Dynamic Parameter Reconnaissance for Stealthy DoS Attack within Cloud Systems

Suaad Alarifi, Stephen Wolthusen

Dynamic Parameter Reconnaissance for Stealthy DoS Attack within Cloud Systems

Suaad Alarifi[2,3] and Stephen Wolthusen[1,2]

[1] Norwegian Information Security Laboratory,
Department of Computer Science,
Gjøvik University College, Norway
[2] Information Security Group,
Department of Mathematics,
Royal Holloway, University of London, UK
[3] King Abdulaziz University, Jeddah, Saudi Arabia
Email: ⟨s.alarifi, stephen.wolthusen⟩@rhul.ac.uk

Abstract. Public IaaS cloud environments are vulnerable to misbehaving applications and virtual machines. Moreover, cloud service availability, reliability, and ultimately reputation is specifically at risk from Denial of Service forms as it is based on resource over-commitment.
In this paper, we describe a stealthy randomised probing strategy to learn thresholds used in the process of taking migration decisions in the cloud (i.e. reverse engineering of migration algorithms). These discovered thresholds are used to design a more efficient, harder to detect, and robust cloud DoS attack family. A sequence of tests is designed to extract and reveal these thresholds; these are performed by coordinating stealthily increased resource consumption among attackers whilst observing cloud management reactions to the increased demand. We can learn the required parameters by repeating the tests, observing the cloud reactions, and analysing the observations statistically. Revealing these hidden parameters is a security breach by itself; furthermore, they can be used to design a hard-to-detect DoS attack by stressing the host resources using a precise amount of workload to trigger migration. We design a formal model for migration decision processes, create a dynamic algorithm to extract the required hidden parameters, and demonstrate the utility with a specimen DoS attack.
Keywords : CIDoS, IaaS security, Cloud Computing Security, Migration Security in the Cloud.

1 Introduction

Attacks specific to cloud infrastructure have recently gained attention [1, 2]. Most of this earlier work targeted availability which affects reliability and might cause Service Level Agreement breaches.

High availability is critical in the cloud and it is a main concern for enterprises moving to cloud. There is competition between providers for high availability; they publish annually service outage status reports, where SLA only specify minimum availability requirements. In this paper, we design a stealthy randomising testing strategy to learn thresholds that are used in the process of taking migration decisions (i.e. reverse engineering of migration algorithms). We perform a series of tests to reveal these thresholds; these tests are based on a cloud specific DoS attack called Cloud-Internal Denial of Service (CIDoS) described earlier [3] together with statistical analysis. The attack mis-uses *migration* and *over-commitment* features, which are essential to cloud systems as permits elasticity, allowing virtual machines (VMs) to expand. Cloud systems rely on resource sharing to reduce cost by maximising utilisation of cloud hosts.

If one of the VMs in the highly utilised host decides to expand, some of the co-resident VMs may be migrated to provide space for expansion. Moreover, as stated in [4] "*the host is oversubscribed; that is, if all the VMs request their maximum allowed CPU performance, the total CPU demand will exceed the capacity of the CPU*"; this is what we call misusing *over-commitment* and it is most harmful when there is *rapid* coordination between a group of malicious VMs.

In CIDoS, m co-resident VMs increase their workload to reach a threshold (time and strength) to trigger migration. In [3], these thresholds were assumed known to the attackers while in reality they are hidden and discovering them requires designing a new attack which we describe in this paper. We perform tests to reveal some of the cloud migration parameters that are used by migration algorithms. We design a formal model for migration decision process then create a dynamic algorithm to extract the required parameters. The mechanisms to extract these thresholds are adapts to dynamic changes in cloud algorithms. Revealing parameters is hence a security threat by itself; moreover, these can be used by malicious VMs to accurately generate the needed workload to repeated trigger migration resulting in *thrashing*. It is vital that the generated workload is no more than required to avoid detection and make the attack live longer. Conversely, attackers may also use the revealed parameters to avoid being migrated as we will see later. The rest of the paper is structured as follows. CIDoS attack is explained in section 2. Section 3 is the threat model while the literature review is in section 4. The attack is discussed in section 5; analysis and discussion are shown in section 6. Finally, section 7 covers conclusion and future work.

