Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

If you want to know about a hunter, study his prey: detection of network based attacks on KVM based cloud environments

Nikolaos Pitropakis[1*], Dimitra Anastasopoulou[1], Aggelos Pikrakis[2] and Costas Lambrinoudakis[1]

Abstract

Computational systems are gradually moving towards Cloud Computing Infrastructures, using the several advantages they have to offer and especially the economic advantages in the era of an economic crisis. In addition to this revolution, several security matters emerged and especially the confrontation of malicious insiders. This paper proposes a methodology for detecting the co-residency and network stressing attacks in the kernel layer of a Kvm-based cloud environment, using an implementation of the Smith-Waterman genetic algorithm. The proposed approach has been explored in a test bed environment, producing results that verify its effectiveness.

Keywords: Cloud computing; Security; Co-residency; Network stressing; Malicious insider; KVM; System calls; Smith-waterman

Introduction

Distributed systems have made a huge renovation in Information Technology (IT) infrastructures. Their continuation is the Cloud Computing. Despite a modern trend and a new economic model, Cloud Computing has made its statement turning into the technological model employed by the majority of large companies and organizations for facilitating their everyday needs. It is well known however that every novelty, despite offering a lot of advantages, also brings several disadvantages. The latter usually remains hidden, until a horror story appears. We refer to the security threats that the new technology has raised. They can be classified as: related to the service provider or to the infrastructure or to the host of the Cloud System.

Several of them are well known from conventional IT infrastructures: Distributed Denial of Service [1] came with distributed systems and still draws the attention of security experts, while social engineering attacks [2], malware and Trojan horses [3] are also popular for their impact on modern IT infrastructures. Despite the inherited threats, there are newly generated risks that need confrontation. The most important of them are Loss of governance [4], data interception [3] and replay attacks [3].

Our work focuses on the older and most unpredictable threat that existed before IT systems were born: the human factor. We refer to malicious insiders [4,5] of a Cloud Computing Infrastructure. Their activities can harm the confidentiality, integrity and availability of the data and services of a cloud system. The commonest role that a malicious insider has in a cloud infrastructure is that of the administrator; either the administrator of the host or one of the administrators of the virtual machines (VM). The privileges of an administrator allow several kinds of attacks to be launched. However, our work focuses on the network attacks and especially the stressing of the host network and the co-residency attack [6]. To be specific the stressing of the network is the basic component of DOS and DDOS attacks [7] , where packets are continuously sent to the target in order to stop it from behaving properly and eventually deny its services to others. In the case of co-residency attack [6], we talk about the detection of neighbouring VMs and the retrieval of information about them such as their operating system. The leakage of so important information can seriously harm the cloud infrastructure.

There have been numerous attempts to counter networking stressing attacks [7,8] in their DOS and DDOS

* Correspondence: npitrop@unipi.gr
[1]Department of Digital Systems, University of Piraeus, Piraeus, Greece
Full list of author information is available at the end of the article

form. There are also attempts aiming to handle the activities of a malicious insider through the implementation of several different IDSs, connected through an event gatherer [9]. However, none of these attempts has managed to successfully prevent the actions of malicious insiders.

This paper, presents a novel method for identifying network based attacks in a cloud infrastructure. To this respect a KVM-based [10] system has been employed with its host OS Dom0 having direct access to all I/O functions of the system. . This access is materialized by monitoring the system calls made by the kernel of the Dom0 operating systems. The proposed method has utilized the Smith-Waterman algorithm [11] to prove that by monitoring the system calls, the malicious actions of a potential cloud insider can be detected.

The rest of the paper is organised as follows: Section Related work and network attacks briefly describes the co-residency and the network stressing attacks. Section Detection method provides background information about the Smith-Waterman algorithm and detailed description of the proposed method. Section Test-bed environment and results of the experiments presents the test-bed environment, the applied automation methodology and the results of the tests conducted. Section Discussion contains annotations about the results, while section Conclusion and future work draws the conclusions giving some pointers to future work.

Related work and network attacks

There are several approaches attempting to track, disable or even eliminate the malicious insider threat. Some of them focus on a specific aspect of the cloud such as the employees or the network, while others try to present a global solution. Few of them are able to differentiate themselves from existing solutions, inherited by conventional information systems.

Spring suggests that a firewall at the cloud border that blocks troublesome packets can limit, but cannot eliminate, access to known malicious entities [12]. Alzain, Pardede, Soh and Thom suggest that moving from single cloud to multi-clouds will greatly reduce the malicious insider s threat as the information is spread among the interclouds and can t be retrieved from a single Cloud Infrastructure [13]. Another effort focuses on employing logistic regression models to estimate false positive/negatives in intrusion detection and identification of malicious insiders. Furthermore, it insists on developing new protocols that cope with denial of service and insider attacks and ensure predictable delivery of mission critical data [14].

Magklaras, Furnell and Papadaki [15] suggest an audit engine for logging user actions in relational mode (LUARM) that attempts to solve two fundamental problems of the insider IT misuse domain. Firstly, is the lack of insider misuse case data repositories that could be used by post-case forensic examiners to aid incident investigations and, secondly, how information security researchers can enhance their ability to accurately specify insider threats at system level.

Tripathi and Mishra [16] insist that cloud providers should provide controls to customer, which can detect and prevent malicious insiders threats. They add that malicious insider threats can be mitigated by specifying human resources requirements as part of legal contracts, conducting a comprehensive supplier assessment. This procedure would lead to reporting and determining security breach notification processes.

