM can observe the latency in accessing shared resources to predict the allocation of the same to other VMs. Two processes running on two different VMs use these latency in accessing shared resources to establish covert communication channel.

In Disk-based covert channel, sender(target) VM does nothing to send 0 bit, but it reads from random locations on a shared disk volume to send 1 bit. Listener VM observes the latency for reading from fixed location on the disk volume. Higher latency indicates 1 bit and lower latency indicates 0 bit.

Co-residence of VMs on the same physical machine can be detected using network-based techniques and load based techniques. In Network based co-residency, Dom0 IP address, small packet RTT and numerically close internal IP addresses are used to

detect co-residency. A cloud provider can obfuscate this co-residence by making Dom0 not to respond to traceroute and assign random IP addresses during instance creation. In this case, load based co-residency can be used. An attacker instance can check for co-residence with target instance by observing the differences in load samples taken when externally inducing load on the target versus when not. In my implementation, I have used scp protocol to create variation in load in the target VM.

2 Design

Hypervisor arbitrates the disk contention of multiple VMs by providing sequential access to disk, which results in *leaky* disk abstraction. This has been exploited to establish covert communication channel between two coordinating processes running in different VMs. Whenever a VM requests a disk access, hypervisor provides access to the disk. If two processes/VMs compete for the hard disk access simultaneously, hypervisor will schedule one VM for disk access and other will be kept in wait state. Once first VM completes its disk access, second VM will be granted disk access. Through this sequential access, two VMs can coordinate their read times to communicate data.

For example consider a covert channel between two VMs A and B, to send 1 bit A randomly accesses disk

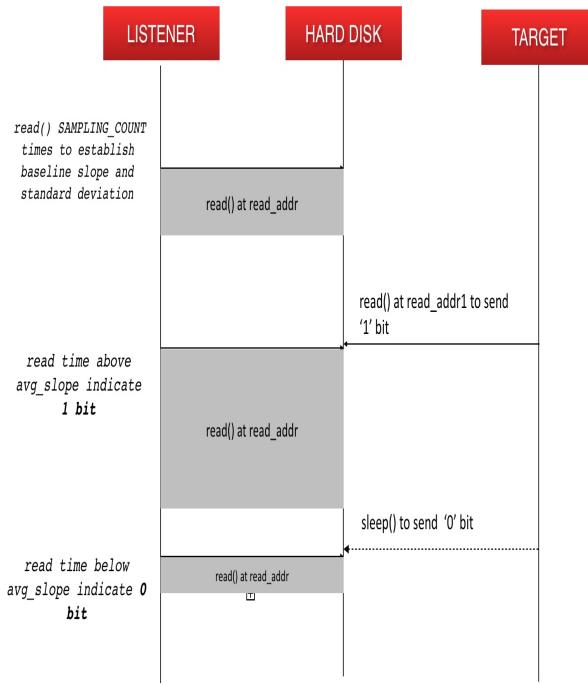


Figure 1: Disk-based covert channel communication

while to send 0 bit, it just does nothing. If B tries to read from the disk when A is using it, it will see very high increase in read time when compared to earlier attempts. This conveys 1 bit to B. If the read time is low, it conveys 0 bit to B. As shown in the figure 1, design of disk based covert channel communication consists of *listener*, *target* programs and hard disk. *listener* program initially take samples to establish baseline slope and standard deviation normalized at 0. once baseline parameters are established, *listener* reads fixed random location on the disk and measure the latency for each *read()*. This low and high latency are used to detect 0 and 1 bit. *target* program reads random locations on the disk to send 1 bit and sleeps

for a second to send 0 bit.

Load based co-residency test In this case, load variation is induced by scp requests and no scp requests. To communicate 1 bit, scp requests will be sent to target which induces sufficiently detectable load on target. To send 0 bit, no requests are sent. By observing these lows and highs during scp requests, one can detect the co-residence. This is similar to the work [3] where HTTP requests are used to induce loads. *listener* used in the previous section is used to detect the lows and highs.

3 Implementation

Implementation consists of two programs for disk-based covert communication is as shown in the Figure 1 :

1. *listener* program is the receiver of data. *listener* program initially establishes baseline parameters - average slope and standard deviation by taking sample read measurements. It takes measurements for SAMPLING_COUNT(10) times. During each second(SAMPLING_TIME), it reads data from fixed random location *read_addr* on the disk and computes latency for *read()*. It establishes the overall average slope and standard deviation using average access times over SAMPLING_COUNT(10) samples. This value is later used to detect the bit. Also it is normalized around 0.

