Detecting Malicious Insider Threat in Cloud Computing Environments

Nikolaos Pitropakis

University of Piraeus

Systems Security Laboratory
Department of Digital Systems
University of Piraeus

Ph.D. Thesis

Piraeus, 2015

Abstract

Driven by the lack of focus on attacks, launched by malicious users, against modern Intrusion Detection Systems (IDSs) for Cloud Infrastructures, a real-time cloud observation mechanism is being proposed along with an augmented authenticator. The authenticator enhances the protection level of the data involved in cloud-based services, while the observation mechanism forms a novel detection method of malicious acts by Cloud insiders and incorporates a new implementation of the Smith Waterman algorithm based on CUDA technology. The proposed mechanisms have been evaluated in terms of the overhead that they introduce, justifying that proper execution, without exhausting the cloud infrastructure's computational resources, is possible.

Field of Science: Cloud Computing Security

Key Words: Cloud Computing, Malicious Insider, System Calls, Intrusion Detection System, Authentication, Steganography, Smith Waterman, Hypervisor, XEN, KVM

Περίληψη

Η συγκεκριμένη διδακτορική διατριβή πραγματεύεται την αντιμετώπιση των επιθέσεων σε νεφοϋπολογιστικά συστήματα, οι οποίες προέρχονται από χρήστες με αυξημένα προνόμια. Προκειμένου να απαλειφθούν οι εν λόγω κίνδυνοι υλοποιήθηκε κατά πρώτον ένα σύστημα ενισχυμένης αυθεντικοποίησης, το οποίο προστατεύει τα δεδομένα των νεφοϋπολογιστικών συστημάτων, αφού αυξάνει σημαντικά την ασφάλειά τους. Κατά δεύτερον, δημιουργήθηκε ένα πρότυπο σύστημα ασφάλειας το οποίο κάνει χρήση της υλοποίησης του αλγορίθμου Smith Waterman σε τεχνολογία CUDA. Και οι δύο προσεγγίσεις αξιολογήθηκαν όσον αφορά τον πρόσθετο φόρτο και παρατηρήθηκε ότι λειτουργούν άρτια, ενισχύοντας την ασφάλειας ενός νεφοϋπολογιστικού συστήματος, χωρίς βέβαια να σπαταλούν υπολογιστικούς πόρους.

Θεματική Περιοχή: Ασφάλεια Νεφοϋπολογιστικών Συστημάτων

Λέξεις Κλειδιά: Νεφοϋπολογιστικά Συστήματα, Εκ των έσω απειλή, Κλήσεις συστήματος, Σύστημα Ανίχνευσης Παρεισφύσεων, Αυθεντικοποίηση, Στεγανογραφία, Smith Waterman, Επόπτης (Hypervisor), XEN, KVM

To all the people

who try every day for the impossible

Security is mostly a superstition. It does not exist in nature, nor do the children of men as a whole experience it.

“Hellen Keller”

Acknowledgements

At this moment, I would like to thank the people who contributed greatly to the success of this research. Firstly, I would like to thank my supervisor Costas Lambrinoudakis for his guidance, as well as his scientific and moral support during my research. His valuable advice has not been only inspirational but also determinant in this research.

Additionally, I would like to thank Lecturer Dimitris Geneiatakis for his invaluable contribution and support to the completion of this work. Appreciation also goes to the members of my advisory committee, Professors Sokratis Katsikas, Dimitris Gritzalis and Christos Xenakis for their directions and significant advice for keeping me up with the Ph.D.'s objectives.

My special thanks go to my friends and colleagues Dr. Nikos Vrakas, Eleni Darra and Nikolaos Yfantopoulos for their care and moral support and also for their accurate comments and advice. I am indebted to my colleagues Vasilis Stavrou, Miltiadis Kandias and Nikos Virvilis for providing a pleasant and fun environment, full of interesting discussions. I wish them to complete their research with success.

This Ph.D. would not have been possible without the invaluable support of my family. Their love and encouragement has given me strength and inspiration throughout my research.

(Signature)

....

PITROPAKIS NIKOLAOS

Ph.D. Thesis, University of Piraeus, Department of Digital Systems.

Contents

List of figures

List of tables

Nomenclature

Acronyms

ALU	Arithmetic Logic Unit
CUDA	Compute Unified Device Architecture
CROW	Cloud Realtime Observation Wards
FPU	Floating Point Unit
GPU	Graphics Processing Unit
IDS	Intrusion Detection System
OS	Operating System
SM	Streaming Multiprocessor
VM	Virtual Machine

Chapter 1: Introduction

The ongoing financial crisis and the increasing computational and storage needs, have imposed severe changes to modern Information Technology (IT) infrastructures. IT cost reduction is achieved by offloading data and computations to cloud computing. In general, cloud services vary from data storage and processing to software provision, posing requirements for high availability and on-demand commitment-free provision of services. Even though this economic model has found versatile ground attracting a lot of investments, many people and companies are reluctant to use cloud services because of several security and privacy violation threats that have emerged.

The main characteristics of the Cloud Computing model are: (1) Scale: In order to achieve significant savings, the cloud model supports massive concentrations of hardware resources for the provision of the supported services, (2) Architecture: Although customers who share hardware and software resources are typically unrelated, they rely on logical isolation mechanisms to protect their data. Computing, content storage and processing are massively distributed. This tendency towards global distribution and redundancy, means that resources are usually managed in bulk, both physically and logically [8].

Cloud Computing can be classified into four categories which are: (1) Software as a Service (SaaS): this is the model where applications are hosted and delivered online via a web browser offering traditional desktop functionality, (2) Platform as a Service (PaaS): this refers to the model where the cloud provides the software platform for systems (as opposed to just software), (3) Infrastructure as a Service (IaaS): this is the model where a set of virtualized computer resources, such as storage and computing capacity are hosted in the cloud and customers deploy and run their own software stacks to obtain services and finally, (4) Hardware as a Service (HaaS): this refers to the model in which the cloud provides access to dedicated firmware via the Internet [9].

Moreover, cloud systems can be categorized to: (1) Public Clouds that are publicly available and any organization can subscribe to, (2) Private Clouds that are only accessible within a private network and services are built according to cloud computing principles, (3) Hybrid Clouds, in which an organization provides and manages some resources in-house and has others provided externally and (4) Partner Cloud that is offered by a provider to a limited and well-defined number of parties [8].

Existing attempts to classify threats identified in cloud environments are either based on major cloud components (such as the network or the shared memory of the virtual machines) or on the use of various risk assessment tools [8], like CRAMM and Octave [25][26]. Enisa in [8] allows an informed assessment of the security risks and benefits of using cloud computing by providing security guidance for potential and existing users. Grobauer et.al in [17] define indicators based on sound definitions of risk factors and cloud computing. They discuss about cloud computing security but they fail to distinguish general security issues from cloud-specific issues. Finally, the authors in [19] tackle the most common security challenges that cloud computing faces.

The classification method, presented in this thesis in section 2.1, uses three distinct categories: Threats related to the infrastructure; Threats related to the service provider; and Generic Threats. The key objective of the proposed classification is to ease the burdens of cloud administrators on security related issues, by highlighting the major problems that emerge and thus saving them time and money. Thus, the proposed classification of cloud threats extends the work presented in [8] in the aforementioned direction.

1.1 Problem Identification

Compared to traditional IT services, cloud attack surface has been expanded not only because of the shared resources, but also due to the additional attacking points that an adversary may utilise for exploiting a potential vulnerability in the VM, or in the cloud management platform, or in any other component of the cloud infrastructure. As a result the "*Malicious Insider Threat*", as described in Chapter 2, has evolved to one of the greatest security challenges in cloud computing environments.

According to [48], the term "insider", for an information system, applies to anyone with approved access, privilege or knowledge of the information system and its services and missions. On the other hand, a "malicious insider" is someone motivated to adversely impact an organization's mission through a range of actions that compromise information confidentiality, integrity, and/or availability taking advantage of his/her privileges. In a similar way, for cloud computing "insider" is considered to be an entity who:

- Works for the cloud host
- Has privileged access to the cloud resources

- Uses the cloud services

Consequently, cloud insiders are mostly privileged users, who may be motivated to compromise the cloud infrastructure's security. Their actions may result in a temporary break or even in permanent interruption of the provided services, or in the violation of legitimate users' privacy, depending on their privileges. It is stressed that VM related information, such as the structure of the virtual network being set up for the internal communication among the provided VMs, can be only extracted by privileged users and exploited during the later steps of an attack. To this direction, a malicious user may try to map all available virtual machines and also extract other VM related information [49], in order to overcome cloud security or violate users' privacy.

For instance, a malicious user may combine various utilities such as the *nslookup*, and the *ping* commands, or the *nmap* tool, to capture publicly accessible information for a specific domain of VMs. Even though each one of the above actions is legitimate, the extraction of such information can be utilized for future attacks (e.g. exploiting a vulnerability in a specific operating system). Furthermore, these actions may collectively result in launching an attack known as "co-residence" or "co-tenancy" attack [33]. Alternatively, an internal malicious user may try to affect directly the availability of a virtual network by congesting the corresponding public and private interfaces with numerous ping requests. Network stressing can be also launched through smurf attacks [50].

In addition to the above, the fact that cloud infrastructures lack physical isolation can lead to memory leakages among different VMs. For instance, a malicious VM may try to get access to the shared memory (cache or main memory) and retrieve personal information for the users of the co-resident VMs. In this context, Ristenpart et. al., [33] perform cross VM side channel attack on Amazon EC2 and measure the cache activity of other users, while Rochsa and Correia [51] prove that any malicious privileged user can use the memory dumps of a VM to acquire information about its users, such as passwords, social security number and other personal information.

1.2 Goals and Contribution

The major security threats against Cloud Computing environments are briefly described in Chapter 2. The majority of them cannot be prevented by employing existing countermeasures. As a result, soon after identifying them and understanding that a malicious insider is one of the major threats in a cloud computing environment, the aim was to propose specific new methods for addressing this threat and thus contributing to the scientific community of cloud computing security. Specifically the goals were prioritized as follows:

i. Review of existing methods that identify and counter the malicious insider threat either in conventional systems or in cloud computing systems.
ii. Identification of the source of the malicious insider threat.
iii. Formulation of a methodology that would detect and counter the malicious insider threat.
iv. Application of the proposed methodology and evaluation of its performance and accuracy.

The main contribution of this thesis is the design and development of an adjustable Intrusion Detection System (IDS) that can detect or/and prevent every known attack launched by a malicious insider, as well as other types of existing or future attacks (Figure 1). More specifically, the scientific contribution of the can be summarized as follows:

i. Detailed review of all existing threats against cloud computing systems and mechanisms that have been proposed for addressing them.
ii. The creation of an augmented authentication mechanism versatile enough to prevent most infrastructures from potential dangers.
iii. The identification and customization of an algorithm that will be able to generate and compare attack signatures to a series of system calls generated by any cloud VM or infrastructure.
iv. The creation of a framework that will track any procedure generated into a cloud computing environment either in host OS kernel level or in VM kernel level.
v. The implementation of this algorithm and its evaluation as matters accuracy and overhead.
vi. The further improvement of the algorithm as matters its performance.

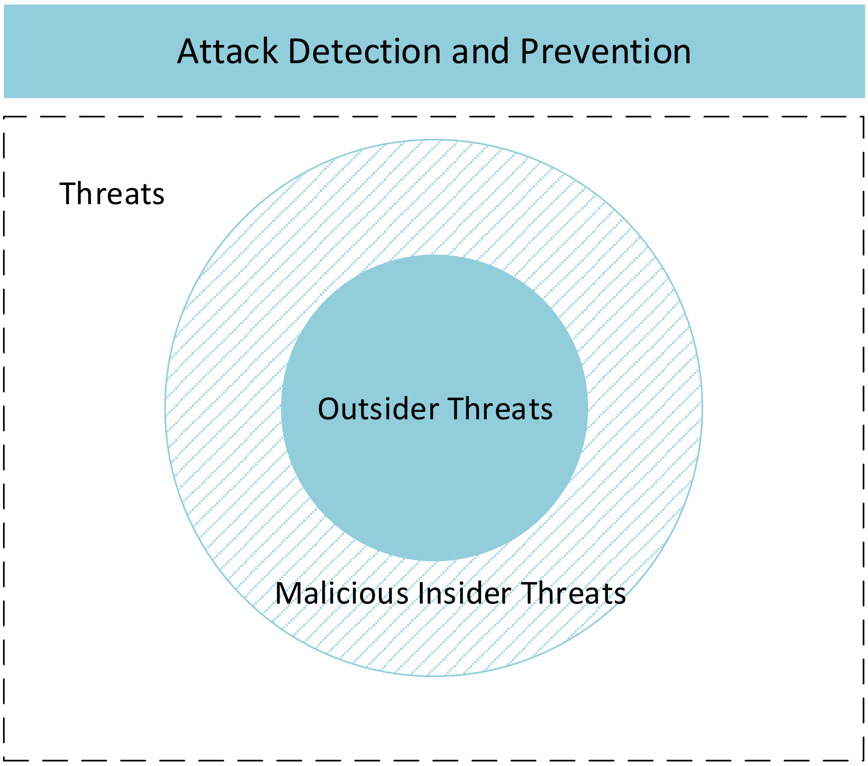

Figure 1: The contribution of the proposed method

To be specific in this Ph.D. thesis after reviewing all existing threats and their known solutions, the malicious insider was identified as one of the major threats in a cloud computing environment. Furthermore, it was concluded that all existing solutions willing to counter this threat would not meet either the global effectiveness or the proper performance, which comes along with the manifestation of attack scenarios generated by malicious insiders. As a result an augmented authentication was created, depending not only on cryptographic but on steganographic methods too. This mechanism is used as an one time authenticator, using mp3 files as stego carrier to transfer the appropriate pieces of information undetectable. In addition to that a lot of algorithms were tested as matters the attack generation and the similarity checks between series of system calls that would offer us the opportunity to identify attacks in a real time environment. This effort resulted in the conclusion that Smith Waterman [52] would be appropriate as for our purpose the behaviour of system calls matches the elements of a DNA sequence, the problem that the algorithm was initially created for. Therefore, a version of this algorithm was implemented in matlab environment in order to generate patterns and test its accuracy. As soon as confidence on the effectiveness of the algorithm was established, a CUDA [53] version of the algorithm was implemented that would greatly reduce the estimated overhead and mitigate into the GPU of a cloud infrastructure instead of its main computational sources.