Fog computing [17] suggests an approach totally different from the others. The access operations of each cloud user are monitored, realising a sort of profiling for each user. This profiling facilitates the detection of abnormal behaviour. When unauthorized access is suspected and then verified, the method uses disinformation attacks by returning large amounts of decoy information to the malicious insiders, keeping this way the privacy of the real users data.

An approach, which is totally different from the latter, is that of Cuong Hoang H. Lee [18], which achieves security in a Xen based hypervisor [19] by trapping hypercalls, as they are fewer than system calls. The hypercalls are checked before their execution and thus malicious ones can be detected. A combination of the two latter methods takes advantage of the system calls, collecting them and classifying them in normal and abnormal through binary weighted cosine metric and k nearest neighbour classifier [20].

Paying special attention to access control mechanisms, Kollam and Sunnyvale [21] present a mechanism that generates immutable security policies for a client, propagates and enforces them at the provider s infrastructure. This is one of the few methods aiming directly at malicious insiders and especially system administrators.

The reference to co-residence or (co-tenancy) implies that multiple independent customers share the same physical infrastructure [22]. This fact results in a scheme where Virtual Machines owned by different customers may be placed in the same physical machine. There are several methods that can achieve the discovery of neighbouring Virtual Machines in a Cloud infrastructure. There are also other methods who wish to counter this specific attack.

Adam Bates [23], claims that co-residency detection is also possible through network flow watermarking. To be specific, this is a type of network converting timing channel, capable of breaking anonymity by tracing the path of the network flow. It can also perform a variety of traffic analysis tasks. However, many drawbacks exist in this method, with the most important one being the introduction of a considerable delay in the network.

Ristenpart [6] presents the co-residency potential attacks on Amazon EC2, one of the largest Cloud Infrastructures. In his methodology he includes network tools such as nmap [24], hping [25] and wget [26], which are utilized in order to create network probes that will acquire the addresses of the potential targets. Additionally, the addresses are used to make a hypothetic map of the cloud network that will be tested in the third step. In the manifestation of the method he explores whether two instances are co-resident or not through a series of checks that depend on:

1. matching Dom0 IP address,
2. small packet round trip times, or
3. numerically close internal IP address

Project Silverline [27] aims to achieve both data and network isolation. Pseudo randomly-allocated IP address are used for each VM, hiding the actual IP addresses provided by the cloud provider. Then, in each Dom0, SilverLine replaces the pseudo IP addresses by the actual addresses before packets leave the machine. Since IP addresses are also discovered through DNS requests, the SilverLine also rewrites DNS responses to appropriate pseudo addresses.

Another approach, namely Homealone [28] allows the verification of the physical isolation of a Virtual Machine through the same tool that can launch co-residency attacks, performed through side channels that usually offer vulnerabilities. L2 memory cache is a popular way to reach the data of another VM. However, in the latter scenario L2 memory is silenced for the period of time needed by the system with upper purpose the residence information not to be acquired by another physical machine. In practice this is rather difficult as the L2 memory in a virtualized environment is never quiet and in most cases there is no physical isolation among the Virtual Machines.

There are numerous attempts to protect Cloud Infrastructures, not only from the co-residency attack but from other network stressing attacks too, by employing Intrusion Detection Systems (IDS). Most of them make use of multiple agents that are installed in different Virtual Machines and collect the data into a centralized point. The disadvantage is that they introduce considerable overhead to the Cloud infrastructure, since they consume significant amount of resources [29-34]. An interesting approach is that of Bakshi and Yogesh [7], who transfer the targeted applications to VMs hosted in another data center when they pick up grossly abnormal spike in inbound traffic.

It can be deduced that the majority of attacks that can be launched by insiders for detecting neighbouring virtual machines or just stressing the network of a Cloud Infrastructure, are based on simple network attacks. In a similar fashion the attacks that have been utilized in this paper for demonstrating the proposed detection method are very simple. Before explaining the attacks it should be stated that in order to launch them the attacker should know the ip address of the virtual machine. In our scenario the attacker is the administrator of a virtual machine with the Kali Linux Operating System [35], the ancestor of Backtrack Operating System [36], which offers to our hypothetic malicious insider a variety of tools.

In the case of the co-residecny attack, the attacker after obtaining the ip address of his virtual machine, is working on finding the Domain Name System (DNS) address. This can be easily retrieved through the command *nslookup* followed by the ip address of the Virtual Machine (VM). This command, executed in the Kali Linux kernel, will return the DNS address. After obtaining the DNS address, the attacker can use the *nmap* command to acquire the ip addresses of all virtual machines (including host) utilising the specific DNS. Specifically the command executed is *nmap sP DNS_Adress/24*. Having the ip addresses of all virtual machines that use the same DNS, the attacker can identify the Operating System of either the Host or of the other Virtual Machines, by executing the command *nmap v O Ip_address*. Through the aforementioned three distinct steps, all co-residents can be identified along with additional information about their operating systems, something that can allow the attacker to launch further attacks harming the Cloud Infrastructure.