2 Attack Mechanism Outline

In the CIDoS family of attacks, attackers are assumed to be co-resident VMs; these VMs w.l.o.g. co-ordinate their consumption of host resources following a pattern distributed by the attack leader. The pattern is designed so that the total of the attackers' resource consumption plus the regular consumption will break a threshold causing migration for some of the VMs to balance the load in the over-utilised host. The migration process is targeted as it has a relatively high cost for the cloud operator (network, host utilisation, and ultimately energy), and may affect availability and reliability, also threatening SLA breaches. VMs can be migrated either *live* or *offline*, but both migration methods are expensive, particularly where service dependencies exist. Online migration implies migrating even the memory while it is running and in use and offline migration implies turning the VM off, disrupting services.

If attackers successfully force the cloud to enter a continuous state of migration, the whole service will have difficulty functioning. Furthermore, more dramatic orders might be issued such as turning *on* a new host or evacuate a host and turn it *off* to save energy which cost more than regular migration.

To prepare for the attack, there should be m malicious co-resident VMs; achieving co-residency is popular topic in cloud attacks; many techniques are suggested in [1, 3] to attain co-residency or increase its possibility i.e. by misusing the placement algorithm (VMs distribution algorithm) or by using brute force strategy (create large number of VMs in the same area and terminate not co-resident ones).

After achieving co-residency, the attack leader has to create a covert channel to communicate and coordinate the attack among malicious VMs. Covert channels are usually used in cloud attacks, see [1, 2]. Then the leader checks the capacity of the covert channel; if the covert channel is too narrow, the attack will be converted to a brute force scenario. In brute force, the leader only distributes the signal "attack now" to all participants and they in turn increase their consumptions as a response to the attacking order then the host might not be able to cope with the stress and there will be a wave of migration. This scenario is easy to detect and block by security defences in the cloud; regular host based intrusion detection system, HIDS, that is anomaly based can detect the sudden change in behaviour with high accuracy. Brute force scenario is not practically strong but it is a solution for very narrow covert channels.

If the covert channel is wide enough (over a threshold based on i.e. the amount of data need to be sent, the value of m and the available time to coordinate the attack), the leader will check the available number of co-resident malicious VMs (m); if m is *just over* the required number of VMs to form the attack ($m >= T_1$), then apply scenario 2. If m is *far over* T_1 (over another bigger threshold T_2, $m >= T_2 > T_1$) then go to scenario 1. T_1 and T_2 are safe thresholds that are calculated based on the host specifications, high specifications (i.e. number of processors and their speeds) means bigger T_1 and T_2 values [3].

Scenario 1: Attack pattern random sampling

In this scenario, there is no scarcity of co-resident malicious VMs, therefore, there is no need for a neat distribution of the workload between attackers. Attackers will benefit from abundance to make it a more robust attack. Each malicious VM decides locally its part of the attack (the strength and timing of the workload waves). These choices are not completely random but they rely on two factors; first the peaks should be around specific points distributed by the leader but where exactly? and how strong they are? are decided locally. Second, the workload pattern of each of the attackers should be as close as possible to its previous workload (history) to avoid being caught by anomaly based HIDS.

So each VM decides locally but they are following the same plan distributed by the leader and the sum of the workloads should trigger migration by breaking the severity threshold T_s which is also distributed by the leader. The calculation of T_s will be discussed later in the paper.

Scenario 2: Spread-spectrum attack distribution

Because of the shortage of participants in this scenario, the leader should design the attack neatly and each participant should know *exactly* where and how strong is the peaks that he/she should create. The leader predicts the normal workload pattern of the host for the next short interval of time and also predicts the attack pattern (the amount of workload need to be added to the predicted workload to break the severity threshold T_s); The leader also has to calculate the value of T_s and distribute it among attackers, see figure 1; the highlighted area in the figure will be divided into units and distributed among attackers; the distribution is by using spread-spectrum technique, see [3] for details.

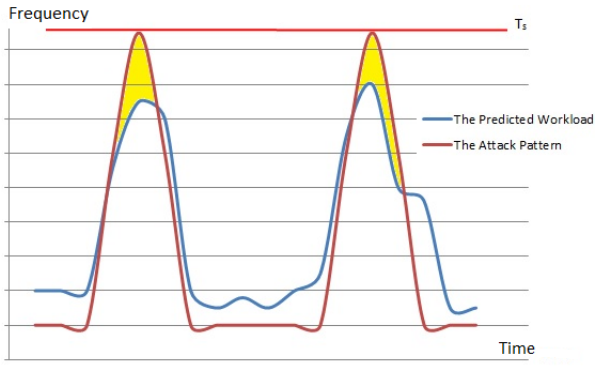

Fig. 1. Qualitative illustration of spread-spectrum attack [3].