The following table summarizes the contribution of this Ph.D. thesis:

Table 1 Thesis Contribution

	Short description	**Contribution**
i	The complete review of all existing threats as matters cloud computing systems and their known solutions.	[55]
ii	Creation of an augmented authentication mechanism depending on steganography.	[54], [122]
iii, v	Implementation of Smith Waterman algorithm in XEN and KVM hypervisor based cloud systems.	[56], [125]
iv, v, vi	Creation of the framework and implementation of Smith Waterman in CUDA environment.	[123]

Chapter 2: Threats In Cloud Environments

2.1 Classification of Cloud Computing Threats

In order to facilitate the analysis of the security risks faced in Cloud Computing Systems, it is necessary to classify the identified threats [8] into distinct categories. The following sections present the proposed classification, utilizing three main categories: (1) threats against the cloud infrastructure and hosts (2) threats against the service providers that may affect clients who seek a service in the cloud and (3) generic threats that may affect both the infrastructure and the service providers/clients.

2.1.1. Threats against the Cloud Infrastructure and Hosts

Natural disasters that can harm critical infrastructure: Earthquakes, floods, hurricanes, fire and other natural disasters can be regarded as serious threats that can harm the entire cloud infrastructure. As a result, they can have devastating effects on the system and, in several occasions, on human life. Risk assessment tools have been developed, such as CRAMM and Octave [25][26], which can be utilized for minimizing the consequences of natural disasters [8].

Unauthorized physical access to facilities or equipment: Unauthorized users may try to access the facilities of cloud systems. Such an unauthorized physical access can threaten system’s devices and equipment and can lead to Denial of Service (DoS) for a prolonged period of time. Risk assessment tools like CRAMM and Octave [25], [26], can prevent such problems and must be considered during the initial stage of the Cloud System development [8].

Deficient training/negligence of employees: In many occasions, the employees can pose a serious threat to the cloud system. Deficient training or negligence are heavily concerned with erratic and unpredictable actions of the average employee. Such actions may involve the accidental loss or deletion of the backup data and operational or security logs. A risk management plan in conjunction with the development of a thorough security policy can contribute in avoiding similar events. These measures aid the employees to follow a series of procedures, significantly minimizing in this way the probability of making critical or/and unrecoverable mistakes.

Dumpster diving: Dumpster diving is the risk that each organization or individual takes to discard possible useful information. Sometimes this information that is extracted from the trash can be valuable for anyone who wants to attack the cloud system. Such trashed information may include passwords, phone and credit card numbers. There is no limit in exploiting information found in the trash. Such an information leak can be utilized by malicious users, in order to launch social engineering attacks or to facilitate more threatening scenarios. Each organization must adopt/establish a certain policy regarding the life cycle and the protection of secret information and shall dictate that this policy must be followed by the employees without any exceptions [19].

Password guessing: By employing social engineering or other tools, like Social Engineering Toolkit and TrustedSec [27], [28], malicious users can make educated guesses regarding the passwords used. This kind of attack needs a lot of attempts (brute force attack) and thus it is rather easy to prevent it by setting a limit of invalid password attempts [19].

Unauthorized access to data or information technology systems: This kind of access can be illegally granted by launching social engineering or hacking attacks. In a social engineering attack, the attacker can grant access by simply eliciting the required information, such as users' credentials. Otherwise, privilege escalation techniques may provide the malicious user with the required clearance to access these data. An example of this problem is the SYSRET exploit, where malicious third parties took advantage of AMD's instruction set on Intel platforms [29]. In order to avoid such scenarios, it essential to employ the appropriate and up-to-date security countermeasures and strict access control [8].

Compromisation of operational security logs: Every action, in a large scale Information System, is monitored and stored into detailed security logs. These logs, which are mainly used by system administrators and auditors, provide critical pieces of information that malicious parties can use to launch attacks. Furthermore these logs can expose the identity of the users as they contain sensitive and private data. Protection of security logs must be a matter of high importance, since once compromised they may affect the entire Information System or its users [8], [30], [31].

Network breaks: Each information system and especially a cloud infrastructure provides access to its services through different networks. Every network, depending on its characteristics such as topology and hardware, has known vulnerabilities. Malicious users may use these vulnerabilities in order to either compromise the security of the network or to stop its proper function. These network breaks can pose a serious threat to the provision of cloud services. Thousands of customers may be affected at the same time and the cloud provider (CP) will become untrustworthy to its current and to the potentially new customers [8], [9], [10], [11], [12]. IDS usually reduce such kind of risks. Maybe the solution of Cheng F., Roschke S. and Meinel C., suggesting the installation of IDS mechanisms in Virtual Machines, may reduce the specific threats [41], [42], [43].

Privilege escalation: A malicious user may utilize a virtual machine (VM) in order to attack another VM, escalating his access rights. This can be achieved either by using the hypervisor of the cloud host or the shared memory of the virtual machines. An up-to-date version of hypervisor and countermeasures for privilege escalation are necessary for every cloud provider in order to prevent such acts [8], [32].

Insecure or ineffective data deletion: In a Cloud Computing Infrastructure it may be necessary to delete a recourse. Most operating systems do not fully wipe the data while, in other cases, timely data deletion may also be unavailable. A cloud provider may need to perform several modifications to its architecture, such as changing the location of the server, making a hardware reallocation or even destroying older hardware. During these changes the data might not be transferred or destroyed correctly, due to technical reasons, leaving them exposed. In several occasions, the physical destruction of hard disks may affect clients' data that should not be deleted [35].

Malicious scanning or observation: Malicious parties, in order to acquire information about the Cloud System, use network probing tools such as hping [22], nmap [23] and wget [24], to monitor the network of the cloud infrastructure. They often install malware that collects information for mapping the Cloud System. When a user knows his current position, either in the network or the physical machine of the Cloud Infrastructure, he can use it in order to escalate his privileges and gain access to other Virtual Machines. In such an occasion, the malicious user can illegally retrieve information that would not have been allowed to access [33].

Insecure or obsolete cryptography: Cryptanalysis advances can render any cryptographic mechanism or algorithm insecure. On the other hand, it is a common phenomenon that many Cloud Systems do not accurately implement the encryption/cryptographic protocols or, in the worst case, encryption does not exist at all. Thus, a thorough implementation of contemporary cryptographic techniques must always comprise a high priority since it can protect the system from numerous malicious acts [17].

Economic Denial of Service (EDoS) and exhaustion of resources: Economic denial of service can be recognized in several different scenarios. The most important of them are:

• Identity theft: An attacker may steal the account and the resources of a customer, in order to use them for his own benefit. In such a scenario, the attacker can have access to services for free while the victim's account is charged for these services. Also, the attacker may use the stolen identity and by acting maliciously to threat victim's reputation.

• The Cloud Customer (CC) may have no effective limits on the use of paid resources. As a result he may impose unexpected loads on these resources.

• An attacker may use a public channel so as to use the customers' metered resources. An example is a DDoS attack, when the customer pays per HTTP.

In these scenarios, services may not be available to customers and access control may be compromised. In addition to that, the trustworthiness of the cloud provider is inevitably threatened. EDoS attacks have as their primary target the cloud provider and as a secondary target the clients [8]. Kaliski Jr, B. S., and Pauley, W. suggest risk assessment as a way to avoid EDoS [47].

Isolation malfunction: The infrastructure provider must be able to isolate services from each other. The term 'isolation' refers to performance and security isolation. As a result, the execution of one service must not interfere with another. Typically, isolation can be achieved either by using unique physical machines or isolated network infrastructures. However, when it comes to cloud computing it is rather difficult to have complete isolation, as the Virtual Machines share resources. As a result, in case of isolation malfunction someone who has access to shared resources will be able to retrieve confidential information [8], [34].

Billing fraud: Billing data manipulation and billing evasion is one of the most important vulnerabilities in cloud environments. Cloud services have a metering capability, at an abstraction level appropriate to the service type, such as storage and processing. The metering data is used for service delivery and billing support [17]. An approach has been proposed from Widder, A., Ammon, R. V., Schaeffer, P., & Wolff, that suggests the use of Complex Event Processing Engine [46].

Insufficient logging and monitoring: No standard mechanisms have been proposed to enable logging and monitoring services concerning the cloud resources. This can raise significant concerns. As the existent logging mechanisms usually monitor users and services of an infrastructure, the retrieval of information that affects a single user or service becomes rather difficult. Until efficient monitoring and logging mechanisms are implemented, it is appropriate to consider security controls in Cloud computing [17]. Several tools have been proposed for logging, monitoring and provisioning services such as OpenQRM [36], Cobbler [37], Crowbar [38], Spacewalk [39] and Cloudaudit [45], but no one offers a complete solution.

Cloud Service failure or termination: In addition to DoS attacks that may turn cloud services unavailable for a short period of time, it is also possible to experience service failure or termination. Service failure or termination indicates a permanent inability of the Cloud Infrastructure to provide its services. That may be due to malicious acts of users that have earned elevated privileges into the infrastructure and consequently access to mechanisms that can disturb or disable the functionality of the offered services [8]. The installation of multiple type of IDSs in several Virtual Machines, as Cheng F., Roschke S. and Meinel C. suggest, can significantly reduce that threat [41], [42], [43].

Failure of third party suppliers: Cloud computing providers often outsource several tasks to third party suppliers. That means that the cloud infrastructure's security depends on the security mechanisms utilized by the third party. Suppliers are not always trustworthy. Keeping low security standards or not paying attention to the security policy of the cloud infrastructure may result either to exposing several segments of the infrastructure or even making aspects of the system available to malicious users. Any partner can severely damage the cloud integrity, availability and confidentiality with further impact to its viability. As a result, the cloud provider must be cautious with its partners and preferably have alternate choices in matters of outsourcing [8].

Lock in: Several problems occur when the cloud infrastructure changes ownership and/or policy, while existing users remain as customers. A difficulty of great importance appears when customers cannot easily transfer either their services or their data from one cloud provider to another. In this case we have a variety of 'lock in' problems depending on the architecture of the cloud system. In all three architectures SaaS, PaaS and IaaS the data lock in problem is evident. It is extremely difficult to extract the data of each customer due to technical or legal reasons. Concerning the SaaS architecture, the problem of services lock in can emerge. This means that every cloud provider uses different tools for provisioning and monitoring like openQRM [36], Cobblerd [37], Crowbar [38] and Spacewalk [39]. In PaaS architecture, the problem exists on the API layer since every cloud provider does not use the same virtualization platform. Customers should check whether the new provider uses the same platforms or compatible ones. IaaS lock in varies depending on the infrastructure that is used by each customer. In order to avoid such circumstances, the selection of the appropriate cloud provider must be decided after extensive research, while special attention must be paid to any change in the Cloud Policy [8], [20].

Compliance problems: It is common, that several companies and organizations can migrate into cloud systems for several reasons. Since these companies have been utilizing security certificates and other standards before the migration, compliance problems may emerge. This is mainly because the cloud provider may not utilize the same security standards or policies, or even because the security schemes may not be compatible with each other. It is therefore necessary for the clients to check if the cloud provider can offer services that are compatible with their deployments and can host their services according to their needs. Otherwise, this may lead to denial of service for a prolonged period of time, while the users' disappointment will inevitably threaten operator's reputation [8].

Cloud data provenance, metadata management and jurisdiction: This is an open issue which includes:

- Cloud Process Provenance: Dynamics of control flows and their progression, execution information, code performance tracking, etc.
- Cloud Data Provenance: Dynamics of data and data flows, files' locations, application input/output information, etc.
- Cloud Workflow Provenance: Structure, form, evolution, etc., of the workflow itself.
- System (or Environment) Provenance: System information, O/S, compiler versions, loaded libraries, environment variables, etc.

Considering these issues, it can be concluded that there are a lot of open challenges concerning data provenance. That creates a high degree of uncertainty to the cloud customers, who need to know the provenance of the data they are using. Every cloud provider should form its own provenance system, in order to guarantee the quality of the provided services and protect data confidentiality and users' privacy. In cases that these requirements are threatened, jurisdiction problems may be raised concerning the data and their storage [8], [18].

Infrastructure's modifications: As the technology develops, better and contemporary hardware and software solutions are introduced. Cloud providers may update or upgrade their software/equipment. This can result in extra charge for each customer, even if the latter continues to use the same number of resources through the cloud. Furthermore, the intellectual property of the stored/exchanged data may be at risk, if they are not adequately protected by the appropriate security mechanisms. Cloud providers should care about these matters and put special effort to develop strict rules and security policies concerning the proper use of their systems, in order to avoid legal issues. In addition, the development of risk assessment procedures, through the utilization of the appropriate tools [25], [26], can offer to cloud customers even more secure services [45].

Data processing: In addition to data provenance, another serious concern in cloud computing is data processing. A customer cannot be sure how his data are manipulated by the cloud system and if the processing complies with the legal framework of the country he resides in. Some cloud providers describe the procedures they follow and the certifications they may have. But even if the data are protected against malicious users, it cannot be assured whether the users' stored data have been lawfully obtained or not. That raises another issue: how can these data be evaluated in terms of legality and (at the same time) be protected from disclosure, without violating users' privacy [8].

Administrative and ownership changes: It is possible that a cloud provider may change its administrative personnel (e.g. network or system administrators) or even the whole cloud system may be sold to another company. This can raise many security concerns due to the fact that the security requirements of the former owner/administrator are not always satisfied by the new one. This may have consequences on the data confidentiality, integrity and availability and consequently on the cloud provider's reputation. Thus, it is essential to maintain the previously established security measures for a period of time until the new administration decides to change them. This can prevent malicious entities from taking advantage of such situations.

Denial of service to co-tenants due to misjudgment or misallocation of resources: Since cloud systems provide resource sharing, malicious activities carried out by one tenant may have impact on another. For example, if an IP is banned or blocked to prevent security incidents (e.g. this IP has been used for initiating attacks), some users who have not been involved in malicious acts may still not be able to use the cloud services. Furthermore, a customer may not be able to access a specific service because some other user may have reserved the available resources. This may turn to a major problem since it significantly degrades company's reputation due to the customers' dissatisfaction (they cannot have access to the services they pay for). Therefore, cloud providers shall consider and preserve the customer's right to access the provided services [8].

