www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 5, May 2014, Page No. 5752-5758

Optimizing Node Position using Ant Colony Optimization Algorithm (ACO)

A.P.Leela Vinodhini

M.E, A.Justine Jerald M.Tech, Assistant Professor

Abstract: **Modern research has offered confirmation signifying how a malicious user could perform coresidence profiling and public-to-private IP mapping to target and exploit customers which share physical resources. Twp steps are relayed for this attack they are resource placement on the target's physical machine and extraction. In this paper, in part inspired by mussel self-organization, relies on user account and workload clustering to mitigate coresidence profiling. Users with similar preferences and workload characteristics are mapped to the same cluster. To obfuscate the public-to-private IP map, each cluster is managed and accessed by an account proxy. Each proxy uses one public IP**
Address, which is shared by all clustered users when accessing their instances, and maintains the mapping to private IP addresses. In this paper gives the risk assessment for mussel behavior. This paper presented arguments to show how our strategy increases the effort required for an adversary to carry out a directed attack against a target set. This paper proved the experimental result from a risk assessment that suggests a reduced per-individual chance of being randomly victimized given a non directed attack

Keywords: Security, Risk Assessement MUSSEL BEHAVIOR

INTRODUCTION

Equipped with the ability to leverage virtual resources on-demand, cloud computing systems have recently emerged as a viable low-cost alternative to traditional computing platforms. This has sparked widespread interest, adoption, and/or research initiatives from all institutions alike (e.g., academic, industrial, government, etc.), which in turn, has led to myriads of success stories [1], [2] that give credence to its potential and effectiveness. Though promising, this technology suffers from the same fate as any other new development in its infancy stage. It solves some problems while newly introducing unanticipated and not readily understood challenges [3]. At the core of these concerns lies privacy and security [4]–[6]. Recent research [7] has shown that it is possible to identify and target a cloud user, launch malicious virtual machines (VMs) which perform coresidence checks, and possibly extract confidential information once coresidency with the victim has been established. An example such as this exposes the volatility of cloud security.

A cloud computing scenario can be modeled using three different classes of participants: service users, service instances (or just services), and the cloud provider. Every interaction in a cloud computing scenario can be addressed to two entities of these participant classes. In the same way, every attack attempt in the cloud computing scenario can be detailed into a set of interactions within this 3-class model. For instance, between a user and a service instance one has the very same set of attack vectors that exist outside the cloud computing scenario. Hence, talking about cloud computing security means talking about attacks with the cloud provider among the list of participants [14].

Badishi et al. propose an ack-based port-hopping [7] protocol focusing on the communication only between two parties(client-server), modeled as cloud sender and cloud receiver. The cloud receiver sends back an acknowledgment for every data message received from the sender sending data message, and the sender uses these acknowledgments as signals to change the destination port numbers of its messages. Since this protocol is acknowledgement based port hopping, For this time synchronization is not necessary in this cloud services and users

SYSTEM MODEL, THREAT MODEL, AND EXPLOIT DESCRIPTION

Cloud computing systems provide innovative solutions while introducing new avenues for research direction. One aspect of cloud systems which serves in this capacity is hardware virtualization – the ability for multiple customers to share the same physical resources simultaneously. Though providers benefit from resource consolidation, this feature poses new security challenges and possibly serves as a significant system vulnerability. Consider two competing organizations which both lease resources from the same cloud provider. It is foreseeable that one customer's motive could consist of exploiting the shared nature of the cloud to identify, target, and victimize its competitor. Possible attacks could include: monitoring workflow patterns, extracting valuable information, conducting denial of service (DoS), distributed DoS (DDoS), or EDoS (Economic Denial of Service), where the victim's bill causes a shock at the end of the accounting period because they used more instances than planned. Given this, we consider customer VMs, data, and information to be assets.

A. *System and Threat Model*

From a system model perspective, we classify customers based on intent. Malicious users are those with malevolent intent – those who target other users and seek physical machine coresidence for unauthorized surveillance and/or data extraction via certain exploits, e.g., side channel attacks. We consider these type of users to be threats which launch attacks comprised of two steps: VM placement on the machine upon which the target resides and data extraction. Below, we identify 4 types of attackers and list the possible goals for each.