- Bit 0 - If the disk read time for the same random location on the disk is *less than* normalized slope, it is considered as bit 0.
- Bit 1 - If the disk read time for the same random location on the disk is *more than* normalized slope, it is considered as bit 1.

Here is the description of the functions used to implement *listener* functionality :

- *get_read_addr()* This function returns the random read address. It picks a random number as read address returned by *rand()* which is less than maximum size of the disk.

- *get_time_elapsed()* This function returns time elapsed in nanoseconds for a single disk read operation. It basically takes the difference between start and end times.
 - *get_mean_access_time()* This function returns average read time, which is computed by taking average of all the read times in a single SAMPLING_TIME time.
 - *get_baseline_results()* This function takes initial measurements to establish baseline slope and standard deviation. It takes measurements over SAMPLING_COUNT(10) times.
2. *target* program is the sender of data. To send 1 bit, it reads data from a random location on the disk and to send 0, it sleeps for a second. Below is the description of the functions used to implement *target* functionality :

- *get_read_addr()* This function returns the random read address. It picks a random number as read address returned by rand() which is less than maximum size of the disk.
- *send_bit()* It reads random location to send 1 bit and sleeps to send 0 bit.
- *send_data()* This function sends stream of bits using *send_bit()*.

synchronize() is the function which has been used to establish synchronization between *listener* and *target* program. It makes these processes to start communication only at time seconds, which is odd multiple of 5.

Load based co-residence test *ssh_target* program generates ssh load on the target machine. *listener* program is same here also. Following are the library functions specific to the load based test.

- *get_file* This function is used to emit bit 1. It basically spawns scp process to copy file from *target* VM.

4 Evaluation

To evaluate the system, I created two VMs running *Ubuntu x86_64* GNU Linux. *listener* program was instantiated on one VM and *target* program was instantiated on other VM. For both the programs, */dev/sda* was passed as an argument. *listener* correctly detects lows and highs.

- As shown in the Figure 2, *listener* initially establishes average slope and standard deviation which is normalized around 0. Later this will be used as reference value to detect 0 and 1 bits. *It is evident that when there is no target, all the detected bits are 0 as listener is the only instance accessing disk.*
- When *target* starts emitting 0 and 1 bits in terms of high and low disk access times as shown in 5, *listener* has correctly detected the same. This is evident from the Figure 3.
- Again when *target* stops accessing disk or *target* exists, all the detected bits are 0 which is evident from 4.

From the above points, it is clear that algorithm correctly detects lows and highs.

Load based co-residence detection It also consists of two programs - *listener* and *ssh_target*. To evaluate the system, *listener* was ran on one VM and *ssh_target* was ran on guest OS to generate load on target VM. The system has correctly detected load variations.

5 Discussion

Though the implemented algorithm correctly detects lows and highs and establishes covert communication between *listener* and *target*, there are certain issues:

1. *Synchronization* After some significant time, *listener* will go out of synchronization with *target* and as a result x_{th} high bit on *target* might appear as $x + y_{th}$ high bit on *listener*. This is because *listener* spends significant amount of time in computing average mean and deciding the bit

```
Applications Menu Restore Session - Mozilla Fi... Terminal
File Edit View Terminal Tabs Terminal
ubuntu@jgdnhubvm:~/Desktop/akumarlk/covert_channel/fixed[18:25] (master)$ sudo ./blisteren /dev/sda
get_real_addr(52): Size of disk : 21474836480
main([233]: Read addi : 846930886

Creating baseline parameters for read bit detection ...
Please wait ...
get_baseline_results(182): fd = 3 read_addr = 846930886
get_baseline_results(239): baseline std dev = 9332 slope = 38406976
----- BASELINE PARAMETERS -----
std dev : 9332
avg slope : 38406976
synchronize(75): Synchronizing 1420338390...
synchronize(83): covert communication started 1420338395...
----- READ BIT DETECTION -----
curr = 8382101 slope = -2960196 normalized = -2960196 Bit detected -----> (0)
curr = 88647680 slope = 4828649 normalized = 1866653 Bit detected -----> (0)
curr = 93728316 slope = 4630456 normalized = 6497109 Bit detected -----> (0)
curr = 98676918 slope = 3598602 normalized = 10995711 Bit detected -----> (0)
curr = 93527496 slope = -4992272 normalized = 5192150 Bit detected -----> (0)
curr = 95252946 slope = -55670 normalized = 5748289 Bit detected -----> (0)
curr = 91140733 slope = -1388763 normalized = 4359526 Bit detected -----> (0)
curr = 95957239 slope = 4816506 normalized = 9176632 Bit detected -----> (0)
curr = 87898682 slope = -7968575 normalized = 1287475 Bit detected -----> (0)
curr = 89554554 slope = 1760636 normalized = 3078110 Bit detected -----> (0)
curr = 90705354 slope = -145436 normalized = 3924053 Bit detected -----> (0)
curr = 90659944 slope = 954666 normalized = 3878737 Bit detected -----> (0)
curr = 84409345 slope = -6250599 normalized = -2371862 Bit detected -----> (0)
curr = 92916821 slope = 8507476 normalized = 6135614 Bit detected -----> (0)
curr = 91201849 slope = -1715772 normalized = 4419842 Bit detected -----> (0)
curr = 90911130 slope = 1000000 normalized = 1999999 Bit detected -----> (1)
curr = 599011130 slope = 484495633 normalized = 504149923 Bit detected -----> (1)
curr = 86299944 slope = -504631186 normalized = -481263 Bit detected -----> (0)
curr = 85487055 slope = -892889 normalized = -174152 Bit detected -----> (0)
curr = 94776553 slope = 9369498 normalized = 7795546 Bit detected -----> (0)
curr = 808738890 slope = 5589237 normalized = 14577683 Bit detected -----> (0)
curr = 808738890 slope = -1589237 normalized = -7695343 Bit detected -----> (0)
curr = 97203341 slope = -11177844 normalized = 10422334 Bit detected -----> (0)
curr = 99926413 slope = -6277028 normalized = 4145206 Bit detected -----> (0)
curr = 93059254 slope = -2132841 normalized = 6278847 Bit detected -----> (0)
curr = 93973942 slope = 6320188 normalized = 12592835 Bit detected -----> (0)
curr = 88007320 slope = 1373000 normalized = 1236999 Bit detected -----> (0)
curr = 94059331 slope = -5996755 normalized = -728124 Bit detected -----> (0)
curr = 84316045 slope = -9743286 normalized = -2465162 Bit detected -----> (0)
curr = 87874976 slope = 3558931 normalized = 1093769 Bit detected -----> (0)
```