Subpoena and e-discovery: Every country has a different legal framework on the protection of privacy and processing of personal data. The centralization of storage as well as shared tenancy of physical hardware, put many clients' data at risk since the disclosure of private information does not comprise a punishable action in every country. It is therefore very difficult for each agency of each country to take special care of every cloud system hosted under their jurisdiction. Consequently, customers shall consider the legal framework of the cloud provider in order to avoid privacy related issues.

2.1.2. Threats against the Service Providers

Replay Attacks: During a replay attack, an attacker intercepts and saves the transmitted messages. After spoofing these messages, the attacker re-sends them to the service, impersonating one of the communicating participants. The use of fresh and randomly generated alphanumeric strings (nonces), in the message, can adequately tackle this problem. Other countermeasures may only include a timestamp which indicates the time when the message was sent [19].

Data interception: It consists a group of attacks, which contains:

• Man in the middle: In this type of attack the attacker can impersonate the victim by changing the public key/user association. As a result, the sender encrypts the message with the attacker's public key while the latter can receive, decrypt and modify it. Finally, the attacker encrypts the forged message with the actual victim's public key and forwards it to the latter [19].

• Eavesdropping: Data scavenging, traffic or trend analysis, social engineering, economic or political espionage, sniffing, dumpster diving, keystroke monitoring, and shoulder surfing are all types of eavesdropping. Their purpose is to gain information or to create a foundation for a later attack.

• Side channel attack: The use of side channels in shared hardware enables attackers to infiltrate into sensitive data, across virtual machines of the cloud infrastructure [19].

Browser security: One of the most common risks in cloud systems is the browser security level. Generally, a computer client in cloud is only used for I/O, authentication and authorization. Cloud providers do not develop browsers suitable and safe for this purpose. Consequently, computer clients use a variety of browsers with security features that mainly depend on their software version. Thus, whenever a security breach or exploit emerges on a specific browser it will have impact on the whole Cloud Infrastructure [14].

XML signature element wrapping: It is an attack on protocols using XML signature for authentication or integrity protection. This type of attack applies to web services as well as to cloud systems. It has been only in theory, until 2008, when it was discovered that Amazon's EC2 services were vulnerable to wrapping attacks. The specific vulnerability was a soap architecture exploitation that was used in conjunction with this technique. This group of attacks cannot be easily detected and it still remains a great threat for the Cloud [14], [15], [16].

Injection vulnerabilities: This kind of vulnerabilities are exploited by manipulating service or application inputs. Such a manipulation can force the interpretation and consequently the execution of illegal code. Characteristic examples are the SQL injection, command injection and cross site scripting attacks. Since these attacks are very popular and in most cases easily exploitable, cloud providers shall consider deploying countermeasures and protection schemes even from the first stages of their establishment [17].

Customer's negligence and Cloud Security: Cloud customers fail or neglect to properly secure their cloud environments, enabling malicious users to attack the cloud platform. Customers must realize that they have the responsibility to protect their data and resources. In some cases, cloud customers wrongly assume that the provider is responsible to ensure the security of their data. This kind of risk cannot be addressed through auditing or other techniques. Each company should always keep a high security standard even if their customers do not follow the appropriate procedures [8].

Management interface exposure: Malicious parties can take advantage of internet browsers' and remote access's vulnerabilities in order to have access to several controlling interfaces of the cloud system. This includes customer interfaces that control a number of virtual machines and the operation of the overall cloud system [8]. Frequent browser updates and installation of different kinds of IDS in multiple Virtual Machines, as Cheng F., Roschke S. and Meinel C. Suggest, can reduce this threat [41], [42], [43].

Loss of governance: Frequently the security methods that cloud customers employ significantly deviate from cloud providers' directions. Such a contradiction may lead to loss of governance and control which can have a determinant impact to the cloud system and of course to its data. To this end, every cloud provider shall keep its customers up-to-date with clear and strict security procedures and directions while, in cases of outsourcing, the partners' service must be compatible to these directions/policies [8].

2.1.3. Generic Threats

Social engineering attacks: Classified data and other critical information can be disclosed by users or employees due to inadequate education, negligence or social pressure. An attacker can impersonate (e.g. though a phone call or e-mail) a supervisor, a chief technician or other important entities in order to elicit confidential data, that can be used for attacking the system directly or indirectly. Such information may include passwords, networking topologies, utilized software's and hypervisor's version and others, which can provide the attacker with the appropriate knowledge to launch an attack. That proves that people are the weakest link in such occasions. Social engineering can be mitigated through strict procedures and of course by auditing, which has an essential role in avoiding such attacks [10][45].

Distributed Denial of Service (DDoS): The DDoS attack is an advanced form of DoS attacks. The difference from other attacks is its ability a) to deploy its weapons in a ‘‘distributed’’ way over the Internet and b) to aggregate these forces to create overwhelming traffic. The main goal of a DDoS attack is to cause damage on a victim either for personal reasons, or for material gain or for popularity. DDoS attacks have become more powerful because they have taken advantage of the Cloud architecture which has inherited the distributed systems advantages and disadvantages [21]. However a solution is proposed by Aman B. and Yogesh B. which suggests the implementation of an IDS into a Virtual Machine [40].

Encryption keys exposure or loss: In this type of attack, employees’ negligence or lack of security policies, make the secret keys (file encryption, SSL, customer private keys) vulnerable to malicious users who are neither authorized, nor authenticated to use those [8]. Such negligence can give access to unauthorized users who may launch attacks against the cloud infrastructure or other customers.

Service engine exposure: The service engine is developed and supported by the cloud platform vendors and, in some cases, by the open source community. Specifically, the service engine code is prone to attacks or unexpected failure which means that it can be vulnerable to different malicious operations. For instance, an attacker can manipulate the service engine and gain access to the data contained inside the customer environment [8]. Frequent security updates of the service engine will be able to partially solve the problem. Furthermore, this threat should be considered throughout the risk assessment process [25], [26], [45].

Malware and Trojan horses: Malware and Trojan horses are malicious codes, hidden inside a useful program, that attack the workstation, the server or network or allow unauthorized access to those devices. Trojan horses can be carried via Internet traffic, such as FTP downloads or downloadable applets from websites, or can be distributed through e-mail. Some Trojans are programmed to open specific ports to illegally allow access to attackers or for possible exploitation of systems vulnerabilities [19]. The installation of multiple IDSs on the Virtual Machines connected through an event manager, as Cheng F., Roschke S. and Meinel C. suggested, may be an excellent counter measure [41], [42], [43]. Due to the fact that malware and Trojan horses increase and advance every day, addressing them is not a trivial task.

Malicious Insider of Cloud Provider: The activities of a malicious insider can threaten the confidentiality, integrity and availability of cloud's system data and services. This makes a malicious insider one of the greatest threats of information systems and especially cloud computing, since cloud architectures necessitate certain roles (system administrators and auditors, managed security service providers) which are considered extremely high-risk [8].

2.2 Threat assessment

Table 2 depicts the threats against cloud systems, divided into the three distinct categories presented in the previous section. The two columns in the middle of the table, provide information on whether the specific threat can be addressed either through some technical countermeasures (technical solution) or through some organizational or/and procedural countermeasures (non-technical solution). The proposed solution itself is given in the last column.

Table 2: Classification of Cloud Computing Threats

Solutions				
	Threats	**Technical**	**Non-Technical**	**Known Solutions**
Infrastructure and Host	Natural disasters		●	CRAMM, Octave, CloudAudit
	Unauthorized physical access		●	CRAMM, Octave, CloudAudit
	Deficient training/negligence of employees		○	CRAMM, Octave, CloudAudit
	Dumpster diving		●	CRAMM, Octave, CloudAudit
	Password guessing	●		Limit invalid password attempts
	Unauthorized data access	○		CloudAudit, Multilayer IDS on VMs
	Security logs compromisation	●		CRAMM, Octave, CloudAudit
	Network breaks	○		Multilayer IDS on VMs
	Privilege escalation	○		Access control, Hypervisor update
	Ineffective data deletion	●		CRAMM, Octave, CloudAudit
	Malicious scanning/observation	–		--
	Insecure/obsolete cryptography	○		Contemporary Cryptographic techniques
	EDoS and resources exhaustion	○		Risk assessment as a service
	Isolation malfunction	–		--
	Billing fraud	○		Complex event processing engine

Solutions				
	Threats	**Technical**	**Non-Technical**	**Known Solutions**
	Insufficient logging/monitoring	○		OpenQRM, Cobblerd, Crowbar, Spacewalk, CloudAudit
	Cloud Service failure/termination	○		Multilayer IDS on VMs
	Third party suppliers' failure	○		Flexible Security Policy
	Lock in	○		OpenQRM, Cobblerd, Crowbar, Spacewalk
	Compliance problems	●		Migration compatibility check
	Data provenance and jurisdiction		–	Provenance Policy for CPs
	Infrastructure's modifications	○		CRAMM, Octave, CloudAudit
	Data processing	○	–	Destruction strategies on Service-level Agreements
	Administrative/ownership changes	○	–	Maintenance of established security measures
	DoS to co-tenants	○	–	Customer's access rights preservation
Service Provider	Replay	●		Timestamps, fresh nonces
	Data interception	○		SSL support, jam the emitted channel with noise, Homomorphic encryption
	Browser security	○		Browser updates, WS-security
	XML signature element wrapping	○		Digital certificates
	Injection vulnerabilities	○		Validate length, range, format, and type. Constrain, reject, and sanitize input. Encode output
	Customer's negligence and Cloud Security		○	Effective Security Policy
	Management interface exposure		○	Browser update, IDS on VMs
	Loss of governance		○	Security procedures for handling human factor and outsourcing impact
Generic	Social engineering		○	CloudAudit
	DDoS	○		IDS on VMs
	Encryption key exposure/loss	○	○	Key management techniques, proven platform-provided cryptography
	Service engine exposure	○		Service engine updates, CRAMM, Octave, CloudAudit
	Malware and trojan horses	○		Multilayer IDS on VMs
	Malicious insider of Cloud Provider	–		--
●: Covered		**–: Not Covered**		**○: Partially Covered**

Chapter 3: Literature Review

3.1. Introduction

Since the research work presented in this thesis focuses on several aspects of a cloud computing systems, the literature review has been divided into two distinct categories so as to match the contribution directions. The first subsection addresses the related work on steganography and the authentication schemes employed in cloud computing systems, while the second one discusses the use of IDS in cloud environments and their performance issues.

3.2. Steganography and Cloud Computing Authentication Schemes

Several methods of utilizing steganography have been proposed in the past. The methods that matter most this work are the ones focussing on digital files. When we refer to digital files what comes to our mind is text, image, sound and video files. However, text files are not a common choice due to their structure. More specifically, hiding a message into a text file [57] is easily achieved but it suffers from a significant and visible overhead to the original file. This fact causes suspicions and increase the chance someone to retrieve the hidden message.

The most common method employed in steganography for sound, image and video files is the LSB [58] (Least Significant Bit) method. The LSB has several implementations in various different steganographic algorithms. One of the methods used in images is the spatial domain method [59] where the steganographer modifies the secret data and the cover medium in the spatial domain, an action involving encoding at the level of the least significant bits. However, this method has a large impact on the quality of the file. Another method that has been proposed is to use the LSB into the frequency domain [57] of the files. An evolution of the two former methods gave birth to the adaptive steganography, also known as “Statistics –aware Embedding”, “Masking” or “Model-Based”. The revolution in this method is that it collects statistical global features [60] of the image in order to decide where to apply the LSB method, achieving maximum efficiency.

More or less the same methods are used in sound files, depending on their encoding. Video files are a combination of sound and pictures. Pictures change rapidly while sound is being reproduced. As a result steganography can be applied separately into pictures and sound at the same time. The methods used to apply steganography into videos vary, depending again on the codec used.

Through the past few years several applications have been presented with the purpose either to hide some pieces of information through the use of the former methods, or to make their existence revealed to public with applied steganalysis. Some of them, like *invisible secrets*, have commercial use[61]. There are also tools aiming only for steganography in pictures such as EzStego [62], S-Tools [63], Jsteg [66], Jphide and Jpseek1 [63]. Few academic efforts are in the direction of hiding information into high quality video files. The main problem that they faced was the size of the files. It is not easy to transfer large files through the internet in order to hide a few pieces of information. However, a very good effort regarding FLV files and high definition video has been made by Pro-Chyi Su, Ming-Tse Lu and Ching-Yu Wu, who succeeded in embedding information using the H.264 Advance Video Coding and proved that it is possible to have such files for steganographic purposes [64].

What all efforts, either academic or commercial, have in common is that they make use of the least significant bit algorithm. Up to now no one has tried some alternative method. Furthermore, there are a lot of file types that have not attracted the attention of steganography [65].

A lot of research work has utilized steganography for enhancing cloud security. Wange and Rathod [67] describe a method that uses multiple image steganography to hide confidential data. In another approach Hemaanand and Varalakshmi [68] propose to hide data into images that can be later converted into Jar files. Furthermore, Mahale and Sonale [69] combine cryptography and steganography aiming to enhance data security.

3.3. Cloud Computing IDSs

Over the last years there have been several attempts to track, disable or counter the malicious insider threat. The majority of these solutions achieve their goal by focusing on a very specific aspect of the cloud, such as the employees or the network, while only a minority of them aim to provide a general purpose solution.

Spring suggests that a firewall at the cloud border that blocks troublesome packets can reduce, but not eliminate, the risk of known malicious entities to gain access [70]. Alzain, Pardede, Soh and Thom suggest that moving from “single-clouds” to “multi-clouds” will greatly reduce the malicious insider’s threat as the information is spread among the interclouds and cannot be retrieved from a single Cloud Infrastructure [71]. Another approach focuses on employing logistic regression models to estimate false positive/negatives on 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 [72].

Magklaras, Furnell and Papadaki [73] propose an audit engine for logging user actions in relational mode, named LUARM, which attempts to solve two fundamental problems of the insider’s IT misuse domain. The first one is the lack of data repositories for insider misuse cases, which could be utilized by post-case forensic examiners to aid incident investigations. The second area highlighted is how information security researchers can enhance their ability to accurately specify insider threats at system level.