Network stress is executed by launching a smurf attack [37] on a specially configured virtual network. In order to perform a smurf attack, the attacker needs the IPv6 address of the victim. The victim can be the Host or any other Virtual Machine on the same network. His IPv6 address can be obtained using two methods. The first one is via the ifconfig command, which can be executed on the Host. The second method is detecting IPv6-active hosts on the same network via the ping6 command [38]. The attacker can easily ping the link-local all-node multicast address ff02::1 from any virtual machine by executing the command "ping6 -I < interface > ff02::1". After obtaining the IPv6 address, the attacker can use the smurf6 tool to perform the attack, executing the command "smurf6 < interface > victim_ipv6_address". Through this method the attacker VM (or the Host) will flood the Virtual Network with spoofed ICMPv6 echo request packets, the source address of which is the IPv6 address of the victim machine and destination address is the link-local all-node multicast address ff02::1. Then the remaining machines on the same network will flood the victim with ICMPv6 echo replies, thus stressing the virtual network even more.

Detection method

Algorithm

The proposed detection scheme has adopted the standard Smith-Waterman algorithm which was originally introduced in the context of molecular sequence analysis [9]. This was possible because the data streams under study consist of symbols drawn from a finite discrete alphabet. A minor modification introduced has to do with two parameters which refer to the number of horizontal and vertical predecessors which are allowed to be scanned in order to determine the accumulated cost at each node of the similarity grid. In other words, these two parameters define the maximum allowable gap length, both horizontally and vertically. This type of minor modification causes a significant improvement in response times and it is also in accordance with the nature of the data that are processed. The values of these two parameters, along with the gap penalty have been the result of extensive experimentation. Next the adopted Smith-Waterman algorithm is presented.

First of all, the pair wise (local) similarity between the individual elements of the two symbol sequences must be defined. To this end, let *A* and *B* be the two symbol sequences and *A(i),i* = *1, M*, *B(j), j* = *1, N*, be the *i*-th symbol of A and *j*-th symbol of B, respectively. The local similarity, *S(i,j)*, between *A(i)* and *B(j)* is then defined as

$$S\ i,j \qquad 1,\quad if\ \ A\ i \qquad B\ j$$

and

$$S\ i,j \qquad -Gp,\quad if\ \ A\ i\ \neq B\ j\ ,$$

where *Gp* is the penalty for dissimilarity (a parameter to our method).

Initialization

Then a similarity grid, *H,* is created with its first row and column being initialized to zeros, i.e.,

$$H\ 0,j \qquad 0,\quad j \qquad 0,\quad N$$

and

$$H\ i,\mathbf{0} \qquad 0,\quad i \qquad 0,\quad M$$

As a result, the dimensions of the similarity grid are *(M* + *1)x(N* + *1),* its rows are indexed 0,..,M and its columns are indexed 0, N.

Iteration

For each node, *(i,j),i* > =*1, j* > =*1,* of the grid, the accumulated similarity cost is computed according to the equation:

$$H\ i,j \qquad \max\left\{\begin{array}{c} 0, \\ H\ i{-}1,j{-}\mathbf{1} \qquad S\ i,j\ , \\ H\ i{-}k,j\ -\ 1 \qquad k*Gp\ ,k \qquad 1,\quad Pv, \\ H\ i,j{-}l\ -\ 1 \qquad l*Gp\ l \qquad 1,\quad P\boldsymbol{h}, \end{array}\right\},$$

$$\boldsymbol{i} \qquad \mathbf{1},\quad ,\boldsymbol{M},\mathbf{j} \qquad \mathbf{1},\quad ,\mathbf{N},$$

where *Pv* and *Ph* are the maximum allowable vertical and horizontal gaps (measured in number of symbols) respectively and *Gp* is the previously introduced dissimilarity penalty (which in this case also serves as a gap penalty). The above equation is repeated for all nodes of the grid, starting from the lowest row (*i* = *1*) and moving from left to right (increasing index *j*). It can be seen that vertical and horizontal transitions (third and fourth branch of the equation) introduce a gap penalty, i.e., reduce the accumulated similarity by an amount which is proportional to the number of nodes that are being skipped (length of the gap).

In addition, if the accumulated similarity, H(i,j), is negative, then it is set to zero (first branch of the equation) and the fictitious node *(0,0)* becomes the predecessor of *(i,j)*. If, on the other hand, the accumulated similarity is positive, the predecessor of *(i,j)* is the node which maximizes *H(i,j)*. The coordinates of the best predecessor of each node are stored in a separate matrix. Concerning the first row and first column of the grid, the predecessor is always the fictitious node (0,0).

Backtracking

After the accumulated cost has been computed for all nodes, the node which corresponds to the maximum detected value is selected and the chain of predecessors is followed until a *(0,0)* node is encountered. This procedure is known as backtracking and the resulting chain of nodes is the best (optimal alignment) path.

In the experiments performed, different values of the parameters *Pv, Ph* and *Gp* have been used and finally the values that provided the most satisfactory performance have been selected.

Proposed method

Fictional character David Rossi, inspired by John E. Douglas, one of the creators of criminal profiling program, once said *If you want to know about a hunter study his prey* [39]. The proposed methodology has been inspired by the above quote. The work of a malicious insider on a KVM-based cloud system, is performed with system calls of the host operating system. In order to investigate the type and sequence of system calls employed, the Linux Audit [40] tool has been used for capturing them.

The procedure that has been followed is the following:

- The system calls engaged during the execution of the *nslookup* command (first step of the co-residency attack), *nmap sP DNS_Adress/24* command (second step of the co-residency attack), *nmap v O Ip_address* (third step of the co-residency attack) and *smurf6* < *interface* > *victim_ipv6_address* (smurf attack) are captured.
- The system calls engaged during the same time period of normal system operation (no attack is being launched)are captured.