- ***eavesdropping nondirected attacker*** goal is to read data or find out about any target
- ***malicious nondirected attacker*** goal is to cause a DoS on any or all instances
- (***eavesdropping directed attacker*** goal is to get data from a specific competitor's instance or learn about their workload pattern
- ***malicious directed attacker*** goal is to cause one of the following attacks on a particular target: DoS, DDos, or EDoS.

Honest users, on Honest users, on the contrary, are those that use cloud resources for their intended purposes. These users have sincere intent. They abide by the protocols, procedures, and regulations as outlined in the terms of service agreement. We would like to prevent these users from being identified and targeted by malicious users. A peer is simply one that shares the same physical resources – a coresident user. A peer can either be a malicious or honest user. We assume the cloud provider to be trusted and honest – providing the services to its customers as outlined in the service license agreement.

B. Exploit Description

Since the inception of cloud services, the possibility of users being exploited by a rogue peer has always been amajor issue of concern. However, the realization of these fears never quite materialized until researchers began to uncover the extent of cloud user vulnerability. The exploit we consider is described by Ristenpart et al. In [7], they use Amazon's EC2 [17] " as a case study to demonstrate that careful empirical mapping can reveal how to launch VMs in a way that maximizes the likelihood of advantageous placement." To investigate this notion, they assume a placement and extraction attack strategy. They use domain name system (DNS) resolution queries and traditional network tools, e.g., nmap, hping, wget, to determine the external name of an instance and to derive a map which exposes the correlation between the external public IP address and the internal private IP address of an instance. They additionally found that the internal IP addresses are statically assigned to physical machines according to availability zone and instance type. Thus, the map could be used to deduce the availability zone and instance type for any given target – effectively reducing both the search space for finding a target and the number of "probe instances" needed to be deployed before achieving coresidence. A probe instance is simply a malicious VM that performs a coresidence check to determine whether or not a target is a peer. If the target is a peer, it proceeds with data extraction – the next phase of the attack. Otherwise, it terminates. Ristenpart et al. identify 3 different methods which could be used to determine coresidence, and present two strategies an attacker could use to exploit placement in EC2 – brute-forcing placement and placement locality. The brute-forcing placement strategy deploys a large number of instances over time in the same zone and of the same type as that of the instances belonging to a large target set. They conduct an experiment using this strategy and receive a success rate of 8.4%. This means that 8.4% of the probe instances actually achieved coresidence with instances of the target set. The placement locality strategy, on the other hand, assumes a smaller target set, and also presumes that the attacker can launch probe instances soon after a targeted victim's instances are launched. They conduct another experiment, and find that this strategy yields a success rate of 40%. They make the following conclusions concerning Amazon's VM placement algorithm

- N parallel instantiations launched from a single account tend to result in placement on N different machines.
- If a VM which runs on machine A is terminated and another VM is launched immediately thereafter, then that new VM tends to be placed on machine . This may explainwhy the brute-forcing strategy did not fare as well.
- Two VMs launched around the same time, from two different accounts, tend to be assigned to the same machine.
- There is a small inherent bias in assigning new VM instances to machines with light loads.

MUSSEL BEHAVIOR USING K-MEANS ALGORITHM

It is foreseeable that a combinatorial rise in the possible combinations of user preferences could result in large computational overhead with deterministic or complete enumeration algorithms. Thus, the use of heuristic algorithms may prove to be beneficial. We extend the self-organization behavior of mussels to develop an algorithm to address such a problem.

Interactions between organisms, themselves, and the environment in which they live lead to feedback which affects both the organisms and the environment. For mussels, the magnitude of this feedback varies with distance – a phenomenon known as scale-dependent feedback (SDF) [5]. There are two types of SDF: positive and negative. Mussels experience positive SDF over short-range distances with respect to peers. This leads to cooperation between individuals in the vicinity. If there is shortrange density, or a certain number of peers per unit area in its immediate surroundings, an individual mussel tends settle, or maintain its current position. It then secretes byssal threads to attach itself to the shells of peers, rocks, or other various substrates.