Tripathi and Mishra [74] insist that cloud providers should provide tools to the customers, which can detect and defend against the malicious insiders threats. They also mention 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” [75] suggests an approach totally different from the others. Each user’s data access log is monitored in the cloud and a sort of profiling is maintained. This type of monitoring facilitates the detection of abnormal behavior. An alternative approach is that of Cuong Hoang H. Lee [76], which achieves security in a Xen based hypervisor [77] by trapping hypercalls since they are fewer than system calls. The hypercalls are checked before their execution and thus malicious ones can be detected and countered. Combining the last two approaches, [78] takes advantage of the system calls and classifies them into ‘normal’ and ‘abnormal’ through binary weighted cosine metric and k-nearest neighbor classifier.

Paying special attention to access control mechanisms, Kollam and Sunnyvale [79] present a mechanism that generates immutable security policies for a client and then propagates and enforces them at the provider’s infrastructure. This mechanism is one of the very few methods that aim directly at malicious insiders and especially system administrators.

The term “co-residency” (or “co-tenancy”) means that multiple independent customers share the same physical infrastructure [49]. It is therefore possible to have Virtual Machines (VMs) owned by different customers being placed on the same physical machine. Since there are several methods to discover neighboring VMs on a Cloud infrastructure, it is necessary to employ countermeasures for this specific attack.

Adam Bates [80], through his approach reveals that “co-residency” detection is also possible through network flow watermarking. 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 being the introduction of a considerable network delay.

Ristenpart [33] presents the “co-residency” attack on Amazon EC2, one of the largest Cloud Infrastructures. His methodology employs network tools such as nmap [23], hping [22] and wget [24], 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. In the manifestation of the method he explores whether two instances are “co-resident” or not through a series of checks that depend on (a) matching Dom0 (host OS of a cloud infrastructure) IP address, (b) small packet round trip times, or (c) numerically close internal IP address. Project “Silverline” [81], 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. There are numerous attempts to protect Cloud Infrastructures not only from the “co-residency” attack but also from various other network stressing threats, by employing IDS. Most of them make employ multiple agents installed on different Virtual Machines and collect the data into a central point. The disadvantage is that most of these approaches introduce considerable overhead to the Cloud infrastructure since they consume significant resources [82]-[83]. An interesting approach is that of Bakshi and Yogesh [40], who transfer the targeted applications to VMs hosted in another data center when they pick up grossly abnormal spike in inbound traffic.

Alarifi and Wolthusen [84] propose to monitor the system calls in every VM host of an IaaS environment based on KVM hypervisor [85], and then to invoke statistical analysis for classifying the system calls after having collected a large amount of data that includes both normal operation and malicious actions. Rawat with Gulati, Pujari and Vemuri [78] and Sharma with Pujari and Paliwal [86] in their work utilize the kNN classifier and the binary weighted cosine metric in order to achieve a similar goal and classify the processes into normal or malicious using DARPA-1998 database. The ancestor of the latter techniques is the work of Fofmeyr, Forrest and Somayaji [87] who suggested the separation of system calls into normal and malicious using the profiling of the operation of a system. A further extension of their methodology came from Eskin, Lee and Stolfo [88] who implemented dynamic window sizes as the length of the subsequence of a system call trace which is used as the basic unit for modelling program or process behavior.

Kang, Fuller and Honavar [89] further improve the above suggestions by introducing machine learning techniques using the “bag of system calls” representation in system call sequences. Machine learning techniques are also used by Azmandian et.al. [90] and Fatemeh et.al. [91].

Although recent research efforts have significant contribution in the area of intrusion detection mechanisms, a decade ago Coull and his team [92] inspired what we have adopted in CROW. They used the system calls as a series of genes and used the Smith Waterman algorithm. However they did not use whole patterns something that has resulted in many false positives and false negatives. Furthermore, their idea has been implemented in an isolated system and has nothing to do with distributed systems or cloud computing.

Not too long ago Sotiris Ioannidis and his team [93] made use of the CUDA architecture for executing Snort [94], a modern network intrusion detection system (NIDS), calling their system Gnort. They managed to transfer large portion of the overhead to the GPU, thus speeding up the efficiency of the NIDS, reducing at the same time the overhead on the CPU.

During the past few years the continuous evolution of the CUDA technology and the power enhancement of GPUs, have attracted the attention of researchers who have done numerous attempts to parallelize and implement genetic algorithms into CUDA versions. A well-known effort is the CUDASW++ [95], a project which accelerates the Smith Waterman algorithm through the GPU. However, this implementation focusses on protein database searches and does not take into consideration system calls sequences or other intrusion detection models.

Chapter 4: Enhancing the Cloud Security

4.1. The Authentication Problem

The unique cloud computing characteristics, such as elasticity, scalability and the pay-as-you-go model, can offer cost-savings and rapidly available virtual resources for the easy deployment of different types of services. In case of services that handle sensitive data, like personal healthcare records (PHRs) and e-government services, security and privacy requirements, according to the corresponding legislation, should be also considered. For instance, in USA a healthcare provider must comply with the Health Insurance Portability and Accountability Act (HIPAA) rules [96]. Thus, important challenges that must be faced before the migration of data on a cloud infrastructure are both entity authentication, as well as retention of sensitive data processed in a healthcare scenario.

In that context, various approaches have been proposed to enhance users' authentication and confidentiality of "sensitive" data. One aspect for enhancing cloud security is the utilization of steganographic techniques. Mahale and Sonale [69] combine cryptography and steganography aiming to achieve better data security. Similarly, Hemaanand and Varalakshmi [68] propose to hide data into images that can be later converted into compressed files. In addition, Wange and Rathod [67] describe a method that uses multiple image steganography to hide data, creating multilevel security. Finally, Liu [97] has developed a mechanism to counter attacks based on the steganography of audio files in cloud environments.

Though such approaches can enhance the security of cloud-based environments, none of them focuses on the enhancement of users' identification and the preservation of users' sensitive data. In this thesis, an authentication scheme for cloud-based environments that employs two-factor authentication through a password and a one-time secret key hidden on a stego is introduced. Despite the fact that encryption by itself is a secure layer, the stego layer is added in order to significantly improve the security level with the introduction of only minimum overhead. Moreover, this extra security layer may further enhance the power of the cryptographic key that cloud providers, with low security awareness, employ for the encryption of all their users' data. Its usability is tested by considering a healthcare service scenario offered through the cloud. The proposed scheme exploits the advantages of Steganographic Cryptographic Algorithm Reallocating Available Bytes (SCARAB) presented in [54].

The proposed method has been evaluated in terms of the overhead introduced when it hides a message in different file formats such as TEXT, WAV and MP3. Results demonstrate that this overhead depends on the file size of the stego-cover and can range from one to three seconds.

4.2. The Proposed Authentication Method

4.2.1. The Architecture

To address the aforementioned security and privacy challenges a solution that enhances users' authentication and enables the protection of the confidentiality of the stored data is presented. In particular, the proposed architecture consists of two main components namely the client authenticator and the database confidentiality protector.

During the "client authenticator procedure" the user provides her username/password into the authenticator in order to produce an audio file that is being saved locally in her device and which contains a stego-key message. Afterwards, the user connects to the application server, located in the cloud, requesting a service and providing her two-factor authentication credentials (username, password and the stego key message). The application server validates the username/password and connects to a reverse authenticator in order to decrypt the hidden stego-message, matching the stego-key.

In the "database confidentiality protector procedure" the application server, through the reverse authenticator, produces the database encryption/decryption key from the decryption of the stego-key. The database can then be decrypted providing the appropriate information to the user's request.

4.2.2. Cloud Authenticator

The proposed authenticator procedure is based on the SCARAB method [54] and consists of two distinct phases. In the first phase, the stego-message, which is the database encryption key, containing the value of the expression "(user_id+year)%hour + password", is encrypted using AES [98]. In this way if a malicious user identifies the existence of a hidden stego-message, he will not be able to read it since it is in encrypted form. As for the encryption key the expression "(user_id+month)%minutes + password" is utilized. In this way a single usage key is created that lasts one minute, depending on the user's credentials and on the current date. By using the same expression the key can be decrypted at the server side.

At the second phase, the encrypted message is hidden into a random audio file, exploiting the capabilities of MP3Stego [99]. During the encoding process the new audio file that incorporates the encrypted message, is compressed and encrypted with triple-DES [124]. The passphrase used as an encryption key during this step is the expression “password + (user_id+ day)%minutes”. Following the reverse procedure and the same passphrases, the encrypted text can be extracted from the compressed audio file and eventually decrypted in order to acquire the encryption key.

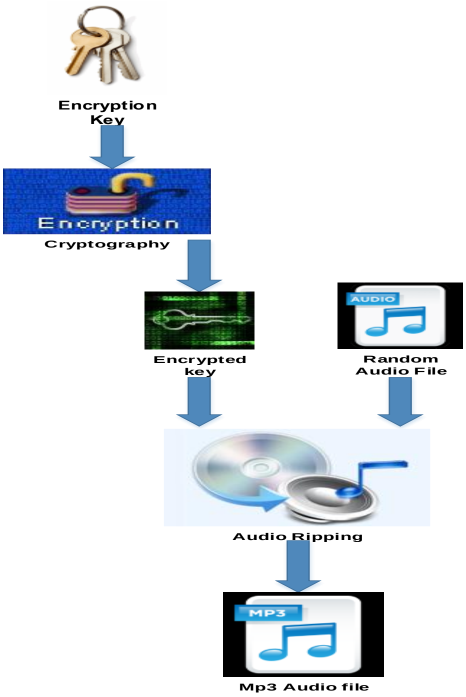

Figure 2: A high level approach of the proposed authenticator. The key is encrypted and incorporated into the audio file during its compression.

4.2.3. Case Study: A Healthcare Scenario

In order to evaluate the authentication procedure and the data processing in a cloud-based database, we illustrate the proposed approach to a healthcare scenario.

Firstly, if a patient wants to enjoy the provided medical services he should register to the medical cloud provider. As soon as the patient completes the registration form he receives a notification message (e.g. email) to prove his identity and receive the two-factor authentication credentials (username/password and the authenticator).

a. The patient uses the authenticator and after providing his username/password, an audio file which contains the hidden stego-message with the database encryption key, encrypted with AES [98], is produced and saved locally in his device.

b. He connects to the cloud-based medical application, through his local or mobile device and sends a request for a healthcare service using his username/password and uploading the extracted audio file.

c. The application server checks the patient's request and connects to a reverse authenticator in order to decrypt the hidden stego-message and thus recover the decryption key of the medical DB.

d. In case of changes in the patient record, the corresponding hospital (Hospital A, B, C) upgrades the "central" medical database with the new data, using the same encryption key that is provided by the patient himself. Data processing and data allocation procedures are upon patient's request and only he can authorize other hospital entities like physician and laboratory personnel to access his medical record, according to the implemented access control policy.

e. The medical DB is decrypted with the database key providing the appropriate information to the patient request.

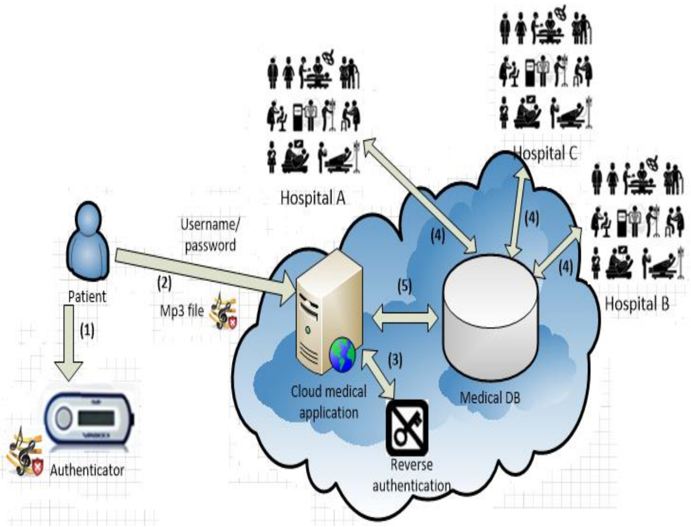

Figure 3: Patient's authentication in a cloud-based medical DB

4.3. Evaluation of the Proposed Schema in Cloud-Based Threat Scenarios

In order to evaluate our approach in terms of confidentiality, integrity and accountability, we facilitate five threat scenarios on cloud computing environment. First, we tested our approach against infrastructure and host related threats, second against service provider threats and third against generic threats. In particular, two types of threats were chosen from the first category; password guessing/cracking and unauthorized data access, two types from the second; replay and data interception and one type from the last category; encryption key exposure/loss. A detailed analysis and terminology of the above threats is given in [55], [100], [101].

Figure 4 depicts four threat scenarios against a service provider, a cloud provider and a user. The red line represents the three threat categories and the blue line represents the normal flow on the cloud and between the user, the service provider and the cloud provider (user-service provider-cloud provider).

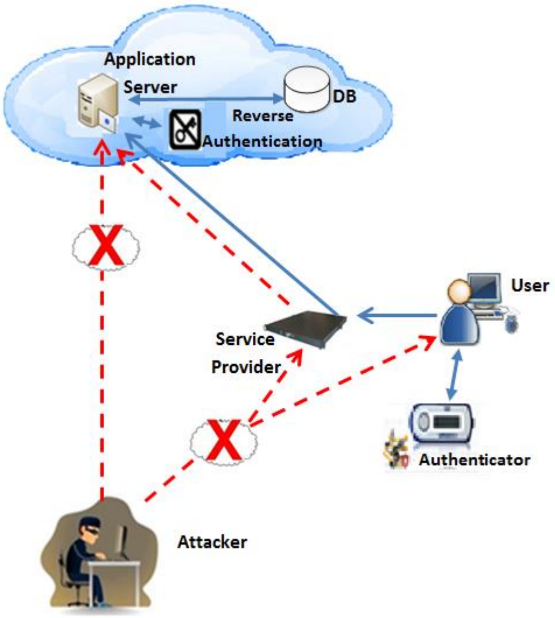

Figure 4: Threat scenarios on cloud computing environment under the auspices of cloud authenticator

Initially the password guessing threat was evaluated and it was derived that for a complex username/password (i.e. containing eight alphanumeric characters with special symbols) an attacker will need in average three hours to crack it. In contrast, the utilization of the propose`d two factor authentication, an attacker must first ensure that the container of the message is useful for him, he must then decrypt the stego-message and finally he has to find the decryption keys from the stego-message. As a result, the additional security layer (steganography) introduces more complexity to the attacker and an additional estimated time of twelve hours in order to find the keys.