- The above log files have been processed with the use of regular expressions and the "*sed*" command [41], leaving only the ID of each system call.
- Finally, the Smith-Waterman algorithm has been employed to compare the logs (every system call ID is being used by the algorithm as a DNA element).

Initially, the similarity between multiple executions of each attack step, at different time periods, was calculated with the use of an automated system that reduced the errors because of the human responsiveness. Then the similarity between an attack step and the respective time period of normal operation was derived. Ideally, this approach would facilitate the identification of specific system call patterns that will form the attack signature.

Test-bed environment and results of the experiments

Setup the environment

In order to launch the attack and monitor the system logs, a minimal Cloud Infrastructure was built using one Dell PowerEdge T410 server with the following configuration: Intel Xeon E5607 as Central Processing Unit, 8 Gigabytes of memory running at 1333 MHz and 300 Gigabytes SAS HDD @10000rpms. The server was running OpenSuse Linux 12.1 [42]. Also the Linux audit [40] tool was installed; this tool has a configuration file that stores a list of rules that specify which type of system calls will be logged. To avoid losing valuable information during our experiments all system calls were captured. Specifically the rule used was *-a entry, always s all* . Finally, two VMs with Kali Linux [35], containing the majority of the tools used for penetration testing and attacks, were set up on the server (see Figure 1).

Automating the attack and system calls auditing procedure

During our effort to automate the attack and the system call auditing procedure, a script was written in Expect [43]. Expect is an extension to the Tcl scripting language and it's used to automate interactions with programs that expose a text terminal interface. This feature can be installed through the expect package. Our script focuses on waiting for expected output with the use of the "expect" command, sending proper input with the use of the "send" command and eventually execute the necessary bash commands with the use of the "system" command. Initially, a directory in which the system calls are going to be saved, was created. Next, the "spawn" command to open the Virsh console [44] and connect to the virtual machine via a configured serial console, was executed. Virsh is a command line interface tool, used for the management of guests and the hypervisor. Then the Linux auditing system was enabled and the attack command was sent to the virtual machine that will be executed. Knowledge about when the attack is finished is acquired by waiting for a specific output of the expect

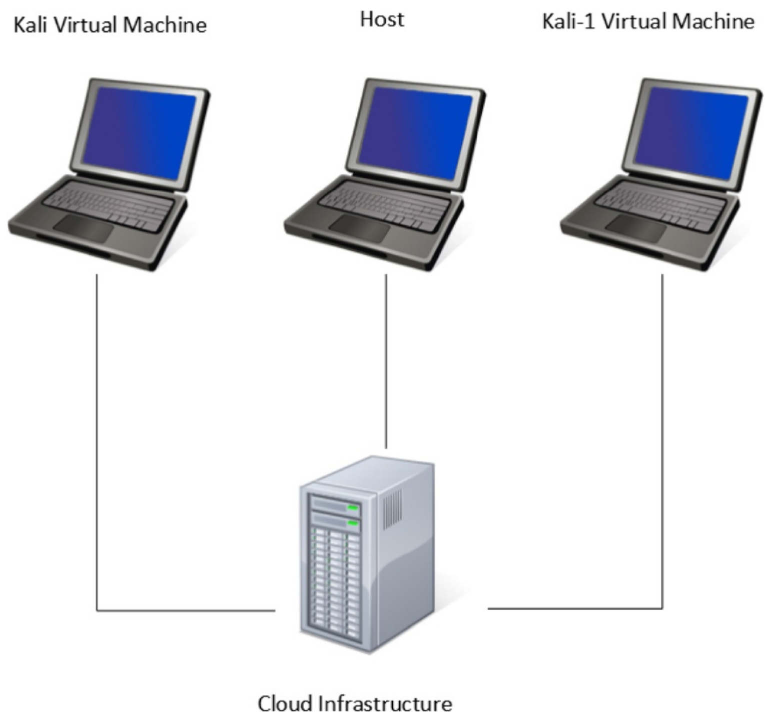

Figure 1 Test-bed environment.