The attack is coordinated by a protocol designed specifically for this task. The protocol establishes a secure communication channel among attackers using Group Key Agreement protocol. It also synchronises time among VMs with the consideration of Packet Delay Variation, tolerates failures, hides the identity of attackers, authenticates participants, and resists number of attacks i.e. reply attack, see [3] for details.

3 Threat Model

The primary motivation is designing a stealthy testing strategy to reveal migration algorithms parameters which can be used to improve CIDoS attack, make it more harmful and harder to detect. We succeed if we can extract these parameters and calculate their reliability. We target large-scale public cloud.

In the attack there are: the attack leader (i.e. the last to arrive malicious VM), other co-resident malicious VMs, the cloud host where malicious VMs live together with other innocent VMs, and a cloud management node where hosts management algorithms are run.

In our model we have the following assumptions; first, we assume that migration feature is *enabled* in the cloud service (some providers disable migration support for security and performance reasons but as we said earlier it is a *main* feature of the cloud). Second, we assume that the attack leader can monitor the host workload to be able to predict the workload pattern for future short interval of time and to be able to build an anomaly host based intrusion detection system (we will see its need later in the paper). This assumption is realistic because the host workload can be monitored by gathering observations from the environment or performing tests and measure the respond time of the host; furthermore, techniques from [2,5] can also be used to monitor the workload. Third, we assume that virtual servers have relatively steady workload pattern (mixing services in one VM is against best practice especially that the cost model is pay-for-use [3]). Furthermore, for simplicity we assume cloud hosts are homogeneous (same specifications), however, if this is not the case the attack still work but instead of discovering the host specifications only once for the whole cloud, attackers should perform this task once per host.

The attack propagation mechanism is not discussed here; several of them are described in [3].

4 Literature Review

There are number of cloud DoS attacks in the literature such as in [3, 6], however, to the best of our knowledge, the proposed attack is new and heavily based on the understanding of migration management algorithms. One of the main challenges we had that migration policies and algorithms used by today's cloud service providers are not publicly revealed; however, many research papers are investigating them.

VM migration is the process of transferring a whole VM (including the running memory) from host to another for various reasons which are: to save energy thus reduce cost by evacuating and turning *off* (or sleep mode) low utilised hosts, for fault tolerance when dealing with faulty or malicious VMs, for maintenance reasons, and to reduce the load in over-utilised host thus avoid SLA violation.

Migration has high cost and should be used with caution; it introduces many challenges in security and performance i.e. minimising migration time to avoid consuming the network. In [7] they found because of live migration, applications performance degraded by 10%. VMs also have to be secured during migration.

Migration mechanisms: Nodes management servers are responsible of managing migration depending on different factors such as utilisation and power-consumption. Status reports have to be collected from each host periodically to show its general status. The data from these reports are the inputs for a collection of migration algorithms. Many different algorithms are used in the cloud; we will discuss some of the most popular ones which are: overload detection, VM selection, and VM placement algorithms.

Beloglazov *et al.* in [4] proposed algorithms based on dynamic measurements generated by statistically analysing historical data. They also proposed four methods to detect overloaded hosts which are Median Absolute Deviation (specify the value of upper utilisation based on CPU utilisation deviation strength), Interquartile Range, Local Regression, and Robust Local Regression. For VMs selection they proposed three polices: migrate VMs with the minimum migration time calculated based on memory usage and NT

bandwidth, random selection based on uniformly distributed discrete random variable, migrate VMs with the highest CPU utilisation correlation with other VMs calculated using multiple correlation coefficient. For VMs placement problem, they sorted VMs based on their CPU utilisations and allocated them to hosts that provide the minimum cost in term of power consumption using Best Fit Decreasing algorithm.

Another research, [8], also used dynamic utilisation thresholds to detect hosts overloading. The dynamic thresholds were calculated based on workload history (statistical analysis); Bala *et al.* measured the statistical dispersion using Median Absolute Deviation. For VM selection, *multipath correlation coefficient* had been used to describe relationship between measurements; these measurements were grouped in different level, each level affect the subsequent ones. The machine with the minimum expected workload and has the least influence on others is migrated (VMs with zero inter-correlation factors can be migrated). This policy has reduced the migration time and the number of migrations needed.