The evaluation of unauthorized data access includes a check on the correct implementation of the access control policy as well as the effectiveness of the authentication process. Considering the second aspect, since the access control policy is a working hypothesis scenario, the proposed approach is certainly more robust, since someone requires a two factor authentication in order to access any kind of information.

Considering the second threat type, replay and data interception, the proposed approach is almost tolerant to those since data is communicated through the network, between the user, the service provider and the cloud provider, in encrypted form. The key lasts one minute, depending on the user's credentials and on the current date.

Finally, for the third threat type, the proposed mechanism is evaluated against encryption key exposure/loss. It has been demonstrated that the additional security layer (steganography) is an efficient solution, since the encryption keys is encrypted in the stego-message, providing a higher level of security without adding a considerable extra time (this is justified in the following chapters).

4.4. Complexity Analysis

As already mentioned, the proposed approach consists of two passphrases: one for steganography using triple DES and one for cryptography using AES. According to our initial assumption the method introduces considerable overhead to the potential attacker while, as it is proved in the next chapter, it does not add considerable overhead to our system.

AES encryption can be broken using biclique attack [102]. Results show that the computational complexity of breaking AES encryption through this attack are $2^{126.1}$ for AES-128, $2^{189.7}$ for AES-192 and $2^{254.4}$ for AES-256 respectively. Related-key attacks [103] can break AES-192 and AES-256 with complexities 2^{176} and $2^{99.5}$, respectively. As matters DES, a bruteforce attack can break it [104], having a complexity of 2^{43}, while in the case of triple DES the complexity is raised to 2^{112}. As a result it is easily proved that the co-existence of triple DES and AES strengthens the security of our key as it enlarges the complexity of breaking it for potential attackers. Furthermore, by the time the key will be retrieved it will not be useful to the attackers.

4.5. Overhead Evaluation And Discussion

We have evaluated the proposed scheme in a test-bed system with 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 @1000rpms. The openSUSE [105] distribution was utilized as a cloud platform, accompanied by the Xen Hypervisor [77]. Furthermore, a virtual machine containing windows 2008 acts as the database server hosting an Oracle database 12C suitable for multitenant environment and scalable and secure deployment. For the AES [98] encryption the cryptopp library [106] has been employed. For launching attacks against the cloud, the security tools of Kali Linux were used.

Three different scenarios (Table 3) have been executed for the evaluation of the cloud authenticator’s overhead. For each test different audio files with different sizes have been used and the average time required for completing the procedure and for the reverse procedure has been monitored.

Table 3: Evaluation Scenarios for Authenticator

Test Case	Description
1st	In the first scenario we embed a secret message of 1kb to an audio file of 1781kbs.
2nd	In the second scenario we embed a secret message of 2kbs to an audio file of 2501kbs.
3rd	In the third scenario we embed a secret message of 4kbs to an audio file of 9037kbs.
Summary	In the summary scenario we embed secret message of 7kbs to audio files of 13319kbs.

Although the proposed authenticator utilizes file sizes that mostly match the first scenario, in order to estimate the overhead two more scenarios have been designed in which the size of the files has been significantly increased. The results of the tests performed appear in Figures 5 and 6. Particularly, during the first step, the size of the text file made no difference in the normal procedure or the reverse one, as the time needed to encrypt or decrypt was in all cases approximately one second. Overhead was mainly introduced during the second step and specifically from the procedure of steganography and steganalysis. The steganography of the encrypted file into the uncompressed audio file required one to four seconds and the steganalysis for the extraction of the encrypted file required from one to six seconds. It should also be noted that the average encryption of the entire database took 88.8 seconds while the average decryption about 90 seconds.

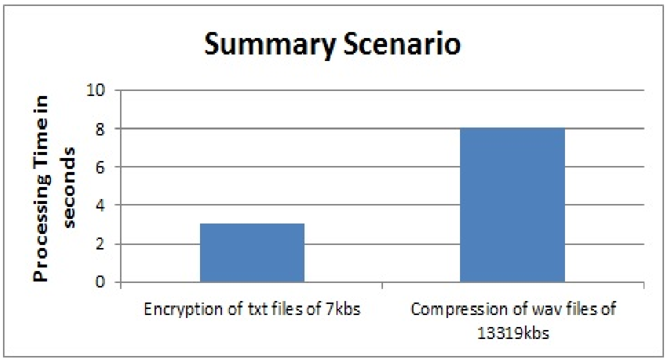

Figure 5: TXT files of 7 kilo bytes have been encrypted and then embedded into WAV files of 13319 kilo bytes.

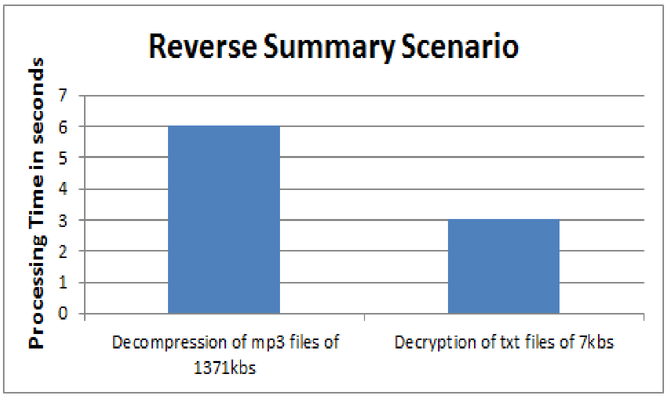

Figure 6: TXT files of 7 kilo bytes have been extracted from the audio file and have been decrypted

The proposed method, as depicted in Table 2, has a lot in common with the former techniques, but it also exhibits unique characteristics. It makes use of the referred methods and at the same time combines them in order to achieve a deeper file hiding, not easily traceable by common tools and steganalytic methods.

Table 4: Comparison of methods

Method	Cryptography	Steganography	Steganographic Container	Multilayer
Mahale and Sonale	√	√	Image	√
Hemaanand and Varalakshmi	–	√	Image	√
Wange and Rathod	–	√	Image	√
EzStego	–	√	Image	–
S-Tools	–	√	Image	–
Jsteg	–	√	Image	–
Jphide and Jpseekl	–	√	Image	–
Mp3stego	–	√	Sound	√
Proposed method	√	√	Sound	√

Chapter 5: Detecting Malicious Insider Threat

5.1. Introduction

Distributed systems have caused an important 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 [21] came with distributed systems and still draws the attention of security experts, while social engineering attacks [10], malware and Trojan horses [19] 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 [8], data interception [19] and replay attacks [19].

Our work focuses on the older and most unpredictable threat that exists even before IT systems were born: the human factor. We refer to the *Malicious Insiders* [8] [107] of a Cloud Computing Infrastructure. Their activities can harm the confidentiality, integrity and availability of the data and services of a cloud system. The most common 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 [33]. To be specific the stressing of the network is the basic component of DOS and DDOS attacks [108] , 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 [33], we talk about the detection of neighboring 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 [108], [109] in their DOS and DDOS 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 [110]. However, none of these attempts has managed to successfully prevent the actions of malicious insiders.

This work, presents a novel method for identifying network based attacks on a cloud infrastructure. To this respect a XEN [77] and a KVM-based [111] system have been employed with their 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 [52] to prove that by monitoring the system calls, the malicious actions of a potential cloud insider can be detected.

5.2. “Co-residency” and Network Attacks

It can be deduced that the majority of attacks that can be launched by insiders for detecting neighboring 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 for demonstrating the proposed detection method are very simple. Before explaining the attacks it should be stressed 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 [112], the ancestor of Backtrack Operating System [113], 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) utilizing 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 achieved by launching a smurf attack [50] 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 ipconfig command, which can be executed on the Host. The second method is detecting IPv6-active hosts on the same network via the ping6 command [114]. 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.

5.3. Detection Method

5.3.1. Algorithm

The proposed detection scheme has adopted the standard Smith-Waterman algorithm which was originally introduced in the context of molecular sequence analysis [52]. 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:

$$S(i,j) = +1, \quad if\ A(i) = B(j)$$
$$and$$
$$S(i,j) = -Gp, \quad if\ A(i) \neq B(j),$$

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

Initialization

A similarity grid, H, is created with its first row and column being initialized to zeros, i.e.:

$$H(0,j) = 0, \quad j = 0, \dots N$$
$$and$$
$$H(i,\mathbf{0}) = 0, \quad i = 0, \dots M$$

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

$$H(i,j) = \max\left\{\begin{matrix} 0, \\ H(i-1,j-\mathbf{1}) + S(i,j), \\ H(i-k,j) - (1+k*Gp), k = 1, \dots Pv, \\ H(i,j-l) - (1+l*Gp), l = 1, \dots P\boldsymbol{h} \end{matrix}\right\}, \mathbf{i} = \mathbf{1}, \dots, \mathbf{M}, \mathbf{j} = \mathbf{1}, \dots, \mathbf{N},$$

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.

5.3.2. Proposed Method

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 [115] 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 [116], 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.

5.4. Test-bed Environments and Results of the Experiments

5.4.1. Setup the Environments

In order to launch the attack and monitor the system logs, two minimal Cloud Infrastructures were built using two 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 servers were running OpenSuse Linux 12.1 [117]. Also the Linux audit [115] 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, in the XEN [77] based server one VM was installed with Backtrack Linux [113] (see Figure 7) while in the KVM based two VMs were installed with Kali Linux [112], containing the majority of the tools used for penetration testing and attacks, were set up on the server (see Figure 8).

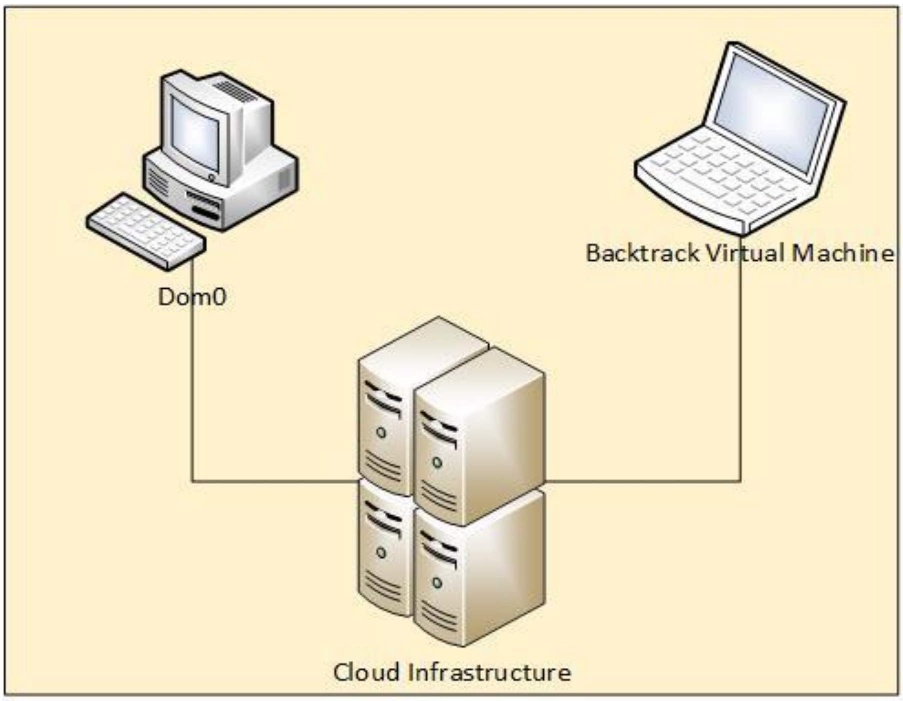

Figure 7: Xen Test-Bed environment

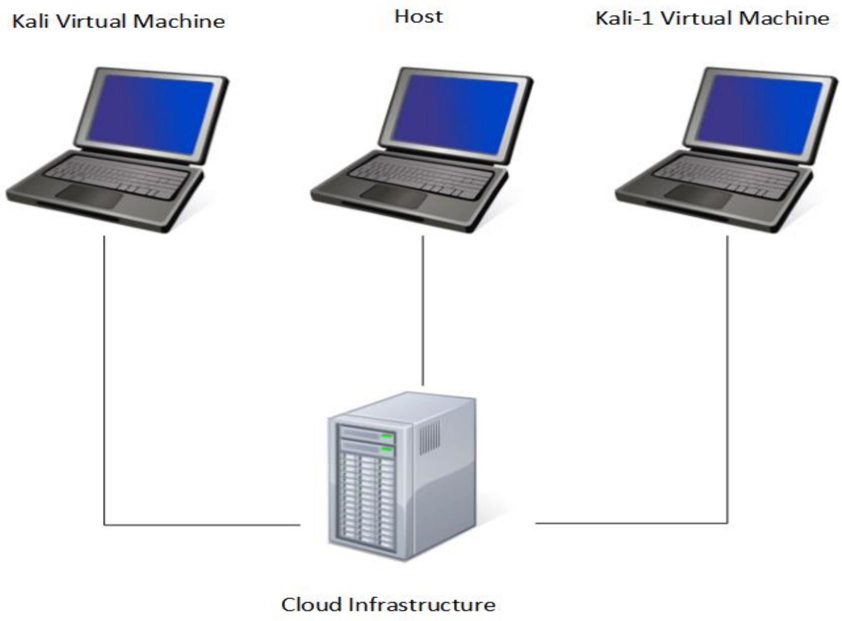

Figure 8: KVM Test-bed Environment

5.4.2. 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 [118]. 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 [119] 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" command. Finally, the Linux auditing system is disabled and the saved system calls are extracted.

5.4.3. 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) were executed in both servers and the step of smurf attack (smurf6 <interface> victim_ipv6_address) in the KVM server. The attack steps were executed six times, each time capturing all 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 Figure 9 for XEN server and in Figures 10 and 11 for KVM server.