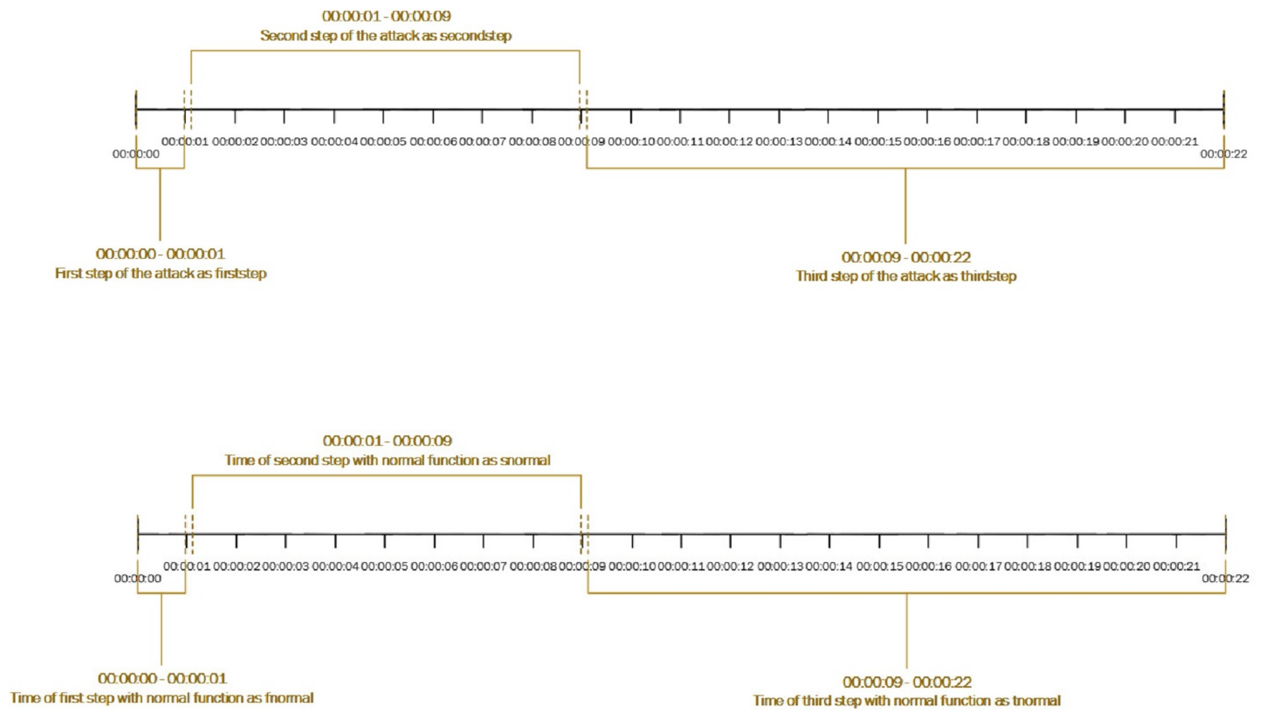

Figure 2 **Time periods for the execution of the three attack steps and the respective time periods that the system was kept in normal state.**

command. Finally, the Linux auditing system is disabled and the the saved system calls are extracted.

Launching the attack

Having setup the environment, each one of the three steps of the co-residency attack (nslookup , nmap and nmap v O Ip_address commands; see section Proposed method) and the step of smurf attack (smurf6 < interface > victim_ipv6_address) were executed six times, each time capturing the system calls engaged.

After every single execution of a command (attack step), the system was left working in normal state for a time period equal to the execution time of the command, capturing again all the system calls engaged during that period. The time periods for the attack and the respective normal state periods are depicted in Figures 2 and 3.

Then by employing the Smith Waterman implementation (see Section Algorithm) in Matlab, using *Gp* equal to 1/3 and 1/5, *Pv* and *Ph* equal to 5 the following log sets were compared between them:

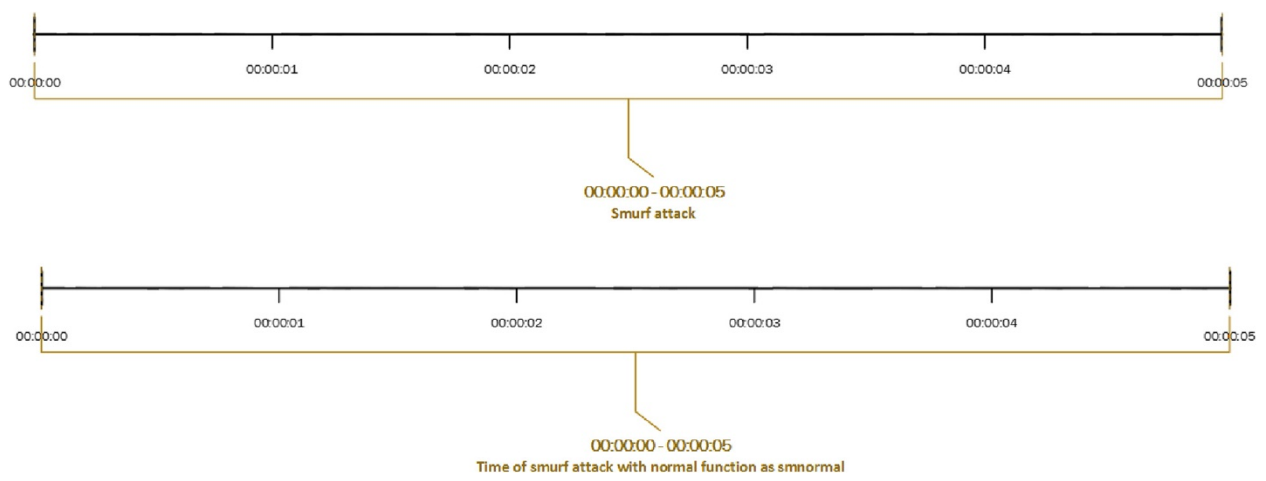

Figure 3 **Time periods for the execution of the smurf attack and the respective time periods that the system was kept in normal state.**

- The six log files (one for each execution round) of the first attack step; *nslookup* command.
- The six log files (one for each execution round) of the second attack step; *nmap sP DNS_address/24* command.
- The six log files (one for each execution round) of the third attack step; *nmap v O Ip_address* command.
- The six log files (one for each execution round) of the *smurf* attack step.
- The twenty four log files of the attack (six log files for all executions of each attack step and smurf attack) with the respective log files for normal system operation.

As demonstrated in the next section, the results met our initial hypothesis. Greater similarity was found between the log files corresponding to the attack steps rather than between the attack logs and the logs of a normal system state.