Overload detection: Upper utilisation threshold can be set to decide if the host is overloaded or not; however, as stated in [4] "fixed utilization thresholds are not efficient for IaaS environments with mixed workloads that exhibit nonstationary resource usage patterns"; they suggested dynamic thresholds. Prediction algorithms are also needed to create these dynamic thresholds; for prediction, different techniques are used such as statistical analysis or machines learning algorithms [4, 9].

VM selection algorithm: Many policies are applied such as migrate the minimum number of VMs, the least active VM, a VM randomly, or the VM with the highest correlation [4, 10]. Discovering the used policy can help revealing the required parameters and improve the CIDoS as we will see later.

Placement algorithm: The new location can be chosen based on different factors [4]; the most popular ones are to reduce power consumption and utilisation reasons (i.e. the minimum utilised host).

5 Estimating Cloud Migration Parameters

Attackers aim to extract some of the main parameters used by migration algorithms and use them to build more efficient and harder to detect DoS attack. Because of the power saving policy, the number of running hosts is dynamic and, as a consequence, the used thresholds are dynamics and the process of extracting parameters has to be dynamic too. Furthermore, we need to discover these thresholds as fast as possible because of the dynamicity of the environment. We also need to extract the required thresholds, measure their reliability, reduce the probability of accidental errors, consider the noise, consider not changing the host behaviour heavily to avoid affecting prediction algorithms, and measure the success of the attack. We dealt with this problem as a regular statistical experiment.

Extract the required parameters: this task is accomplished by reverse engineering the algorithms responsible for migration decisions. We start by designing a formal model of migration decision process.

Migration decision process: for the shortage of space we consider over-loaded host detection migration policy..The host management node collects status reports periodically from all hosts. Data from status reports and other data from the environment are the inputs of the algorithms. Then an algorithm runs to decide whether the host under examination is over-loaded or predicted to be overload (to prevent over-utilisation before it occurs). The output of the algorithm is *zero* (if the host is not over-loaded) or *one* (if the host is over-loaded). If the output is *one* the management node will run VM selection algorithm to decide the best candidate VMs for migration. Then VMs placement algorithm will run to choose the best candidate destination hosts for VMs under migration. Lastly, VMs will be migrated either online or offline. The inputs of the over-load detection algorithm are: -the general overall status of all hosts in the same availability zone, -the specifications of the host under examination, -history, current and predicted

CPU utilisations, -history, current and predicted memory utilisations, -history, current and predicted network traffic rate to and from the host, -possible errors, -time, and -hidden unknown variables.

We target large-scale public cloud; usually the effects of change on the general overall hosts status are not dramatic so it can be represented by a constant value (i.e. the effect on migration decision when having 1000 or 1003 hosts *on* is too low). We assumed that the hosts are homogeneous, therefore, the effect of host specifications can be constant value too. The error can be reduced by replicating the test many times and use hypothesis testing to decide whether to accept the revealed parameters or not (as we will see later). For simplicity, we only consider history, current and predicted CPU utilisations and time.

There are many methods for CPU utilisations prediction most of them are based on the history of the host; Multivariate Linear Regression model, MLR, can be used to perform the prediction as in [9]. History of CPU utilisations are partitioned into intervals and analysed to measure "how closely prediction matches observed utilization across the utilisation spectrum" [9].

Algorithm 1 is for over-loaded host detection; the algorithm notations are: -$x_{history}$, $x_{current}$, and $x_{predicted}$ are CPU utilisation history, current, and prediction, -$x_{current}$ is the current CPU utilisation, -x_{time} is the time, -α is the constant value, -upperU is the dynamic upper utilisation threshold f ($\alpha+\beta_1 x_{current}+\beta_2 x_{history}+\beta_3 x_{time}$), -mig is a Boolean variable which is set to '1' if a migration required, and -if the current or predicted CPU utilisation is over upperU, migration is required.