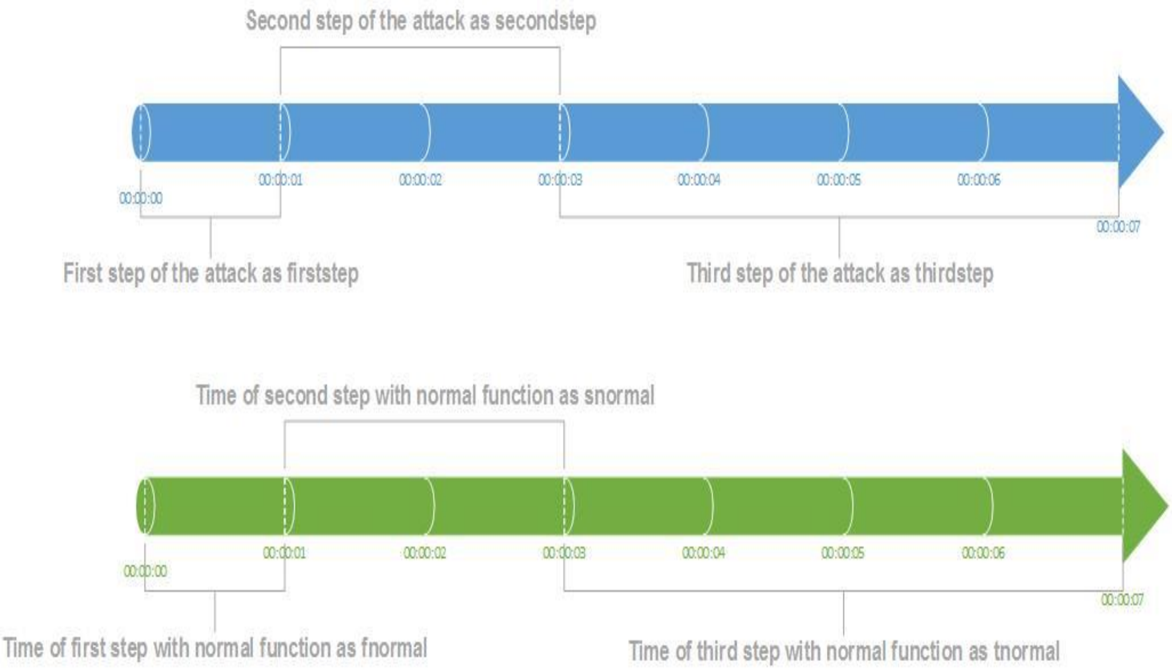
Second step of the attack as secondstep
00:00:00
00:00:01
00:00:02
00:00:03
00:00:04
00:00:05
00:00:06
00:00:07
First step of the attack as firststep
Third step of the attack as thirdstep
Time of second step with normal function as snormal
00:00:00
00:00:01
00:00:02
00:00:03
00:00:04
00:00:05
00:00:06
00:00:07
Time of first step with normal function as fnormal
Time of third step with normal function as tnormal

Figure 9: Time periods for the execution of the three attack steps (blue) and the respective time periods that the system was kept idle (green) (XEN server)

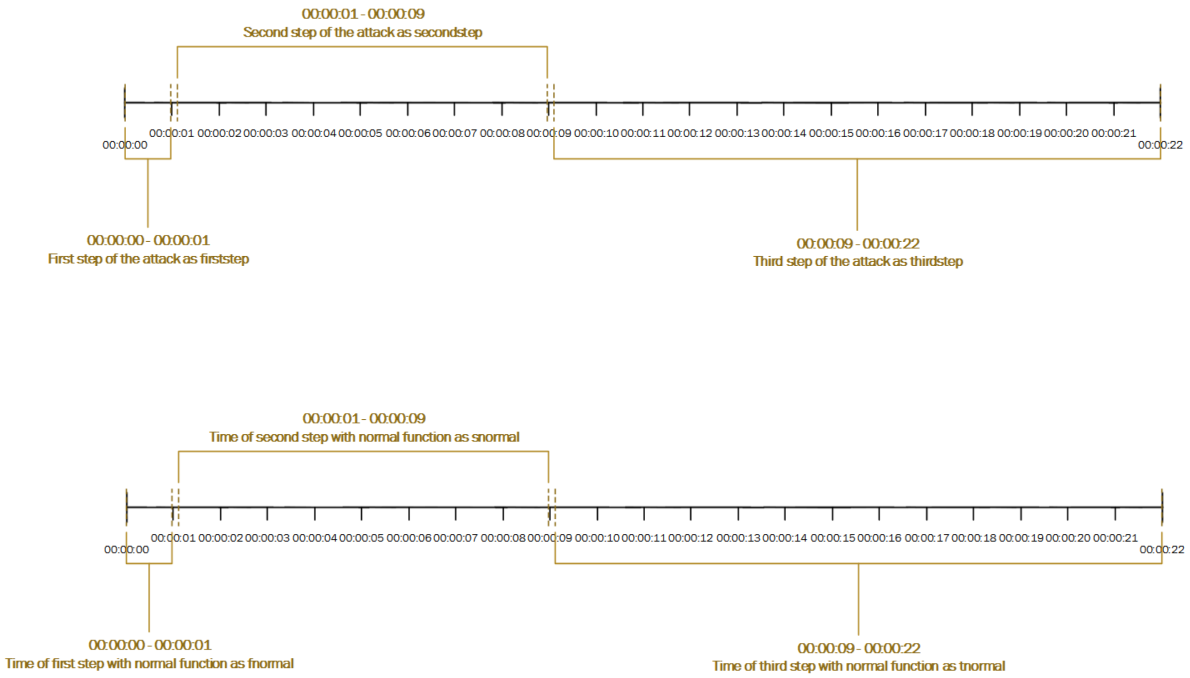

Figure 10: Time periods for the execution of the three attack steps and the respective time periods that the system was kept in normal state (KVM server)

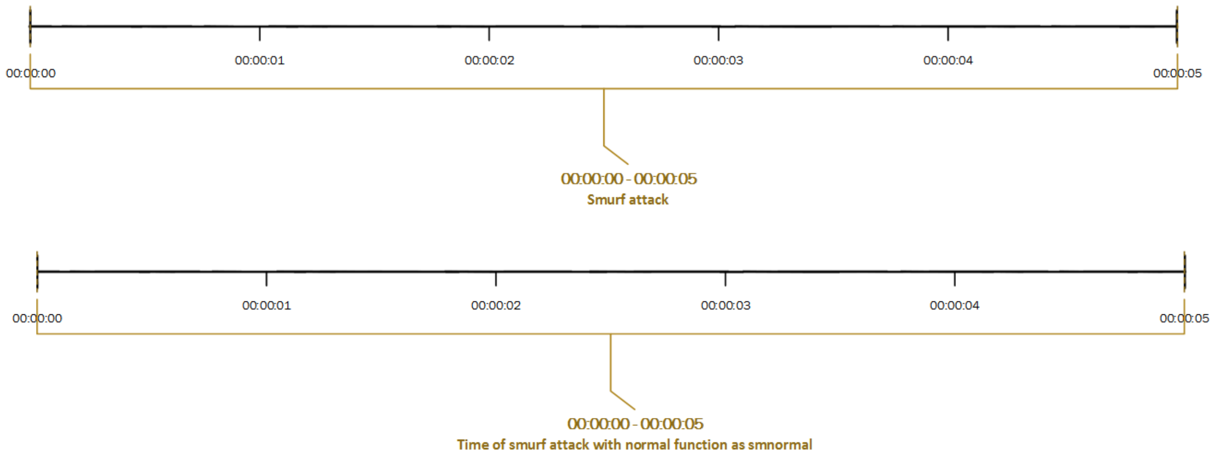

Figure 11: Time periods for the execution of the smurf attack and the respective time periods that the system was kept in normal state (KVM server)

5.4.4. Results

The results of the log files comparison are presented in the following tables. As illustrated in Figures 9-11, 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 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 matrix where we compare the execution of each step of the attack with the noisy normal operation of a system which performs continuous network operations greatly increasing the system calls.

Table 5: 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 (XEN)

Log File Comparison	*Gp* =1/3	*Gp* =1/5
firststep 1-2	10471	10627.200000
firststep 2-3	9292	9393
firststep 3-4	9136,333333	9221.800000
firststep 4-5	8255,333333	8448.800000
firststep 5-6	8693,333333	8935.600000

Table 6: 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 (XEN)

Log File Comparison	*Gp* =1/3	*Gp* =1/5
firststep1 –fnormal1	1783	1807.400000
firststep2 –fnormal2	1993	2020.200000
firststep3 –fnormal3	1983.333333	2005.200000
firststep4 –fnormal4	2601	2688
firststep5 –fnormal5	2297	2326.400000
firststep6 –fnormal6	1721.666667	1740.200000

Table 7: 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 (XEN)

Log File Comparison	*Gp* =1/3	*Gp* =1/5
secondstep 1-2	12433.666667	12607
secondstep 2-3	12442	12608
secondstep 3-4	16093	16310.400000
secondstep 4-5	13762	13937.200000
secondstep 5-6	13617	13818

Table 8: 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 (XEN)

Log File Comparison	*Gp* =1/3	*Gp* =1/5
secondstep1 –snormal1	7281.666667	7379.400000
secondstep2 –snormal2	847.333333	1236.200000
secondstep3 –snormal3	5693.666667	5919.400000
secondstep4 –snormal4	7755.666667	7863.800000
secondstep5 –snormal5	7247.666667	7360.200000
secondstep6 –snormal6	7411.666667	7558.800000

Table 9: 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 (XEN)

Log File Comparison	*Gp* =1/3	*Gp* =1/5
thirdstep 1-2	10054.666667	10330.800000
thirdstep 2-3	12834	13008.600000
thirdstep 3-4	10699.666667	10972.200000
thirdstep 4-5	10606.666667	10892.000000
thirdstep 5-6	9751.666667	9905.600000

Table 10: 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 (XEN)

Log File Comparison	*Gp* =1/3	*Gp* =1/5
thirdstep1 –tnormal1	5602	5687.400000
thirdstep2 –tnormal2	6070	6137.200000
thirdstep3 –tnormal3	5812.666667	5891.600000
thirdstep4 –tnormal4	6175.666667	6333.000000
thirdstep5 –tnormal5	6391.666667	6476.800000
thirdstep6 –tnormal6	4263	4332.400000

Table 11: 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 (KVM)

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 12: 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 (KVM)

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 13: 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 (KVM)

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 14: 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 (KVM)

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 15: 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 (KVM)

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 16: 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 (KVM)

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 17: 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 (KVM)

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 18: 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 (KVM)

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 19: Comparison of the two log files for each attack step with normal execution with generated noise from network operations for Gp equal to 1/3 (KVM)

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

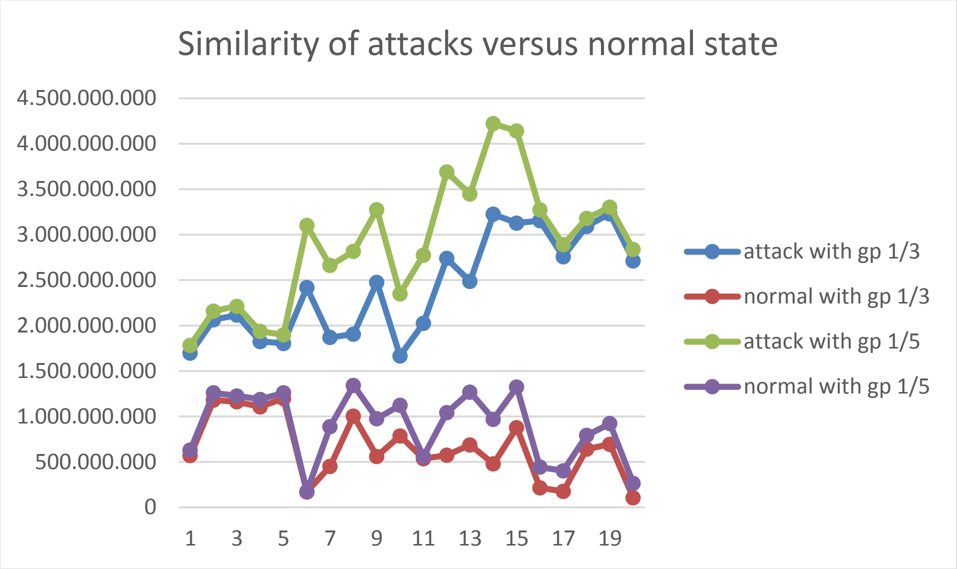

Figure 12: Graph depicting similarity between attacks and between attacks and normal system state for gp 1/3 and 1/5 respectively. Lower Gp offers greater similarity (KVM)

5.5. Discussion

Let's recall the main objective of identifying 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 of 1/3. Of course this is proved with our results, which were presented in the previous section.

In addition to that, attention must be paid 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). 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.

Chapter 6: Improving the Detection Performance

6.1 Introduction

Cloud Computing cannot offer physical isolation among virtual machines (VMs), since its resources are shared by design. Various attack vectors have been developed to identify shared resources and gain unauthorized access to them. Shared memory vulnerabilities [33], privilege escalation [8], and co-residency [33] are only a few examples of attack vectors harming cloud's confidentiality, integrity and availability. Compared to the traditional IT services, cloud attack surface has been expanded not only because of the shared resources, but also due to the additional attacking points that an adversary may utilise in order to exploit a vulnerability, e.g., a VM, a cloud management platform, or any other component of the cloud infrastructure.

Current approaches inherit methods from conventional information systems in order to reduce the effects of malicious actions performed through the VMs. Spreading the information into multiple parts in the cloud [71], creating multiple ids [82] or audit mechanisms [73], are a few of the solutions currently being proposed. In other approaches a network or data isolation is employed for securing the cloud infrastructure [81]. Others monitor the system calls in order to detect malicious activities [84], [120], [86].

The aforementioned approaches can be effective in detecting attacks launched on conventional information systems, but they are not appropriate to detect attacks launched against cloud infrastructures from privileged users; the reason being that the majority of them may be executed from separate VMs and do not appear as a threat to conventional IDS systems.

The CROW (Cloud Realtime Observation Wards) methodology is a new proposal for detecting malicious activities against the VMs and the cloud infrastructure. The principle of the proposed methodology is to monitor the system calls of each VM independently, in a way similar to a host based IDS, and then to combine the gathered information in order to detect attacks not only against individual VMs but also against the infrastructure through various compromised VMs.

The proposed solution operates on the cloud infrastructure as a service layer, in a transparent manner – meaning that no modifications to the underlying layers are required.