Results

The results of the log files comparison are presented in the following Tables 1, 2, 3, 4, 5, 6, 7, 8 and 9. As illustrated in Figure 2 and Figure 3, the logs of the first attack step are referred as *firststep*, the logs of the second attack step as *secondstep*, the logs of the third one as *thirdstep* and the logs of the smurf attack as *smurfstep*. Furthermore, the logs corresponding to normal system operation for a time period equal to that of the first attack step are referred as *fnormal,* of the second attack step are referred as *snormal,* of the third attack step are referred as *tnormal* and of the smurf attack as *smnormal*. The estimated similarity numbers that appear in the Gp columns represent the longest subseries of system calls that ware found similar using the Smith Waterman algorithm. It is expected from the

Table 1 Comparison of the six log files (one for each execution round) of the first attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
firststep 1-2	1697.000000	1783.800000
firststep 2-3	2065.000000	2160.600000
firststep 3-4	2116.333333	2212.600000
firststep 4-5	1825.000000	1939.400000
firststep 5-6	1805.333333	1898.600000

Table 2 Comparison of the six log files (one for each execution round) of the first attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
firststep1 fnormal1	571.333333	630.800000
firststep2 fnormal2	1180.666667	1261.400000
firststep3 fnormal3	1162.666667	1227.800000
firststep4 fnormal4	1107.666667	1189.000000
firststep5 fnormal5	1198.000000	1261.200000
firststep6 fnormal6	144.000000	247.000000

Table 3 Comparison of the six log files (one for each execution round) of the second attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
secondstep 1-2	2419.333333	3103.000000
secondstep 2-3	1870.666667	2662.200000
secondstep 3-4	1907.666667	2816.600000
secondstep 4-5	2477.333333	3276.600000
secondstep 5-6	1668.000000	2351.200000

Table 4 Comparison of the six log files (one for each execution round) of the second attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
secondstep1 snormal1	171.333333	174.400000
secondstep2 snormal2	452.333333	889.200000
secondstep3 snormal3	1004.666667	1343.800000
secondstep4 snormal4	562.000000	977.600000
secondstep5 snormal5	787.000000	1123.400000
secondstep6 snormal6	595.000000	1051.800000

Table 5 Comparison of the six log files (one for each execution round) of the third attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
thirdstep 1-2	2024.000000	2776.000000
thirdstep 2-3	2739.666667	3691.000000
thirdstep 3-4	2486.666667	3447.000000
thirdstep 4-5	3226.000000	4222.800000
thirdstep 5-6	3129.333333	4140.600000

Table 6 Comparison of the six log files (one for each execution round) of the third attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
thirdstep1 tnormal1	536.666667	559.200000
thirdstep2 tnormal2	573.666667	1042.400000
thirdstep3 tnormal3	688.666667	1269.000000
thirdstep4 tnormal4	478.666667	970.600000
thirdstep5 tnormal5	878.000000	1323.400000
thirdstep6 tnormal6	562.333333	973.200000

Table 7 Comparison of the six log files (one for each execution round) of the smurf attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
smurfstep 1-2	3155.333333	3277.000000
smurfstep 2-3	2758.333333	2891.400000
smurfstep 3-4	3093.333333	3179.800000
smurfstep 4-5	3230.666667	3304.800000
smurfstep 5-6	2712.666667	2838.400000

Table 8 Comparison of the six log files (one for each execution round) of the smurf attack step for Gp equal to 1/3 and 1/5

Log file comparison	*Gp* = 1/3	*Gp* = 1/5
smurfstep1 smnormal1	217.000000	443.600000
smurfstep2 smnormal2	176.666667	403.400000
smurfstep3 smnormal3	641.333333	791.600000
smurfstep4 smnormal4	695.666667	922.400000
smurfstep5 smnormal5	106.000000	265.000000
smurfstep6 smnormal6	738.333333	1052.800000

Table 9 Comparison of the two log files for each attack step with normal execution with a large amount of network operations for Gp equal to 1/3

Log file comparison	*Gp* = 1/3
firststep1 fnormal1	422.000000
firststep2 fnormal2	449.000000
secondstep1 snormal1	529.666667
secondstep2 snormal2	556.333333
thirdstep1 snormal1	218.666667
thirdstep2 snormal2	259.666667
smurfstep1-smnormal1	126.333333
smurfstep2-smnormal2	211.666667

training procedure that the similarity values will be larger when comparing the logs of the attack steps, and smaller when comparing the logs of an attack step and the respective log of normal system operation; i.e. it is expected that for the same Gp the firstep 1 2 will have larger similarity from the similarity of firstep1-fnormal1. This assumption is strengthened with the results of our last Table 9 where we compare the logs of the execution of each step of the attack with the logs of a system that performs a large amount of network operations that greatly increases the number of system calls. All results are visualized in Figure 4.

Discussion

Recalling our main objective, that was to identify the existence of an attack through the sequences of the system calls. The results, which were presented in the previous section, have indeed verified that approach, since the comparison of the system calls triggered during the attack steps exhibits a much larger similarity than that produced when comparing the logs from some attack step and the respective logs for normal system operation. This assumption came true for all three steps of the co-residence attack and the smurf attack.