Algorithm 1 Over-loaded Host Detection

Input: α, x_{time}, $x_{history}$, $x_{current}$, $x_{predicted}$ **Output:** mig
mig = 0
upperU = UpperThreshold(α, x_{time}, $x_{history}$, $x_{current}$)
if $x_{current}$ or $x_{predicted} \geq$ upperU **then**
    mig = 1
end if
Return mig

Cloud migration parameter estimation: We developed an algorithm to extract the required parameters, see algorithm 2. First, the attack leader specifies the range of the test (the minimum, maximum CPU utilisation and time) that might cause migration (this is the *test range*). Then the leader designs a series of all possible test phases, portions them into chunks, and gathers them into a list called *chunksList*; each item in the list is called *chunk* and it has two variables, $x_{current}$ and x_{time}. The inputs of the algorithm are *chunksList* and $Bprofile_{normal}$ which is the profile of the host normal behaviour before the attack. The algorithm then tests the chunks in the list one by one until finding parameters that cause migration. *CIDoS.run* is a function with two arguments (CPU utilisation and time) to run a phase of CIDoS attack (the whole attack that has been described in 2). This function is responsible of coordinating malicious VMs resource consumption to stress the host and cause migration. It attacks using the time and strength passed to it in the variables $chunk.x_{current}$ and $chunk.x_{time}$.

To be able to measure the success of the attack (does the tested parameters cause migration or not?), we create a new behaviour profile (for after attack) using the function *updateProfile* and calculate the distance between the old normal behaviour profile and the new normal behaviour profile using the function *compare(profile1, profile2)*. The function *updateProfile* is a regular anomaly based IDS to detect anomalies in the workload pattern of the host i.e. Hidden Markov Model based IDS (more details are shown later in the section). The behaviour profile is updated using fresh data (newly collected data from the current workload). The result of comparison between profiles is in the variable *SuspicionValue*; if *SuspicionValue* is greater than the threshold *simThreshold* that means the host behaviour has changed (probably because

Algorithm 2 Dynamic attack permitting cloud migration hidden parameter estimation

```
1: Input: chunksList, Bprofile_normal Output: x_current, x_time
2: for chunk in chunksList do
3:     wait()
4:     CIDoS.run(chunk.x_current, chunk.x_time)
5:     Bprofile_current = updateProfile ()
6:     SuspicionValue = compare (Bprofile_current, Bprofile_normal)
7:     if SuspicionValue > simThreshold then
8:         successCounter = 0
9:         for i = 0 → replicationNum do
10:            x_current1 = increment x_current
11:            wait()
12:            Bprofile_normal = updateProfile()
13:            CIDoS.run(chunk.x_current1, chunk.x_time)
14:            Bprofile_current = updateProfile()
15:            SuspicionValue = compare (Bprofile_current, Bprofile_normal)
16:            if SuspicionValue < simThreshold then
17:                successCounter = successCounter + 1
18:            end if
19:        end for
20:        if successCounter ≥ successThreshold then
21:            Return x_current, x_time
22:        end if
23:    end if
24: end for
```

of migration) so initially accept the tested parameters and replicate the test *replicationNum* number of times to increase reliability and reduce the effect of accidental errors. *successCounter* is the number of successful replications, if it is greater than or equal to a threshold *successThreshold* then accept the tested parameters as reliable and exit the algorithm. The function *wait()* is to create a gap of time between test phases to avoid affecting the prediction algorithms.

If a VM has been migrated in the middle of the tests for another non malicious reason, this will not affect the reliability of the test because it will be discovered in the replications, as we said replications are to increase reliability and decrease the effect of accidental errors. The attacker need m malicious VMs to attack with, the value of m can be calculated depending on the host specifications. The stronger the host the more malicious VMs are needed to attack.

The time complexity of the algorithm depends on how many tests are needed, the size of the gap between tests and how many replicates should we make to increase the reliability of the result. The smaller *test range*, the less number of tests are needed. Furthermore, if we distribute the tests among different hosts we can perform them on parallel and the gap of time will be less or there will be no need for gap at all, however, the communication between attackers through the network increases the possibility of being caught by network based IDS.

How to check migration: the leader checks migration (measure the success of the attack) by building normal behaviour profiles for the host workload before and after each phase of the test; if there a deviation in the workload (anomaly detected in the terms of HIDS), that means it is highly probable that a migration has happened. If there is no deviation (no anomaly detected), that means the current test phase has failed and another phase should be performed using different $x_{current}$ or x_{time} values.