Specifically, we make use of the ‘strace’ command [121] to monitor the system calls of each VM and we then process them in order to generate the attack patterns and detect abnormal behaviors. In contrast to other cloud IDSs [108] that use machine learning classifiers as black-box, the proposed system generates attack patterns using the Smith-Waterman algorithm [52] and performs similarity tests between the attack patterns and the data (system calls) collected, in order to decide if an attack has been launched or not. The similarity tests are performed through a parallel implementation of the Smith Waterman algorithm on NVIDIA CUDA technology [53], which offers a significant improvement of performance in comparison to the sequential execution of the algorithm. On top of that, this approach frees the main resources of the system (CPU and memory) transferring the processing to the Graphic Processor Unit (GPU).

6.2 Threat Model

According to [48], the term “insider”, for an information system, applies to anyone with approved access, privilege, or knowledge of the information system and its services and missions. On the other hand, a “malicious insider” is someone motivated to adversely impact an organization’s mission through a range of actions that compromise information confidentiality, integrity, and/or availability taking the advantage of his/her privileges. Similarly, in the case of cloud computing we define an insider as an entity who:

- Works for the cloud host
- Has privileged access to the cloud resources
- Uses the cloud services

Consequently, cloud insiders are mostly privileged users, who may be motivated to compromise the cloud infrastructure’s security. Their actions may result in a temporary break, permanent interruption of the provided services, or in legitimate users’ privacy violation, depending on their privileges. Note that there is VM related information that can be extracted only by privileged users, such as the structure of the virtual network build up for the internal communication, and exploited during attack’s next steps. In this direction, a malicious user may try to cartography all the available virtual machines and extract other VM related information [49] in order to achieve his aim that is to violate cloud security or users’ privacy.

For instance, malicious users may combine various utilities such as nslookup, ping commands and the nmap tool, to identify publicly accessible information for a specific domain of VMs. These actions will result in launching an attack named “co-residence” or “co-tenancy”[33]. Even though these “scans” are harmless, the extraction of such information can be used for future attacks (e.g. exploiting vulnerability in a specific operating system).

Alternatively, an internal malicious user may try to affect directly the availability of a virtual network by congesting the corresponding public and private interfaces with numerous ping requests. Network stressing can also be launched through smurf attacks [50].

Furthermore, the fact that cloud infrastructures lack physical isolation can lead to memory leakages among different VMs. For instance, a malicious VM might try to get access to the shared memory (cache or main memory) and retrieve personal information for the users of the co-resident VMs. In this context, Ristenpart et.al. [33] perform cross VM side channel attack on Amazon EC2 and measure the cache activity of other users, while Rochsa and Correia [51] prove that any malicious privileged user can use the memory dumps of a VM to acquire information about its users, such as passwords, social security number and other personal information.

6.3 Cloud Realtime Observation Wards

6.3.1 Overview

A novel scheme, namely CROW, that specifically aims at detecting malicious privileged users in the cloud and also provides IDS functionality for the entire infrastructure by individually monitoring the health of each employed VM, is presented. To the best of our knowledge CROW is the first of its kind. Its high level architecture is depicted in next Figure

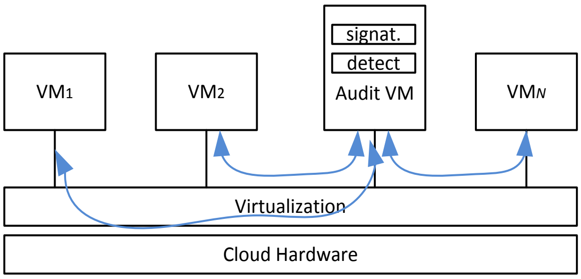

Figure 13: The CROW Architecture

The audit sub-system monitors the health of each of the provided VMs and is responsible for generating new attack signatures, based on the system call patterns of the attacks. The initial attack signatures have been generated through the analysis of well-known attacks. The detection module monitors each VM and utilizes the attack signatures for computing their similarity with the system calls issued by the VM.

6.3.2 Attack Signature Generation

The attack signature generation process consists of two steps. During the first step, the strace [121] command is used for recording the system calls produced during the execution of the attack. Having collected a significant number of system call patterns, following multiple executions of the same attack, they are processed with the Smith-Waterman algorithm [52]. The choice of this specific algorithm has been based on the fact that the data set (system call patterns) that we need to process consists of symbols drawn from a finite discrete alphabet.

The Smith-Waterman algorithm is a dynamic programming algorithm which relies on the construction of a similarity grid between two data sequences that are aligned. The goal of the algorithm is to extract a part of grid nodes which reveal the optimal sequence alignment. To achieve this goal, the algorithm processes the grid iteratively and accumulates a similarity score at each node. During this mode of operation, a node is examined with respect to a possible set of predecessors and the best predecessor is selected. The transition from a predecessor to the target node has the effect of increasing or decreasing the accumulated similarity on the target node, depending on the geometry of the transition. In our work we run the Smith-Waterman algorithm in pairs of two sequences of system calls for the same attack and in each run we reduce the number of our sequences to half, taking the best similarity match. Continuing this iteration for a number of times, we end up with the best similarity match for system calls after having processed all of our results, creating a pattern of the attack.

Specifically the generated attack signature is the sequence of the system calls invoked during the execution of the attack commands. For instance, every time that the nslookup command is executed it produces a series of system calls with some of them being consistently present, while others appearing only occasionally. The latter system calls actually represent “produced noise” and they do not belong into the command pattern. The use of the Smith-Waterman algorithm aims to diminish that noise and thus keep only the pattern of system calls that clearly represent the execution of the specific command, in our example the nslookup command.

Let’s assume that we need to create a signature for “co-residency” detection. In order to do that, we should first load a test OS on the test VM and then proceed with the execution of the attack in three distinct steps.

During the first step the “nslookup” command is executed and the systems calls invoked are recorded with the help of the “strace” command. The command should be executed several times (let’s assume x times) in order to be statistically correct, storing every time the generated system calls. After the x sequences of system calls have been collected, the Smith-Waterman algorithm will be invoked x/2 times, as it is necessary to compare sequence 1 with sequence 2, sequence 3 with sequence 4, etc. In this way x/2 sequences of system calls will be generated by the similarity of the x initial ones. Then we shall be able to produce x/4 sequences of system calls etc. until we reach the final sequence which has no other to be compared with. This sequence will form the signature for the first part of the attack, which is the “nslookup” command. A sample of the “nslookup” signature is illustrated in Table 1.

Then the same procedure will be followed for the remaining two steps of the attack (commands “nmap –sP DNS_address/24” and “nmap –v –O target_ip_address”). When the signatures for the three distinct attack steps have been generated, they can be combined to form the signature of the “co-residency” attack.

Table 20: nslookup command’s sample signature

11	45	33	91	33	5	28	91	6	33		5	3	28

6.4 Detection Module

The attack signatures can now be utilized for the detection of potential malicious acts. A detection example is illustrated in Figure 14. On the right of the picture a possible pattern is depicted which has been stored in the signature database of the audit VM. This specific signature represents an attack consisting of three different parts (segments). On the left side of the picture the system calls generated by the VM are shown. The VM detection module aims to identify the attack segments into the entire sequence of system calls, avoiding the possible noise that has been created by various other irrelevant system calls. Following the identification of all attack segments, an alert is being sent to the audit station describing the attack match. Then the operators of the audit station will take action contacting the Host VM, which has the authority to do whatever necessary for protecting the entire infrastructure.

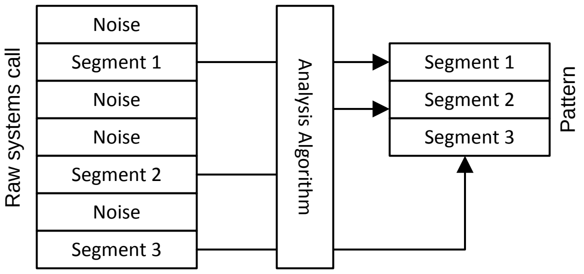

Figure 14: The segments of the attack pattern are found through the system call sequence

As already mentioned, the “co-residency” attack consists of three distinct steps, the nslookup command and two different executions of the nmap command. These three commands when executed in sequence implement the “co-residency” attack. Having the signature of the "nslookup" command, which is harmless in solo execution, will not cause a false alarm consuming unnecessarily time and useful resources. The alarm will be triggered only when all three steps of the attack have been detected in sequence. As the network of a cloud infrastructure is continuously redefined, the solo execution of the "nslookup" or even the "nmap" commands will not bring great results to potential attackers as they have to be performed in sequence and soon enough to earn the pieces of information needed.

To avoid any potential actions that will lead in hiding an attack from the audit station every two seconds the audit station initiates a handshake with each of the VMs to clarify that the communication is good between them.

6.5 Algorithm Implementation and Performance

As already mentioned the comparison between the attack patterns and the sets of system calls is performed through a parallel CUDA-based [53] implementation of the Smith Waterman algorithm [52]. In [56] we have provided generic tests of sequential Smith Waterman implementation. There it was proved that even if we loosen the error gap or even increase the noise inside the data sequence the algorithm was still able to prove itself capable of detecting the similarity. In these tests we used a matlab implementation of the algorithm, simplified to our needs as matters comparison of system calls sequences. The matlab pseudo code is listed in the appendix section.

It can be easily noticed that this version of the algorithm is sequential and thus it cannot be employed in real time due to performance limitations. An option for accelerating it would be to implement the algorithm in C language. This is something that we did try, but again the heavy needs of the Smith Waterman algorithm for computing resources didn't allow for significant improvement.

Aiming not only to improve the security level of a Cloud infrastructure but also minimize the overhead of the algorithm's execution, we have capitalized on the work of Ioannidis and his team [93] for transferring the computational overhead to the GPU. In terms of Cloud Computing systems, GPUs are rarely used autonomously by the VMs although this function can be supported by hypervisors such as XEN [77].

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce. CUDA gives program developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Each

CUDA GPU depending on the CUDA compute capability (a measure for CUDA computational power), architecture and memory, can manage different number of threads at the same time. For example a Fermi architecture GPU consists of 512 CUDA cores. These 512 CUDA cores are split across 16 Streaming Multiprocessors (SM). Each streaming multiprocessor (SM) has 32 CUDA cores. Each CUDA core consists of an integer arithmetic logic unit (ALU) and a floating point unit (FPU). Additionally, each GPU has a maximum number of threads per block, blocks per grid and of course grid blocks. These values can give us the measures for the maximum capacity of threads we can use. Fermi technology and later support 1024 threads per block.

In order to transform the previous algorithm into CUDA capable we had to make a few adjustments. First of all we reduced the size of matrixes used from 2d to 1d, in order to load only one matrix each time to the GPU memory and execute all the procedures needed in a parallel mode. Then, each coordinate used in the algorithm was transformed into 1d coordinate so as to select the exact same element as in our original version. Before each execution we transferred the matrix information onto the GPU memory to lighten the burden of our main system and free the resources. Then, the final step was to merge each sequence of “for loop” with each next one in order to achieve maximum parallelism and execute initialization, horizontal scan and parallel scan in parallel and not sequentially as shown in the upper algorithm.

Thus, the algorithm listed before has been implemented in two versions: (i) one sequential in C and (ii) a parallel version (included in appendix) that employs the CUDA multiprocessors to do the job instead of our CPU. In the tests we have used two separate data sets: one for the pattern of the attack (signature) and the sequence of the system calls produced by the VM, so as to find the similarity among them and thus compare the time spent. The results are shown in the table and figure below.

Table 21: Time spent for Sequential and Parallel execution of the Smith Waterman Algorithm

NUMBER OF SYSTEM CALLS PER DATA SET		TIME IN SECONDS	
Attack Signature	VM System Calls	Sequential Execution	CUDA parallel
10	10	0,156	0,091
20	30	0,357	0,099
80	150	4,268	0,117
334	334	46,919	0,445
500	600	120,365	0,812
1000	1000	401,937	3,209

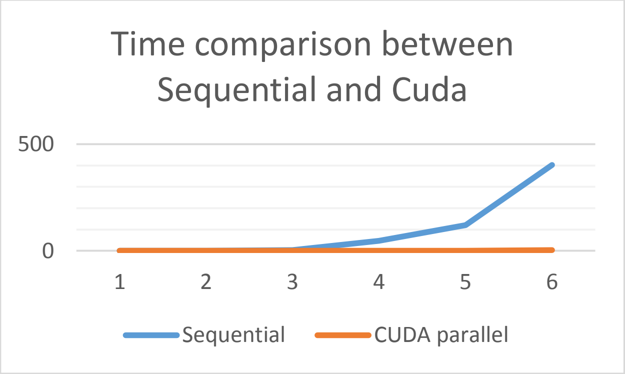

Figure 15: Time comparison between sequential and Cuda parallel execution of Smith Waterman. Sequential scales a lot along with the system calls while Cuda parallel reaches 0 seconds for any number of given system calls.

The results clearly prove our initial assumption that a parallel execution of the algorithm will greatly improve the performance during the similarity comparison procedure. As demonstrated, while the number of system calls used is increased the sequential method needs more and more time resulting in a practically unusable implementation. Furthermore, the sequential implementation is insecure since the system will need to wait a couple of minutes before deciding if someone performs an attack or not. On the contrary the parallel implementation seems to work significantly faster, reducing the execution time to almost one second for smaller numbers and to 3 seconds for larger numbers of system call sequences. Furthermore, the algorithm is able to work in parallel for different patterns of attacks (because each VM wont contain only one pattern) and does not offer significant overhead to the CPU and main memory as the modern GPUs with their memory and compute capability can support a large number of threads and large scaled matrixes in their memory.

Chapter 7: Conclusions and Future Work

7.1 Conclusion

In the introduction of this thesis we prioritized our goals as matters the contribution to science and strategic moves during our research. Our first objective was fully covered as we have completed a detailed review [55] of existing threats in cloud computing environments and have identified the malicious insider threat as one of the major ones, without sufficient counter measures.

Because of the new form of malicious insiders that was introduced along with the cloud computing we identified that this kind of attacker, which has multiple roles and of course more attack points than an outsider, imposes the need to investigate a cloud infrastructure in depth in order to check whether a potential attack can be detected. Our effort met the kernel level of the host OS, where the hypervisor is attached. We implemented a mechanism that would try and detect any anomaly coming from every VM, installed in the same infrastructure, using the system calls sequences. As we proved [56] our assumption was true, but in case of heavy workload the detection of the attacks would be more and more difficult.