Figure 2: *listener* detecting 0s when *target* is sleeping

							Terminal
File	Edit	View	Terminal	Tabs	Help		
curr = 07956682	slope =	2595400	normalized =	11215475	Bit detected	(0)
curr = 90571212	slope =	7425470	normalized =	3790095	Bit detected	(0)
curr = 82310407	slope =	-8260715	normalized =	-4470710	Bit detected	(0)
curr = 95485335	slope =	13174838	normalized =	8704128	Bit detected	(0)
curr = 87345720	slope =	-8139615	normalized =	564513	Bit detected	(0)
curr = 33430050	slope =	246954810	normalized =	247519323	Bit detected	(1)
curr = 345970384	slope =	11669854	normalized =	259189177	Bit detected	(1)
curr = 95328599	slope =	256841785	normalized =	8457932	Bit detected	(0)
curr = 30836163	slope =	24561800	normalized =	25271800	Bit detected	(0)
curr = 10523911	slope =	23303188	normalized =	18457976	Bit detected	(0)
curr = 618797343	slope =	515598160	normalized =	53201636	Bit detected	(1)
curr = 356482221	slope =	-262315122	normalized =	269701014	Bit detected	(1)
curr = 105261856	slope =	251220365	normalized =	18480649	Bit detected	(0)
curr = 104855241	slope =	-406615	normalized =	18074034	Bit detected	(0)
curr = 134687498	slope =	2983257	normalized =	47966291	Bit detected	(1)
curr = 138862162	slope =	-2030100	normalized =	27605010	Bit detected	(0)
curr = 11200009	slope =	-2082292	normalized =	25270000	Bit detected	(0)
curr = 144059212	slope =	31328317	normalized =	57278005	Bit detected	(1)
curr = 1290650798	slope =	1146591586	normalized =	1203869591	Bit detected	(1)
curr = 134624417	slope =	-1156206381	normalized =	47843210	Bit detected	(1)
curr = 98692916	slope =	-35931501	normalized =	11191799	Bit detected	(0)
curr = 112694655	slope =	14001739	normalized =	25913448	Bit detected	(0)
curr = 2386214548	slope =	1259268093	normalized =	151840251	Bit detected	(1)
curr = 122306654	slope =	-116315464	normalized =	35252487	Bit detected	(0)
curr = 122306654	slope =	1259268093	normalized =	43210000	Bit detected	(0)
curr = 106223725	slope =	-23665195	normalized =	19442518	Bit detected	(0)
curr = 116204317	slope =	9980592	normalized =	29423110	Bit detected	(0)
curr = 125857129	slope =	6952812	normalized =	39075922	Bit detected	(1)
curr = 99874943	slope =	-25982186	normalized =	13093736	Bit detected	(0)
curr = 131968236	slope =	32031293	normalized =	45125029	Bit detected	(1)
curr = 123560702	slope =	-8345534	normalized =	37769495	Bit detected	(0)
curr = 1008868	slope =	-15702616	normalized =	22100000	Bit detected	(0)
curr = 141438494	slope =	2519378	normalized =	56442047	Bit detected	(0)
curr = 177155031	slope =	-24275463	normalized =	30373824	Bit detected	(0)
curr = 109021862	slope =	-8131969	normalized =	22446655	Bit detected	(0)
curr = 3377596154	slope =	228574292	normalized =	250819497	Bit detected	(1)
curr = 83000319	slope =	-254595835	normalized =	-3780888	Bit detected	(0)
curr = 83506321	slope =	506002	normalized =	-3274886	Bit detected	(0)
curr = 84869616	slope =	1469595	normalized =	-1814293	Bit detected	(0)
curr = 81018304	slope =	3708289	normalized =	-26474274	Bit detected	(0)
curr = 83529682	slope =	2519378	normalized =	-5770903	Bit detected	(0)
curr = 85232316	slope =	1702634	normalized =	-3251525	Bit detected	(0)
curr = 84698937	slope =	-533379	normalized =	-2082276	Bit detected	(0)

Figure 3: *listener* detecting the pattern emitted by *target*

after each second. I tried to improve this by im-

```
File Edit View Terminal Tabs Help Terminal
curr = 158092367 slope = 33031062 normalized = 71221100 Bit detected . . . . . (1)
curr = 183627781 slope = -54375026 normalized = 16846074 Bit detected . . . . . (0)
curr = 185612857 slope = -1985576 normalized = 18831650 Bit detected . . . . . (0)
curr = 610109325 slope = 504496468 normalized = 523238118 Bit detected . . . . . (1)
curr = 86972434 slope = -521336891 normalized = 191227 Bit detected . . . . . (0)
curr = 821504949 slope = -4821985 normalized = -4630758 Bit detected . . . . . (0)
curr = 84295195 slope = 2144746 normalized = -2486012 Bit detected . . . . . (0)
curr = 86247723 slope = 1952530 normalized = 533492 Bit detected . . . . . (0)
curr = 897050111 slope = -3071110 normalized = -1060061 Bit detected . . . . . (0)
curr = 91600615 slope = -3314988 normalized = -5180952 Bit detected . . . . . (0)
curr = 91363657 slope = -236958 normalized = -5417550 Bit detected . . . . . (0)
curr = 83897357 slope = 2533700 normalized = -2883850 Bit detected . . . . . (0)
curr = 84095106 slope = 197749 normalized = -2686101 Bit detected . . . . . (0)
curr = 85132942 slope = 1037836 normalized = -1642625 Bit detected . . . . . (0)
curr = 85497198 slope = 364256 normalized = -1284009 Bit detected . . . . . (0)
curr = 85139760 slope = -357498 normalized = -1641509 Bit detected . . . . . (0)
curr = 85200011 slope = -207000 normalized = -309096 Bit detected . . . . . (0)
curr = 85615766 slope = 2652475 normalized = -1165531 Bit detected . . . . . (0)
curr = 87103892 slope = -1488216 normalized = -322685 Bit detected . . . . . (0)
curr = 85402527 slope = -1761365 normalized = -1378868 Bit detected . . . . . (0)
curr = 85095981 slope = -306546 normalized = -1685226 Bit detected . . . . . (0)
curr = 82600575 slope = -2495406 normalized = -4180632 Bit detected . . . . . (0)
curr = 81946194 slope = -654381 normalized = -4835013 Bit detected . . . . . (0)
curr = 82753228 slope = 807034 normalized = -4027979 Bit detected . . . . . (0)
curr = 88500011 slope = -100000 normalized = -686000 Bit detected . . . . . (0)
curr = 82819142 slope = -215087 normalized = -3962065 Bit detected . . . . . (0)
curr = 85715651 slope = 2896599 normalized = -1065556 Bit detected . . . . . (0)
curr = 82454834 slope = -1460817 normalized = -2526373 Bit detected . . . . . (0)
curr = 83108611 slope = -1146223 normalized = -3672596 Bit detected . . . . . (0)
curr = 86902524 slope = 2983913 normalized = -688683 Bit detected . . . . . (0)
curr = 84266468 slope = -1826056 normalized = -2514739 Bit detected . . . . . (0)
curr = 80967994 slope = -3298474 normalized = -5813213 Bit detected . . . . . (0)
curr = 80734496 slope = -23594 normalized = -8046711 Bit detected . . . . . (0)
curr = 89010001 slope = -100000 normalized = -686000 Bit detected . . . . . (0)
curr = 91157114 slope = -1053964 normalized = -5624993 Bit detected . . . . . (0)
curr = 81147367 slope = -9747 normalized = -5633840 Bit detected . . . . . (0)
curr = 82780669 slope = -1633302 normalized = -4000538 Bit detected . . . . . (0)
curr = 81925992 slope = -854677 normalized = -4855215 Bit detected . . . . . (0)
curr = 82514062 slope = -588070 normalized = -4267145 Bit detected . . . . . (0)
curr = 81221191 slope = -1292871 normalized = -5560016 Bit detected . . . . . (0)
curr = 88739375 slope = -210000 normalized = -3407671 Bit detected . . . . . (0)
curr = 82162318 slope = -1249715 normalized = -4603999 Bit detected . . . . . (0)
curr = 93407229 slope = -1283418 normalized = -373979 Bit detected . . . . . (0)
*Ubuntu@dnhubwvm:~/Desktop/akumarck/covert_channel/fixed[18:32] (master) 5
```