How to build normal profiles: we assumed that the attack leader can monitor the host workload so it can create a normal behaviour profile for the host (host based anomaly detection system). The host based anomaly detection system detects any significant change in host behaviour. We first have to build a detection system; different algorithms can be used to build the system some are based on machine learning techniques such as Hidden Markov Model and others are based on statistical learning techniques such as

regression [11]. These algorithms are used to model the workload of the host (create the normal behaviour profile of host workload). To obtain the required data for building the model, the attack leader can gather observations from the host by i.e. calculating host respond time or use side channel to collect data. Then, the host normal behaviour profile can be used to detect any deviations from normal workload pattern. The attack leader can calculate the *SuspicionValue* by comparing the normal profile to the current profile; to compare the two profiles different techniques can be used such as Kullback Leibler distance metric. If *SuspicionValue* is high, alert for anomaly. To decide, if the *SuspicionValue* is high or not, the leader has to specify another threshold to perform this task, *simThreshold*. This threshold is calculated based on the available resources to attack and the required degree of assurance attackers need.

If an anomaly has been detected this might mean a migration has happened; as we said earlier we use *Experiment Replication* (in statistical terms) to increase reliability and reduces the effect of noises generated by errors.

Interactive hypothesis testing by the attacker: We replicate the experiment number of times to obtain statistically significant results. The attack leader has to:

- specify the number of replicates needed, *replicationNum*,
- specify the acceptable level of reliability, *successThreshold*,
- count the number of successful replications, *successCounter*, then
- run an interactive statistical hypothesis testing algorithm to decide whether to accept and distribute the tested values of $x_{current}$ and x_{time} as reliable or not.

The attack leader can form the hypothesis testing in many different ways, for example:

1. The null hypothesis H_0: SuspicionValue $= 0$
   H_1: SuspicionValue $\neq 0$ (one sided hypothesis)
   SuspicionValue variable is equal to *zero* if there is no change in the host behaviour (no migration) and is equal to *one* if there a change in the host behaviour (possible migration).
2. Assume H_0 is true
3. The null hypothesis distribution is computed by the number of permutations which is equal to *replicationNum* (it should be *replicationNum*+1 however because successCounter ≥ 1 so there will be at least one successful experiment)
4. Specify the significant level α.
   The values of the signification level is calculated based on the cost of committing a Type I error (accept and distribute inaccurate $x_{current}$ and x_{time}) and a Type II error (reject an accurate $x_{current}$ and x_{time}). The cost can be calculated depending on different factors i.e. the available resources.
5. The leader calculates the *successThreshold* based on α
6. Compute the t-test statistic
7. Then a decision rule is formed based on the threshold to decide whether to reject the H_0 (accept the tested values as reliable thus distribute them and use them for attacking) or to not reject H_0 (not accept the tested values and go to the next test phase)
8. Collect samples by running the experiment *replicationNum* number of times (experiment replications) and count the number of success and number or fail.
9. Draw a conclusion whether to reject or not reject H_0.

Although the test is replicated number of times to obtain statistically significant results, there is still a possibility for Type I and II errors. Type I error is rejecting a true null hypothesis, however, after the rejection the leader will try higher $x_{current}$ and x_{time} values which will trigger the attack using slightly

higher values than required. Type II error is accepting a false null hypothesis; the attacker might attack using not enough time and strength and this might make the attack fail and also increase the possibility of being caught by security defences in the cloud. Therefore, when selecting the *successThreshold* value, the leader should consider the available security defences and balance the two errors.

Attacking using the revealed parameter: After accepting the values of $x_{current}$ and x_{time}, the CIDoS attack can be formed based on them. The value of T_s (the severity threshold to be broken by the attackers) is $x_{current}$ and the duration of the attack is x_{time}. As described in section 2, in *scenario 1* the leader will broadcast the value of T_s while in *scenario 2* the leader will distribute the units each malicious VM has to cover and these units are calculated by the leader depending on the value of T_s.

Without knowing migration parameters accurately, attackers have to increase the workload to put the host in an over-utilised state; while by using relatively accurate parameters, attackers can trigger migration without over-utilising the host but with making the cloud management algorithms predict that the host will be in an over-utilised state then migrate some of the VMs to avoid future SLA violation (so the current workload will not break the thresholds but the predicted workload will do). This will make the attack harder to detect and also attackers will need less resources to attack with and the parameters can be broadcasted to all CIDoS VMs even in other hosts. This will increase the damage heavily especially that usually migration policies are the same for all cloud hosts.