On the other hand, mussels experience negative SDF over long-range distances with respect to peers. This leads to competition which restricts survival over long distances. If there is long-term density in its not so immediate surroundings, an individual mussel tends to move to a new location. The interplay between positive and negative SDF ultimately results in patches of optimal sized clusters – large enough to decrease the risk of predation and water stress yet small enough for the groups to withstand the risk of food depletion.

```
Algorithm

Step 1: K points as the initial centroids

Step 2: repeat

Step 3: Form K clusters by assigning all points to the
closest centroid

Step 4: Recomputed the cancroids of each cluster

Step 5: until, the centroid don't change.
```

Modified Model

We now look to modify this model such that groups of individuals within some given population self-organize – where clustering only occurs between peers which belong to the same group or category. This requires thatmembers bemore selective when choosing peers for cluster formations. In the original algorithm, only peers within short-range distance are eligible for local clustering. This condition S_R, is satisfied when peers $D_i < D_1$ are distance away

$S_R = D_i < D_1$ (1)

Here the distance is calculated by using k-means algorithm. Further, peers that are $D_1 \leq D_i < D_2$ distance away tend to viewed as potential competitors prompting an increase in negative SDF. Thus,

$L_R = D_1 \leq D_i < D_2$ (2)

Here, D_2 is 22.5 – the distance used to determine long-range density. In order to increase selectiveness, in this paper modify the conditions for peer eligibility when determining short and long-range density. For short-range density, we now specify that in addition to being $D_i < D_1$ distance away, a peer must belong to the same group, G_S That is to say

$S_{MR} = (D_i < D_1) \cap G_S$ (3)

In a similar fashion, for long-range density, we now maintain that peers have to belong to the same group and must be $D_1 \leq D < D_2$ distance away. To ensure that there is not extensive overlap between heterogeneous clusters, in this paper also specify that peers that belong to different groups, , are at least distance away. We run multiple simulations, each time varying the value for D_3 , and find that $D_3 = 4$ yields the best results. Given this, we say that the modified condition to determine peers eligible for long-range density is:

$L_{MR} = (D_1 \leq D_i < D_2 \cap G_S) \cap (D_i < D_3 \cap G_D)$ (4)

We now explicitly point out the difference between the original and modified conditions for short and long-range density. The original conditions for short and long-range density are shown in (3) and (4). Here, is the total number of mussels in the population. For short-range density, each individual mussel performs a Boolean comparison – where the distances of its peers are compared to .Apeer whose distance is within short-range yields 1. Otherwise, it yields 0. Taking the sum of all the comparisons excluding the reference mussel, and dividing by the short-range area yields the short-range density. A similar procedure is exercised to compute long-range density. However, there are two differences. One, all mussels determine whether the distances of their peers falls between and ; and two, all mussels use the long-range area instead of the short-range area.

$D_S = \frac{1}{\pi D_1^2} \sum_{i=1}^{N} [S_R] - 1$ (5)

$D_L = \frac{1}{\pi D_2^2} \sum_{i=1}^{N} [L_R] - 1$ (6)

The modified conditions are presented in (5) and (6). The procedures to compute the modified short and long-range densities are similar to those used for the original conditions. The differences there are: all mussels take into consideration whether their peers belong to the same group; and for long-term density, mussels additionally consider whether those from a different group are too close.

$D_{MS} = \frac{1}{\pi D_1^2} \left[[\sum_{i=1}^{N} S_{MR}] - 1 \right]$ (7)

$D_{ML} = \frac{1}{\pi D_2^2} \left[[\sum_{i=1}^{N} L_{Mr}] - 1 \right]$ (8)

The linear expression which describes an individual mussel's chance of movement is now:

$P_{MM} = a - bD_{MS} + cD_{ML}$ (9)

To reverse the process, that is, to have the mussels disperse from groups to random individual positions, we simply invert the magnitudes for short and long-range density in (7). This results in the linear expression shown in (8).

$P_{MMR} = a + bD_{MS} - cD_{ML}$ (10)

Thus, the differences for the modified mathematical formulation are: D_{MS} replaces D_S, D_M replaces D_L , and P_{MM} replaces P_M

Now describe the technical analysis of the mussel-inspired self-organization approach towards reducing the risks

.

