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

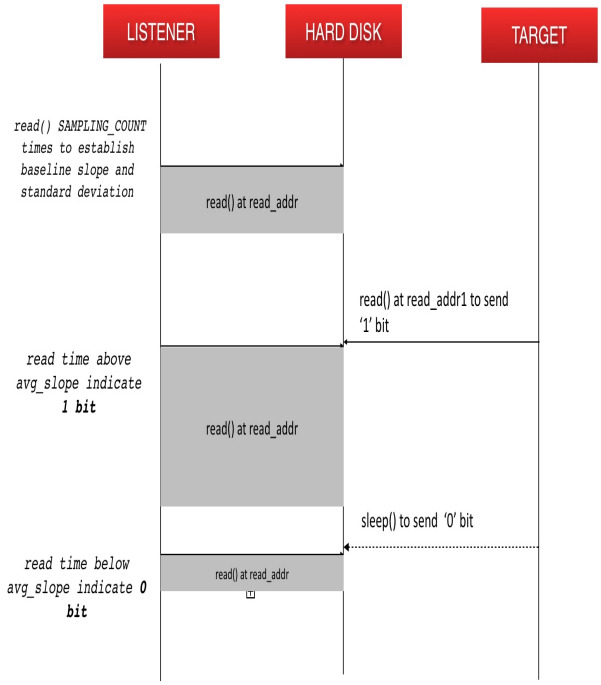

Figure 1: Disk-based covert channel communication

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

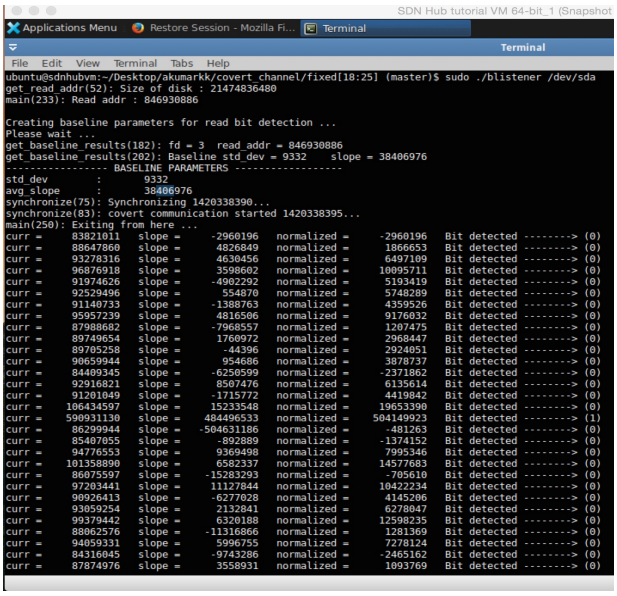

Figure 2: *listener* detecting 0s when *target* is sleeping

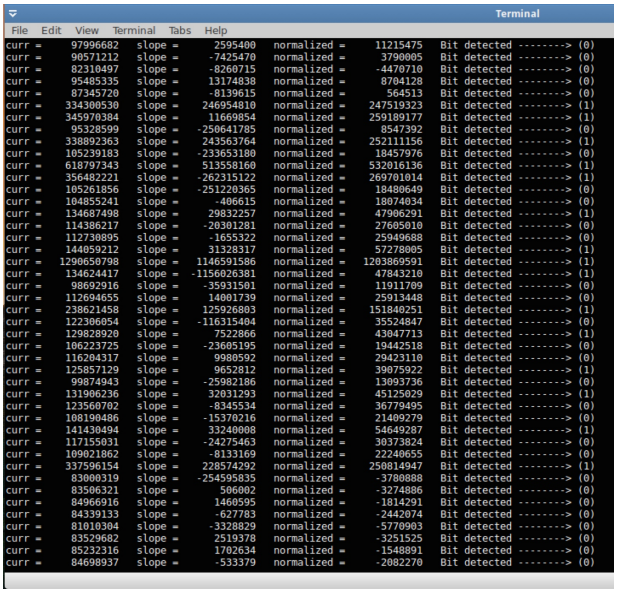

Figure 4: *listener* detecting 0s when *target* exited

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

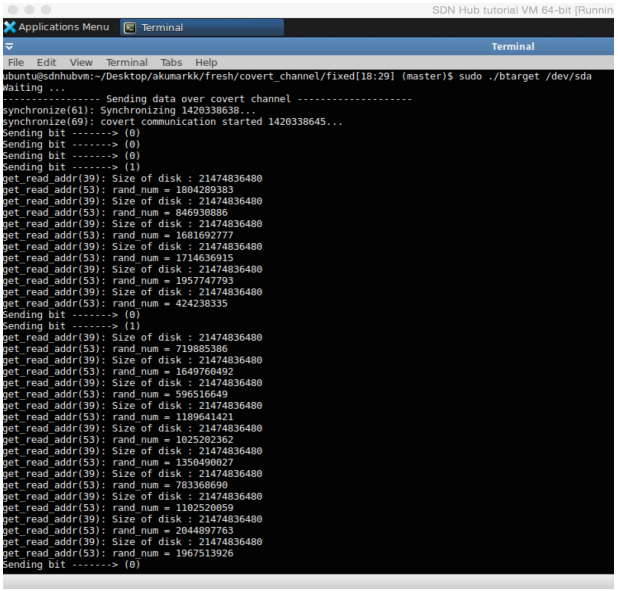

Figure 5: *target* communicating bits

after each second. I tried to improve this by implementing 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., Petschick, 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.