Figure 4: *listener* detecting 0s when *target* exited

```
Applications Menu Terminal Terminal
File Edit View Terminal Tabs Help
ubuntu@ndnhubvm:~$ sudo ./target /dev/sda
Waiting ...
----- Synchronizing(61): Sending data over covert channel -----
----- Synchronizing(69): Synchronizing 1420338658...
----- Synchronizing(69): covert communication started 1420338645...
Sending bit -----> (0)
sending bit -----> (0)
sending bit -----> (0)
sending bit -----> (1)
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1804289383
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 846930886
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 169592377
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1714636915
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1957747793
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 4242383535
Sending bit -----> (0)
sending bit -----> (1)
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 719888186
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1649760492
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 5965166449
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1000000000
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1025202362
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1350496027
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1350496029
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1102520509
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 2044897763
get read addr(39): Size of disk : 21474836480
get read addr(53): rand num = 1967313926
sending bit -----> (0)
```

Figure 5: *target* communicating bits

plementing synchronization method - emitting a bit only at odd seconds and reading a bit only at

odd seconds. But this has not worked as clocks are really different in each VMs.

2. *Disturbance from other processes* Disturbance rate or scheduling frequency of *target* and *listener* is different as they run on different VMs. So *target* might emit a bit and *listener* might have not scheduled during that time and might be scheduled little lately which results in unexpected/erroneous average slope and hence erroneous bit detection. I tried to address this problem by scheduling the processes at high priority using Linux *nice* utility. Improvement was not consistently significant all the time.
3. *Load based co-residence test* In this test, there is no strict pattern possible unlike disk-based covert communication. This is because secure copying files from target VM is not accurately predictable. But *listener* program generates a bit once in each second.

6 Helpful resources

Undermining isolation through covert channels [2] and CloudSteg [1] were really helpful in understanding covert communications and different methods to implement the same.

References

- [1] LIPINSKI, B., MAZURCZYK, W., AND SZCZYPIORSKI, K. Improving hard disk contention-based covert channel in cloud computing environment. *arXiv preprint arXiv:1402.0239* (2014).
- [2] PETER, M., NORDHOLZ, J., PETTSCHICK, M., DANISEVSKIS, J., VETTER, J., AND SEIFERT, J.-P. Undermining isolation through covert channels in the fiasco. oc microkernel.
- [3] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2009), CCS '09, ACM, pp. 199–212.