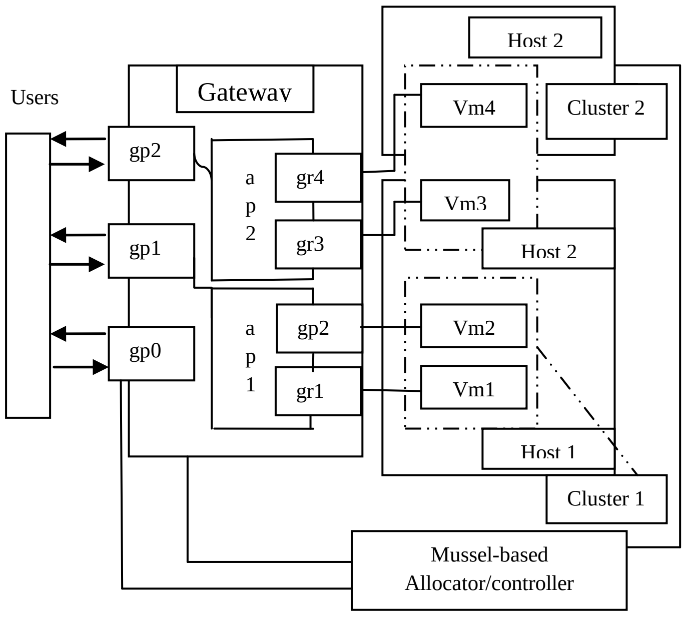


Figure 1: Technical architecture of the account proxies and mussel-based account allocation.

In this figure ap# denotes the account proxy, vm# denotes the virtual machine, gp# denotes the gateway interface with public IP, gr# denotes the interface to private IP, cluster # denotes the logical VM cluster from mussel algorithm and host# denotes the physical host for VM creation. of adversary exploitation as described in [6]

They conclude by stating cloud providers should obfuscate the internal structure of their services and placement policies in order complicate the adversary's attempts. However, obfuscation of topology and placement policy leads to additional computational overhead when doing VM placement, CPU load balancing, traffic shaping and workload migrations. They additionally state that such obfuscation techniques should be demanded only by customers with strong privacy requirements, but this additional differentiation in user classification and infrastructure configuration leads to more complex registration, preference analysis, and configuration options. We suggest defining a single user management and placement solution that comes with low-computation placement and topology obfuscation inherently, without causing a change in the familiar interface exposed to cloud users. Fig. 5 provides an overview of the integrated solution's technical architecture. Here, each logical cluster (cluster#) is managed and accessed by the same account proxy (ap#). An account proxy has one public IP address, which is hence shared by all account owners in a cluster when accessing their instances and the account proxy maintains the mapping to private IP addresses. There is hence no 1-to-1 mapping of public to private IP addresses or dependence on a sequential allocation of private IP addresses. A 1-tomapping of public to private IP addresses is implemented by most modern application-level gateways that include network address translation (NAT) and traversal. The sequence of interactions of a typical user is as follows:

- Subscription of user with the cloud infrastructure via an accessible gateway interface, gp0, with a static public IP address. The user provides a username, password and collection of preferences (duration, CPU, memory), encrypted with the public key of the cloud infrastructure provider.
- The user information is checked against subscription policies and forwarded to the mussel-based allocator/controller, which is responsible for creating/dissolving groups and account proxies, as well as assigning users and VM instances to account proxies, groups, and physical hosts respectively. VMs with similar workload and access preferences are assigned to the same physical host when possible.
- The allocator/controller creates a new account proxy (ap#), if necessary, and assigns the user; or it adds the user to an existing account.
- Asynchronously, the allocator/controller selects a host to create the requested VM instance and starts the VM instance – assigning it a random IP address from a pool of unassigned private addresses.
- The public IP of the newly created VM is mapped to the private IP and returned to the user as a uniform resource identifier (URI) of form /{ip of gp#}/{userid}/{vm_id}.
- The user uses the URI to send requests to the VM including start, stop, modify, or ssh.
- The account proxy translates the URI into a private IP and forwards the requests to the VM.
- Responses from the VM are returned to the user as if the target was the public IP address of the account proxy.

In this paper assume that each user and the cloud provider are able to generate and maintain non compromised public-private key pairs (e.g., RSA [7]) and symmetric keys (e.g., AES [8]) such that the above interactions can be secured using protocols like transport layer security (TLS) [9]. This is among the current best practices from leading cloud providers such as Amazon [10], and is an effective approach for minimizing cloud communication risks such as man-in-the-middle, session high jacking, and replay attacks – as also denoted in [11]–[13]. These types of attacks are hence not the focus of the solution as we are intereste in mitigating the impact coresident placement and data extractions have on an attacker's ability to carry out successful exploits against a given target set.