It would be a common query whether the results are accurate or not, and how can we verify their correctness. This question can be easily answered through the error parameter, Gp, which was used. To be specific, Gp is a variable that offers flexibility to the algorithm and defines how tolerant the algorithm will be during the comparison of the data sets. If we use the error value of 1/3, we have a less tolerant algorithm than when we use the value 1/5. This assumption leads to greater similarity figures being produced with a Gp of 1/5 than with a Gp

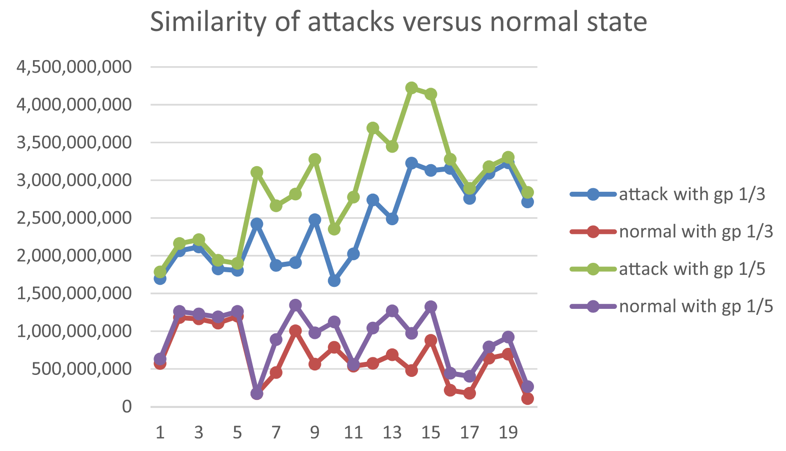

Figure 4 Graph depicting similarity between attacks and between attacks and normal system state for gp 1/3 and 1/5 respectively. Lower gp offers greater similarity.

of 1/3. Of course this is proved with our results, which were presented in the previous section.

In addition to that, we have to pay attention to the fact that the more tolerant the algorithm is, the better the similarity that we get among the logs of the attack steps. However, this is not the case for the comparison of logs produced during an attack step and the respective normal operation; specifically, even though the similarity is better for bigger values of Gp, the scaling is not the same.

Another important issue that should be considered is the workload of the system. During our experimentations we used three Virtual Machines and none of them had any permanent jobs other than those corresponding to the attack steps. In a real time environment, which has extra load on the virtual machines, the number of system calls would be much larger, with results on the time required for processing the log files (as described earlier in the paper). Furthermore, the tracking of the attack in this workload would be more difficult as the algorithm compares identities without being able to recognize whether or not a specific element is useful or not. Nevertheless, an initial set of experiments performed with increased workload indicate that the accuracy and effectiveness of the proposed detection method remains unaltered.

Conclusion and future work

In this paper a practical method for detecting malicious insider attacks from the system calls of the Host Operating System of a KVM based Cloud Infrastructure has been proposed. The approach has been evaluated by comparing the list of system calls produced during the different steps of the attack, not only with other executions of the same attack steps, but also with the normal system state during the same time the attack took place. The results have confirmed the initial assumption that the system calls can be utilized for the detection of an insider attack.

The focus of our current research work is the construction of system call patterns that will be used as *'attack signatures*. The latter will help us build an IDS mechanism, which will be used for the generation of alerts and the prevention of many malicious actions.

Competing interests

The authors declare that they have no competing interests.

Authors contributions

AP was the one who proposed the utilization of the Smith-Waterman algorithm, worked on its configuration and the specific implementation, while he wrote the section about the Smith-Waterman algorithm. DA was responsible for setting up the smurf attack and for conducting the experiments together with NP. She wrote the appropriate sections about the smurf attack and KVM hypervisor. NP was responsible for all technical issues and for setting up the test bed environment and the system calls recovery method. He also wrote the remaining sections of the paper. CL supervised the whole effort providing advice and guidelines on scientific issues, on the experimental methods adopted and on the writing process. All authors read and approved the final manuscript.

Acknowledgements

We sincerely thank all the researchers of the Systems Security Laboratory at University of Piraeus for the inspiration they provide and their useful comments throughout our research.

Author details

[1]Department of Digital Systems, University of Piraeus, Piraeus, Greece.
[2]Department of Informatics, University of Piraeus, Piraeus, Greece.

Received: 9 August 2014 Accepted: 21 November 2014
Published online: 05 December 2014