Also by having a predicted workload that is over the threshold but very close to it by using accurate parameters, the cloud might migrate only one VM from the host (not large number of VMs which is the case if the workload is far over the threshold), migrating one VM increases the lifetime of the attack because other malicious co-resident VMs can increase their workload to cover the loss of one VM and continue with the attack. This is valid especially in *scenario 1* where large number of malicious VMs (far over required) are available. The attackers can keep covering the lost gradually until there are no enough malicious VMs to attack with or there are no other non-malicious VMs in the host; the leader can know this information if, for a series of migrations, only malicious VMs are being migrated. The leader can also know that if the only existed host workload is the collection of malicious VMs workloads. If the leader found out that this host is only occupied by malicious VMs, he/she can either reduce the workload to the minimum to allow for new arrivals or terminate most of the malicious VMs to activate the policy of save energy by migrating all VMs in that host then turn it *off* which is a bigger damage than regular migration especially if the host has to be turned *on* again after short time; it will consume the cloud resources and make the cloud management machine takes decisions based on false reasons. What is more, causing a migration for only one VM will also make the attack harder to detect because migrating large number of VMs in the same time might raise suspicion and lead to further investigations.

6 Analysis and Discussion

If this attack is coordinated between hosts (not only one host) the cloud management node will make a series of false resources consuming decisions which might saturate the network; the cloud management might also start *turning on* more hosts to cope with the fake increased demand. Moreover, if the attacker discovers the VM selection algorithm, he/she can avoid being migrated by for instance intensely using the memory which will make the cost of migrating the attacker VM high thus avoid being selected by the VM selection algorithm for migration. This is just an example, but how to escape migration depends on the used selection algorithm, however, the number of used algorithms is relatively small which ease the attacker task of discovering them. By avoiding migration, the attack will live longer because the group of malicious VMs that form the attack will stay together for long time and constantly attack the same host.

7 Conclusions and Future Work

In this paper we introduced a technique to reverse engineer the cloud migration algorithms, overload detection algorithm and save energy algorithm, to reveal hidden parameters and thresholds. Then these parameters are used to improve the CIDoS attack. We also designed a formal model for migration decision process and then an algorithm has been developed to extract the parameters from the model. We used anomaly based HIDS to measure the success of the attack. The reliability of the extracted values is calculated using an interactive statistical hypothesis testing. These values can be used to attack the host and also can be distributed to other malicious VMs in different hosts.

Based on the theoretical analysis reported in this paper, on-going and future work seeks to validate results experimentally and to refine both the precision and speed of parameter estimation, including by modulating the use of main memory.

References

1. Ristenpart, T., Tromer, E., Shacham, H., 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. CCS '09, New York, NY, USA, ACM (2009) 199–212
2. Zhang, Y., Juels, A., Oprea, A., Reiter, M.: Homealone: Co-residency detection in the cloud via side-channel analysis. In: Security and Privacy (SP), 2011 IEEE Symposium on. (2011) 313–328
3. Alarifi, S., Wolthusen, S.D.: Robust coordination of cloud-internal denial of service attacks. In: Cloud and Green Computing (CGC), 2013 Third International Conference on. (2013) 135–142
4. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput. : Pract. Exper. **24**(13) (September 2012) 1397–1420
5. Bates, A., Mood, B., Pletcher, J., Pruse, H., Valafar, M., Butler, K.: Detecting co-residency with active traffic analysis techniques. In: Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop. CCSW '12, New York, NY, USA, ACM (2012) 1–12
6. Varadarajan, V., Kooburat, T., Farley, B., Ristenpart, T., Swift, M.M.: Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense). In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. CCS '12, New York, NY, USA, ACM (2012) 281–292
7. Voorsluys, W., Broberg, J., Venugopal, S., Buyya, R.: Cost of virtual machine live migration in clouds: A performance evaluation. In: Proceedings of the 1st International Conference on Cloud Computing. CloudCom '09, Berlin, Heidelberg, Springer-Verlag (2009) 254–265
8. Bala, A., Chana, I.: Vm migration approach for autonomic fault tolerance in cloud computing. Int'l Conf. Grid and Cloud Computing and Applications | GCA'13 | (2013)
9. Davis, I.J., Hemmati, H., Holt, R.C., Godfrey, M.W., Neuse, D.M., Mankovskii, S.: Regression-based utilization prediction algorithms: An empirical investigation. In: Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research. CASCON '13, Riverton, NJ, USA, IBM Corp. (2013) 106–120
10. Singh, A., Kinger, S.: Virtual machine migration policies in clouds. Volume 2., International Journal of Science and Research (IJSR) (2013) 364–367
11. Cherkasova, L., Ozonat, K.M., Mi, N., Symons, J., Smirni, E.: Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In: DSN, IEEE Computer Society (2008) 452–461