Figs. 3 – 6 show the set of capabilities and attack path an attacker needs to execute for targeted coresidence. As shown in Fig. 3, do not provide a solution to stopping (1) malicious VMs or scripts from being installed in the cloud infrastructure, as this depends on the types of pre installation scanning mechanisms the provider implements.. in this paper remove the usefulness of public-to-private IP address aims to remove the usefulness of public-to-private IP addr

Step 1: Install VM with DNS lookup scripts

Step 2: Probe web-servers externally to check responsive public IPs

Step 3: Use internal DNS lookup to map public IPs to Private IPs

Step 4: Record unique public to private mappings

Step 5: If more GOTO2, ELSE end

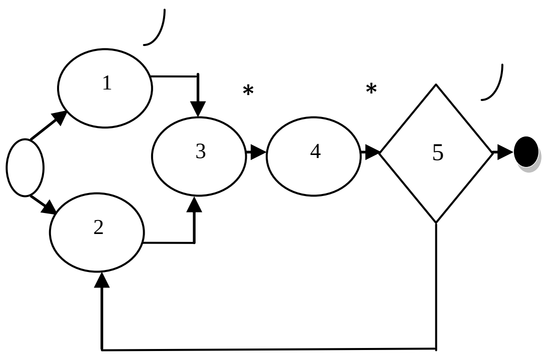


Figure 3: Attack capabilities and path to map public to private IP

* hard/ in distinguishable Easy / distinguishable

? Partial /enumerable

Step 1: Install VM for IP assignment recording

Step 2: Specify VM of type *

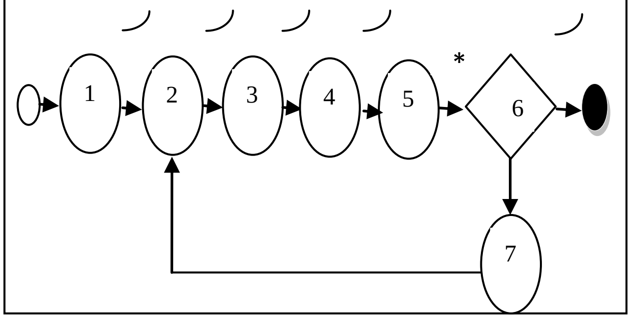
Step 3: Install VM of type * in cloud

Step 4: Record private IP of newly-installed VM

Step 5: Infer IP assignment pattern for type *

Step 6: If try further GOTO 7, ELSE end

Step 7: Very type specification of *, GOTO 2



* hard/ in distinguishable Easy / distinguishable

? Partial /enumerable

Figure 4: A path to determine mapping of VM types to IP ranges and availability zones.

mappings observable by the attacker, which impacts steps (2), (3) and (4) in the attack path, shown in Fig. 3. Mapping 1 public IP address to randomly assigned private IP addresses reduces the specificity of knowledge gained by an attacker with the capability to do internal DNS. The records of mappings will have collisions, which serve to impede targeted coresidence by introducing additional effort and cost for the attacker, in that more brute-force attempts and malicious instances need to be deployed

Fig. 4 shows that the critical step (5) in the attack path is disrupted by our approach, as there is no pattern used for private IP address assignment. The assignment of IP addresses by a dynamic host configuration protocol (DHCP) server will follow a predictable sequence by default; but this can be configured to randomly select from the pool of available IP addresses. There is no need for the cloud administrator to allocate IP addresses per availability zone as groups are dynamically created and assigned responsibility for specific IP addresses.

Step 1: Install VM with tracert script

Step 2: Select private IP of target

Step 3: Do tracert on private IP of target

Step 4: Check firsthop in tracert against last hop of before target

Step 5: If first hop equals last hop infer share Domo

Step 6: If more targets GOTO 2 else end

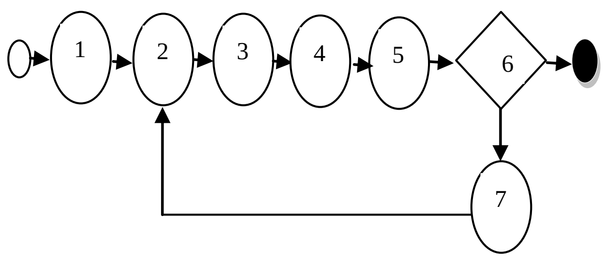


Figure 5: Determine of coresidence using Domo equivalence check

Step 1: Install VM

Step 2: Select private IP within numeric range

Step 3: Do ping on private IP of target

Step 4: Check round trip time (rtt)