References

1. Douligeris C, Mitrokotsa A (2004) Ddos Attacks And Defense Mechanisms: Classification And State-Of-The-Art. In: Computer Networks., pp 643 666
2. Orgill GL, Romney GW, Bailey MG, Orgill PM (2004) The Urgency for Effective User Privacy-education to Counter Social Engineering Attacks on Secure Computer Systems. Proceedings of the Conference on Information Technology Education, CITC5
3. Krutz RL, Vines RD (2010) Cloud Security: A Comprehensive Guide To Secure Cloud Computing. Wiley Publishing Inc., Indianapolis
4. Enisa, Cloud Computing Benefits, risks and recommendations for information security (2009)
5. Kandias M., Virvilis N., Gritzalis D., "The Insider Threat in Cloud Computing", in Proc. of the 6th International Conference on Critical Infrastructure Security (CRITIS-2011), Wolthusen S., et al. (Eds.), pp. 95 106, Springer, Switzerland, September 2011.
6. Ristenpart T, Tromer E, Shacham H, Savage S (2009) Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds . In ACM CCS, Chicago
7. Bakshi, Aman, and B. Yogesh. "Securing cloud from ddos attacks using intrusion detection system in virtual machine." Communication Software and Networks, 2010. ICCSN'10. Second International Conference on. IEEE, 2010
8. Liu, Huan. "A new form of DOS attack in a cloud and its avoidance mechanism." Proceedings of the 2010 ACM workshop on Cloud computing security workshop. ACM, 2010
9. Roschke S, Cheng F, Meinel C (2010) An Advanced IDS Management Architecture . J Inform Assur Security 5:246 255
10. KVM Hypervisor. http://www.linux-kvm.org/page/Main_Page
11. Smith TF, Waterman MS (1981) Identification of common molecular subsequences . J Mol Biol 147.1:195 197
12. Spring J (2011) "Monitoring Cloud Computing By Layer, Part 1.". Security & Privacy, IEEE 9.2, pp 66 68
13. AlZain MA, Pardede E, Soh B, Thom JA Cloud computing security: from single to multi-clouds. System Science (HICSS), 2012 45th Hawaii International Conference on. IEEE, 2012.
14. Sandhu R, Boppana R, Krishnan R, Reich J, Wolff T, Zachry J (2010) "Towards A Discipline Of Mission-Aware Cloud Computing."Proceedings Of The 2010 ACM Workshop On Cloud Computing Security Workshop. ACM, Chicago
15. Magklaras G, Furnell S, Papadaki M (2011) LUARM: An audit engine for insider misuse detection. Int J Digital Crime Forensics 3(3):37 49
16. Tripathi, Alok, and Abhinav Mishra. "Cloud computing security considerations."Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on. IEEE, 2011.
17. Stolfo, Salvatore J., Malek Ben Salem, and Angelos D. Keromytis. "Fog computing: Mitigating insider data theft attacks in the cloud." Security and Privacy Workshops (SPW), 2012 IEEE Symposium on. IEEE, 2012.
18. Hoang C (2009) Protecting Xen hypercalls , MSC thesis. University of British Columbia, Canada
19. XEN, http://www.xenproject.org/developers/teams/hypervisor.html
20. Rawat S, Gulati VP, Pujari AK, Vemuri VR (2006) Intrusion detection using text processing techniques with a binary-weighted cosine metric. J Inform Assur Security 1(1):43 50
21. Sundararajan S, Narayanan H, Pavithran V, Vorungati K, Achuthan K (2011) Preventing Insider attacks in the Cloud. In: Advances in Computing and Communications. Springer, Berlin Heidelberg, pp 488 500

22. Xiao Z., Xiao Y., Security and Privacy in Cloud Computing , Communications Surveys & Tutorials, IEEE, vol. PP no.99, pp.1-17, 2012
23. Bates A, Mood B, Pletcher J, Pruse H, Valafar M, Butler K (2012) Detecting co-residency with active traffic analysis techniques. In: Proceedings of the 2012 ACM Workshop on Cloud computing security workshop. ACM, NC, USA, pp 1 12
24. Nmap, http://nmap.org/
25. Hping, http://sectools.org/tool/hping/
26. Wget, http://www.gnu.org/software/wget/
27. Mundada Y, Ramachndran A, Feamster N (2011) SilverLine: Data and network isolation for cloud services , In Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)
28. Zhang Y, Juels A, Oprea A, Reiter A (2011) HomeAlone: Co-Residency Detection in the Cloud via Side-Channel Analysis . Security and Privacy IEEE Symposium, Berkeley, CA
29. Mazzariello C, Bifulco R, Canonico R (2010) integrating a Network IDS into an Open Source Cloud Computing Environment , Sixth International Conference on Information Assurance and Security
30. Schulter A, Vieira K, Westphal C, Westaphal C, Abderrrahim S (2008) Intrusion Detection For Computational Grids . In: Proc. 2nd Int l Conf. New Technologies Mobility, and Security. IEEE Press, Tangier, Morocco
31. Cheng F, Roschke S, Meinel C (2009) Implementing IDS Management on Lock-Keeper , Proceedings of 5th Information Security Practice and Experience Conference (ISPEC 09). Springer LNCS 5451:360 371
32. Cheng F, Roschke S, Meinel C (2010) An Advanced IDS Management Architecture , Journal of Information Assurance and Security, Dynamic Publishers Inc., vol. 51, Atlanta, GA 30362, USA. ISSN 1554 1010:246 255
33. Cheng F, Roschke S, Meinel C (2009) Intrusion Detection in the Cloud , Eighth IEEE International Conference on Dependable. Autonomic and Secure Computing, China
34. Bharadwaja S., Sun W., Niamat M., Shen F., Collabra: Axen Hypervisor based Collaborative Intrusion Detection System , Proceedings of the 8th International Conference on Information Technology: New Generations (ITNG 11), pp. 695 700, Las Vegas, Nev, USA, 2011
35. Kali Linux. http://www.kali.org/
36. Backtrack Linux. http://www.backtrack-linux.org/
37. Smurf Attack, http://www.ciscopress.com/articles/article.asp?p=1312796
38. IPv6 Ping, http://www.tldp.org/HOWTO/Linux%2BIPv6-HOWTO/x811.html
39. John E. Douglas, http://en.wikipedia.org/wiki/John_E._Douglas
40. Linux Audit, http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.audit.comp.html
41. Sed command, http://linux.die.net/man/1/sed
42. OpenSuse, http://www.opensuse.org/
43. TCL Scripting, http://www.tcl.tk/man/expect5.31/expect.1.html
44. Virsh, https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Administration_Guide/chapVirtualization_Administration_Guide-Managing_guests_with_virsh.html

doi:10.1186/s13677-014-0020-6
Cite this article as: Pitropakis *et al.*: **If you want to know about a hunter, study his prey: detection of network based attacks on KVM based cloud environments.** *Journal of Cloud Computing: Advances, Systems and Applications* 2014 **3**:20.