Step 5: If response and "short" rtt: infer co-residence;

Step 6: If more targets GOTO 2 else end

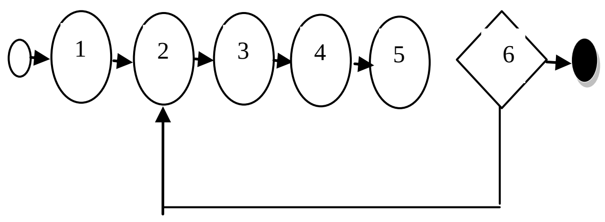


Figure 6: Determination of coresidence using relative round trip time estimate.

Figs. 5 and 6 show that addressing the critical steps (5) in determining coresidence are not addressed explicitly by our solution. It is still possible for the attacker to execute tracert on randomly selected private IP addresses and test for coresidence based on equivalent Dom0 addresses or relatively short round trip times. However, in both cases the attacker is forced to follow a random selection as opposed to following a sequence. Therefore successful coresidence detection does not reveal knowledge about other IP addresses that are numerically close. Figs. 8 and 9 show that addressing the critical steps (5) in determining coresidence are not addressed explicitly by our solution. It is still possible for the attacker to execute tracert on randomly selected private IP addresses and test for coresidence based on equivalent Dom0 addresses or relatively short round trip times. However, in both cases the attacker is forced to follow a random selection as opposed to following a sequence. Therefore successful coresidence detection does not reveal knowledge about other IP addresses that are numerically close.

I. Risk assessment

Up until this point, we have discussed how our solution provides measures to prevent users from being targeted and exploited. However, it is quite possible for users to be random victims of non directed exploits. In this paper perform a risk assessment to determine the likelihood of this event, considering that the impact of non directed exploits is workload and user dependent. With that said, suppose a malicious user decides to randomly target users belonging to any account proxy. Let user A denote a particular user amongst those which could be victimized; and let group B describe all users aside from user A. Below, we list 3 events.

Event A_1: User A is not victimized

Event A_2: Group B users are not victimized

Event B: A successful exploit occurs

Further,

$$\frac{P(A_1)=\alpha}{P(B|A_1)=\beta} \left\| \frac{P(A_2)=1-\alpha}{P(B|A_2)=1-\beta} \right. \qquad (11)$$

In this $P(A_1)$ is the likelihood that user A is not victimized and $P(A_2)$ is the likelihood group B users are not victimized, $P(B|A_1)$ is the likelihood a successful exploit occurs, given that user A is not victimized and $P(B|A_2)$ is the likelihood a successful exploit occurs, given that group B users are not victimized. In this paper assumed that events A_1 and A_2 are mutually exclusive.

Bayes' Theorem is taken in this paper to determine the likelihood user A is NOT victimized, given that a successful exploit occurred.

$$P(A_1|B) = \frac{P(A_1)P(B|A_1)}{P(A_1)P(B|A_1)+P(A_2)P(B|A_2)} \qquad (12)$$

$$= \frac{\alpha\beta}{\alpha\beta+(1-\alpha)(1-\beta)} \qquad (13)$$

Now, suppose that all users each have a VM deployed; each account proxy has M_j members; and there are N total account proxies. Further, consider that each proxy has the same chance of being targeted, and that members assigned to each account have the same chance of being victimized. Then, the chance of user A being victimized is the likelihood that his particular account proxy is targeted, $1/N$ times the likelihood that user A will be randomly targeted $1/M_j$. This yields $1/NM_j$. Thus, the chance that user A will not be victimized is expressed in (10). Notice, here, we use to denote the product N of and M_j. Moreover, as denoted in (11), we say that the likelihood of user A being victimized is the same as the chance that users from group B are not victimized

$$\alpha = 1 - \left[\frac{1}{N}\frac{1}{M_j}\right] = 1 - \frac{1}{\mu} \quad (14)$$

$$1 - \alpha = 1 - \left[1 - \frac{1}{\mu}\right] = \frac{1}{\mu} \quad (15)$$

Substituting (14) and (15) into (13), we receive

$$P(A_1|B) = \frac{\left[1-\frac{1}{\mu}\right]\beta}{\left[1-\frac{1}{\mu}\right]\beta+\left[\frac{1}{\mu}\right](1-\beta)} \quad (16)$$

$$= \frac{\beta\mu-\beta}{\mu\left[\frac{\beta\mu-2\beta+1}{\mu}\right]} \quad (17)$$

$$= \frac{\mu-1}{\beta^{-1}+\mu-2} \quad (18)$$

Also use Bayes' Theorem to determine the likelihood that users from group B are NOT victimized, given that a successful exploit occurred.

$$P(A_2|B) = \frac{P(A_2)P(B|A_2)}{P(A_1)P(B|A_1)+P(A_2)P(B|A_2)} \quad (19)$$

$$= \frac{(1-\alpha)(1-\beta)}{\alpha\beta+(1-\alpha)(1-\beta)} \quad (20)$$

Substituting (14) and (15) into (20), we find

$$P(A_2|B) = \frac{\left[\frac{1}{\mu}\right][1-\beta]}{\left[1-\frac{1}{\mu}\right]\beta+\left[\frac{1}{\mu}\right][1-\beta]} \quad (21)$$

$$= \frac{\frac{1-\beta}{\mu}}{\left[\frac{\mu-1}{\mu}\right]\beta+\frac{1-\beta}{\mu}} \quad (22)$$

$$= \frac{1-\beta}{\beta(\mu-2)+1} \quad (23)$$

Given an exploit, events A_1 and A_2 are equally likely when

$$P(A_1|B) = P(A_2|B) \quad (24)$$

$$\frac{\mu-1}{\beta^{-1}+\mu-2} = \frac{1-\beta}{\beta(\mu-2)+1} \quad (25)$$

Set $c = \mu\beta$ and solve for μ receive

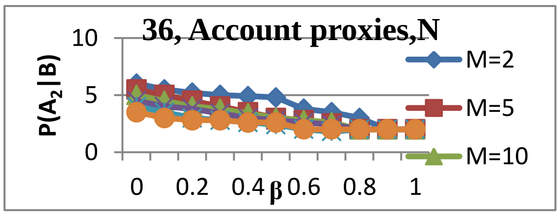
$$(\mu-1)(\in -2\beta+1) = (1-\beta)(\beta^{-1}+\mu-2) \quad (26)$$

$$\mu \in -2c-1 = \beta^{-1}-3 \quad (27)$$

$$\in(\mu-2) = \beta^{-1}-2 \quad (28)$$

$$\mu = 2+\frac{\beta^{-1}-2}{\epsilon} \quad (29)$$

The parameters for (12) (14), and (16) are N, M_j *and* β. . To understand howthe number of members M_j affect and $P(A_1|B)$ and $P(A_2|B)$. we arbitrarily choose N=36 , specify a set of values for parameters M_j and β, and plot the results. The interpretation of the graphs is quite intuitive.



Conclusion

Proposed a solution which relies on mussel-inspired user account, workload clustering, and account proxies to obfuscate the public to private

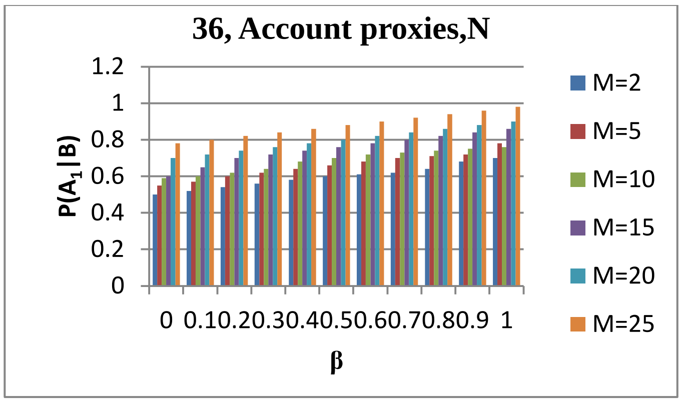


Figure 7: $P(A_1|B)$ The likelihood user A is NOT victimized

The above figure clearly illustrates that it is highly likely a user A will be NOT victimized, given a successful exploit has occurred. The chances increase as the number of members to one's account proxy increases

Figure 11: $P(A_2|B)$ The likelihood that users from group B are NOT victimized.

From the figure observes that the chances of a user from group B not being the victim decrease as increases.

From the figure 12 observes that this only occurs for a select values of M_j and β when $\beta \leq 0.6$ as shown in above figure .When $\beta > 0.6$ the events are equally likely when$\mu = 2$. The values of are negligible in this case.

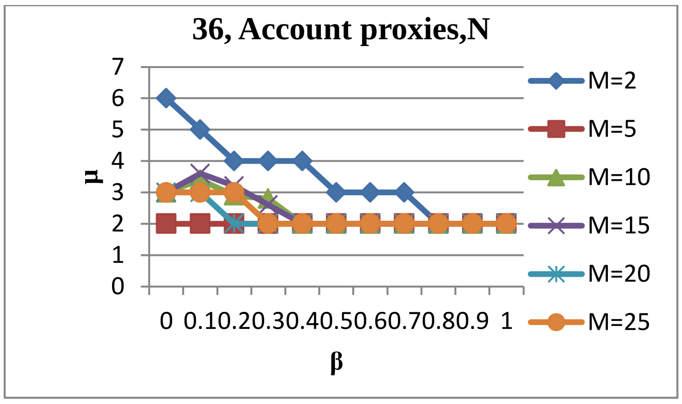


Figure 7: $P(A_1|B) = P(A_2|B)$ The likelihood user A_1 and A_2 is NOT victimized

REFERENCES

[1] M. C. Schatz, "Cloudburst: Highly sensitive read mapping with mapreduce,Bioinform" vol. 25, no. 11, pp. 1363–1369, 2009.

[2] B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg, "Searching for SNPs with cloud computing," Genome

Biol., vol. 10, no. 11, p. R134+, Nov. 2009 [Online]. Available: http://dx.doi.org/10.1186/gb- 2009-10-11-r134

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph,R.Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010 [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[4] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," IEEE Security Privacy, vol. 8, no. 6, pp. 24–31,Nov. 2010 [Online].Available: http://dx.doi.org/10.109/MSP.2010.186

[5] M. Rietkerk and J. Van De Koppel, "Regular pattern formation in real ecosystems," Trends Ecol. Evolut., vol. 23, no. 3, pp. 169–75, 2008 [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/18255188.

[6] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in Proc. 16th ACM Conf. Computer and Commun. Security (CCS '09), New York, NY, USA, Nov. 2009, pp. 199–212 [Online]. Available: http://dx.doi.org/10.1145/1653662.1653687.

[7] B. Kaliski and S. O. T. Memo, Public-Key Cryptography Standards (PKCS) #1: Rsa Cryptography Specifications Version 2.1 2003, rfc 3447.

[8] Specification for the Advanced Encryption Standard (AES) 2001, Federal Information Processing Standards Publication 197 [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[9] T. Dierks and E. Rescorla, The Transport Layer Security (TLS) Protocol 2006, IETF RFC 4346.

[10] Amazon AWS Security Best Practices, A. W. S. LLC, Jan. 2011, White Paper [Online]. Available: http://media.amazonwebservices. com/Whitepaper_Security_Best_Practices_2010.pdf

[11] G. Brunette and R. Mogull, Security Guidance for Critical Areas of Focus in Cloud Computing V2. 1 CSA (Cloud Security Alliance), USA, 2009, vol. 1 [Online]. Available: http://www.cloudsecurityalliance. org/guidance/csaguide.v2

[12] M. Jensen, J. Schwenk, N. Gruschka, and L. Iacono, "On technical security issues in cloud computing," in Proc. EEE Int. Conf. Cloud Comput., 2009 (CLOUD '09), Sep. 2009, pp. 109–116.

[13] J. Wayne and T. Grance, Guidelines on Security and Privacy in Public Cloud Computing, NIST Special Publication, 2011 [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800–144/ SP800–144.pdf

[14] Nils Gruschka and Meiko Jensen, "Attack Surfaces: A Taxonomy for Attacks on Cloud Computing", 3rd International Conference on Cloud Computing, 2010.

[15] G. Badishi, A. Herzberg, and I. Keidar, "Keeping Denial-of-Service Attackers in the Dark," IEEE Trans. Dependable and Secure

[16] Computing, vol. 4, no. 3, pp. 191-204, July-Sept. 2007