As a result we came to the conclusion that we should import our method into an upper level, that of every VM. Each attacker would perform an attack either in a single VM, or by using multiple attack positions, depending on the severity of the damage he would like to cause. This is the reason why we thought that we could not detach one VM from another as matters the attacks, and a unified IDS framework would be appropriate to detect and deflect any type of attack. Consequently we created an augmented authentication mechanism that would reassure that only authorized users would have access to the infrastructure and then we implemented the SW algorithm running at the background of every VM.

Having deep knowledge of the overhead that the algorithm would offer to a potential cloud environment we did search for alternatives to either reduce and/or mitigate the overhead. Advance GPU technology, which makes use of large numbers of microprocessors emerged as the perfect solution for our problem. CUDA technology not only reduced the large overhead, caused by the use of our genetic algorithm but it also mitigated it in a part of a cloud infrastructure that has not drawn the attention of most security experts yet. Thus performance and security met each other, resulting in a framework intelligent enough to allow everyday use and expel any point that would harm the infrastructure.

7.2 Future Work

Driven by the lack of focus on malicious privileged user attacks in modern IDS systems for Cloud Infrastructures, we have proposed CROW along with an augmented authenticator. The one-time authenticator enhances the security of the data involved in cloud-based services. It consists of two distinct steps that have been evaluated, through test scenarios, in terms of security and overhead that they introduce. The results have demonstrated that the summarized encryption of TXT files of 7kbs required 3 seconds. The insertion of those encrypted messages into WAV files, during their compression to MP3, elapsed 8 seconds. As a result the stego-carrier's size greatly affects the introduced overhead of the proposed method.

CROW, is a novel detection method of malicious acts by Cloud insiders and a novel implementation of Smith Waterman algorithm based on CUDA technology. This new parallel implementation results into significant reduction of the overhead as compared to its sequential sibling. Furthermore, a sample creation of insider attacks has been presented as a guide for the creation of the attack signatures databases.

Currently, we focus on adding extra layers of either cryptography or steganography to the authenticator, in order to increase the security level without significantly increasing the overhead introduced, and we are experimenting with different Cloud Infrastructure setups and algorithm tweaks in order to achieve stability and maximum efficiency of the CROW method. Also we test the behavior and results of alternative pattern recognition algorithms that may support real time detection of attacks.

References

[1] Fischer-Hübner S.: IT-Security and Privacy. Lecture Notes in Computer Science 1958, Springer-Verlag (2001)

[2] Gritzalis D., Mitrou N., Skoularidou B.: Protecting Critical Information and Communication Infrastructure of Public Administration: Strategic Planning. eGovernment Forum-CICIP-D1-v2.3 (2008)

[3] Rinaldi S. M., Peerenboom J. P., Kelly T. K.: Identifying Understanding and Analysing Critical Infrastructure Interdependencies. IEEE control systems magazine, Vol. 21, pp. 11-25 (2001)

[4] Moteff J.: Risk Management and Critical Infrastructure Protection: Assessing, Integrating and Managing Threats, Vulnerabilities and Consequences. Congressional Research Service Report to Congress (2005)

[5] American Petroleum Institute/National Petrochemical and Refiner's Association (NPRA): Security Vulnerability Assessment Methodology for the Petroleum and Petrochemical Industries. (2004)

[6] Federal Register, Department of Homeland Security, Coast Guard: Implementation of National Maritime Security Initiatives. Vol. 68, No. 126 (2003)

[7] Latora V., Marchiori M.: Vulnerability and protection of infrastructure networks. Physical Review E. APS, (2005)

[8] Enisa, "Cloud Computing – Benefits, risks and recommendations for information security" (2009)

[9] Robinson N., Valeri L., Cave J., Starkey T., Graux H., Creese S., Hopkins P.: The Cloud: Understanding the Security, Privacy and Trust Challenges. Prepared for the Unit F.5, Directorate- General Information Society and Media, European Commission (2010)

[10] Orgill G.L., Romney G.W., Bailey M.G., Orgill P.M.: 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 (2004)

[11] Mirkovic J., Reiher P.: A Taxonomy of DDoS Attack and DDoS Defence Mechanisms. ACM SIGCOMM Computer Communication Review, Vol. 34, Issue 2 (2004)

[12] Ritchey R.W., Ammann P.: Using Model Checking to Analyze Network Vulnerabilities. Proceedings in Security and Privacy, IEEE (2000)

[13] Maybury M., Chase P., Cheikes B., Brackney D., Matzner S., Hetherington T., Wood B., Sibley C., Marin J., Longstaff T., Spitzner L., Haile J., Copeland J., Lewandowski S.: Analysis and Detection of Malicious Insiders. International Conference on Intelligence Analysis (2005)

[14] Jensen M., Schwenk J., Gruschka N., Lo Iacono L.: On Technical Security Issues in Cloud Computing. International Conference on Cloud Computing, IEEE (2009)

[15] MCintosh M., Austel P.: XML signature element wrapping attacks and countermeasures. Proceedings of the workshop on Secure web services, ACM, pp. 20-27 (2005)

[16] Gruschka N., Lo Iaocono L.: Vulnerable Cloud: SOAP Message Security Validation Revisited. International Conference on Web Services, IEEE (2009)

[17] Grobauer B., Walloshek T., Stöcker E.: Understanding Cloud Computing Vulnerabilities. IEEE Security and Privacy, Vol. 9, Issue 2, pp. 50-57 (2011)

[18] Vouk M.A.: Cloud Computing – Issues, Research and Implementation. International Conference on Information Technology Interfaces, pp. 31-40 (2008)

[19] Krutz R.L., Vines R.D.: Cloud Security: A comprehensive guide to secure Cloud Computing. Wiley Publishing Inc. (2010)

[20] Cloud Business Review, http://www.cloudbusinessreview.com/2011/05/11/three-types-of-cloud-lock-in.html

[21] Douligeris C., Mitrokotsa A.: DDoS attacks and defense mechanisms: classification and state-of-the-art. Computer Networks, pp. 643–666 (2004)

[22] Hping, http://www.hping.org/

[23] Nmap, http://nmap.org/

[24] GNU Operating System, http://www.gnu.org/software/wget/

[25] CRAMM, http://www.cramm.com/

[26] Cert, http://www.cert.org/octave/

[27] Computer Based Social Engineering: Social Engineer Toolkit (SET), http://www.social-engineer.org/framework/Computer_Based_Social_Engineering_Tools:_Social_Engineer_Toolkit_(SET)

[28] TrustedSec, https://www.trustedsec.com/downloads/social-engineer-toolkit/

[29] Xen, http://blog.xen.org/index.php/2012/06/13/the-intel-sysret-privilege-escalation/

[30] Saripalli P., Walters B.: Quirc: A quantitative impact and risk assessment framework for cloud security. IEEE, Cloud Computing (2010)

[31] Subashini S., Kavitha V.: A survey on security issues in service delivery models of cloud Computing. Journal of Network and Computer Applications, Elsevier, Vol. 34, Issue 1 (2011)

[32] Agarwal A., Agarwal A.: The Security Risks Associated with Cloud Computing. International Journal of Computer Applications in Engineering Sciences, Vol. 1, Special Issue on CNS (2011)

[33] Ristenpart T., Tromer E., Shacham H., Savage S.: Hey, You, Get of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. Proceedings of the conference on Computer and communications security, ACM (2009)

[34] Raj H., Nathuji R., Singh A., England P.: Resource Management for isolation Enhanced Cloud Services. Proceedings of the workshop on Cloud computing security, pp.77-84, ACM (2009)

[35] Jamil D., Zaki H.: Cloud Computing Security. International Journal of Engineering and Technology, IJEST, Vol. 3 No. 4 (2011)

[36] openQRM, http://www.openqrm-enterprise.com/community/

[37] Cobblerd, http://www.cobblerd.org/

[38] Crowbar, http://simile.mit.edu/wiki/Crowbar

[39] Spacewalk, http://spacewalk.redhat.com/

[40] Bakshi A., Yogesh B. :Securing Cloud from DDOS Attacks using Intrusion Detection System in Virtual Machine, ICCSN ’10 Proceeding of the 2010 Second International Conference on Communication Software and networks, pp. 260-264, 2010, IEEE Computer Society, USA, 2010

[41] Roschke S, Cheng F., Meinel C. :An Advanced IDS Management Architecture, Journal of Information Assurance and Security 5 2010

[42] Cheng, F., Roschke, S., & Meinel, C. (2009). Implementing IDS Management on Lock-Keeper. In Information Security Practice and Experience (pp. 360-371). Springer Berlin Heidelberg.

[43] Roschke, S., Cheng, F., & Meinel, C. (2009, December). Intrusion detection in the cloud. In Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on (pp. 729-734). IEEE.

[44] Mazzariello, C., Bifulco, R., & Canonico, R. (2010, August). Integrating a network ids into an open source cloud computing environment. In Information Assurance and Security (IAS), 2010 Sixth International Conference on (pp. 265-270). IEEE.

[45] CloudAudit, http://cloudaudit.org/CloudAudit/Home.html

[46] Widder, A., Ammon, R. V., Schaeffer, P., & Wolff, C. (2007, June). Identification of suspicious, unknown event patterns in an event cloud. InProceedings of the 2007 inaugural international conference on Distributed event-based systems (pp. 164-170). ACM.

[47] Kaliski Jr, B. S., & Pauley, W. (2010, June). Toward risk assessment as a service in cloud environments. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (pp. 13-13). USENIX Association.

[48] Maybury, Mark, et al. Analysis and detection of malicious insiders. MITRE CORP BEDFORD MA, 2005.

[49] Xiao Z., Xiao Y., “Security and Privacy in Cloud Computing”, Communications Surveys & Tutorials, IEEE, vol. PP no.99, pp.1-17, 2012

[50] Smurf attack, http://www.ciscopress.com/articles/article.asp?p=1312796

[51] Krutz, Ronald L., and Russell Dean Vines. Cloud security: A comprehensive guide to secure cloud computing. John Wiley & Sons, 2010.

[52] Smith T., Waterman M., “Identification of Common Molecular subsequences”, J. Mol. Biol. (1981)

[53] http://www.nvidia.com/object/cuda_home_new.html

[54] Pitropakis N., Lambrinoudakis C., Geneiatakis D., Gritzalis D., "A Practical Steganographic Approach for Matroska based High Quality Video files", 7th International Symposium on Security and Multimodality in Pervasive Environment (SMPE-2013), pp. 684-688, IEEE Computer Society Press Barcelona, Spain, March 2013.

[55] Pitropakis N., Darra E., Vrakas N., Lambrinoudakis C., "It’s all in the Cloud: Reviewing Cloud Security ", Proceedings of the 10th IEEE International Conference on Autonomic and Trusted Computing (ATC 2013), pp. 355-362, IEEE Xplore, Vietri sul Mare, Italy, December 2013.

[56] Pitropakis, N., Pikrakis, A., & Lambrinoudakis, C. (2014). Behaviour reflects personality: detecting co-residence attacks on Xen-based cloud environments.International Journal of Information Security, 1-7.

[57] Neil F. Johnson, Sushil Jajodia, “Exploring Steganography: Seeing the Unseen”, George Mason University, Computing Practises

[58] M. U. Gelik, G. Sharma, A. M. Tekalp, and E. Saber. Lossless generalized LSB data embedding, IEEE Transactions on Image Processing, 14(2):253-266, February 2005

[59] Ira S. Moskowitz, Patricia A. Lafferty, Farid Ahmed, “On LSB Spatial Domain Steganography and Channel Capacity,” March 21 2008

[60] N. F. Johnson S. C. Katzenbeisser, “A Survey of steganographic techniques. Information Hiding techniques for steganography and digital watermarking”, Norwood. Artech House

[61] "Invisible Secrets 4", http://www.invisiblesecrets.com/

[62] Romana Machado, EzStego, http://www.securityfocus.com/tools/586

[63] S-Tools 4.0, http://www.spychecker.com/program/stools.html

[64] Pro-Chyi Su, Ming-Tse Lu, Ching-Yu Wu, "A practical design of high-volume steganography in digital video files", Springer, 19 April 2011

[65] Abbas Cheddad, Joan Condell, Kevin Curran, Paul Mc Kevitt, "Digital Image Steganography: Survey and analysis of current methods", Elsevier 2009

[66] JSteg, http://zooid.org/~paul/crypto/jsteg/

[67] Wawge, P. U., and A. R. Rathod. "Cloud computing security with Steganography and Cryptoghrapy AES algorthm Technology." World Research Journal of Computer Architecture, ISSN (2012): 2278-8514.

[68] Hemaanand, M., and K. Varalakshmi. "Enhancement of Security for Data Storage in Cloud." International Journal (2013).

[69] Mahale, S. B., and Patil Sonal. "A Survey on various patterns regarding Encryption, a efficient based Method Regarding Cryptography and Steganography."

[70] Spring, Jonathan. "Monitoring cloud computing by layer, part 1." Security & Privacy, IEEE 9.2 (2011): 66-68.

[71] AlZain, Mohammed Abdullatif, et al. "Cloud computing security: from single to multi-clouds." System Science (HICSS), 2012 45th Hawaii International Conference on. IEEE, 2012.

[72] Sandhu, Ravi, et al. "Towards a discipline of mission-aware cloud computing."Proceedings of the 2010 ACM workshop on Cloud computing security workshop. ACM, 2010.

[73] Magklaras, G., Furnell, S., and Papadaki, M. 2011. LUARM: An audit engine for insider misuse detection. International Journal of Digital Crime and Forensics. 3, 3, 37-49.

[74] Tripathi, Alok, and Abhinav Mishra. "Cloud computing security considerations."Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on. IEEE, 2011.

[75] 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.

[76] Hoang C., "Protecting Xen hypercalls", MSC thesis, University of British Columbia, July 2009

[77] http://www.xenproject.org/developers/teams/hypervisor.html

[78] Rawat, S., Gulati, V. P., Pujari, A. K., & Vemuri, V. R. (2006). Intrusion detection using text processing techniques with a binary-weighted cosine metric. Journal of Information Assurance and Security, 1(1), 43-50.

[79] Sundararajan, Sudharsan, et al. "Preventing Insider attacks in the Cloud."Advances in Computing and Communications. Springer Berlin Heidelberg, 2011. 488-500.

[80] Bates A., "Dtecting Cloud Co-Residency with network flow watermarking techniques", MSC Thesis, University of Oregon, September 2012

[81] Mundada Y., Ramachndran A., Feamster N., "SilverLine: Data and network isolation for cloud services", In Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud) 2011

[82] Mazzariello C., Bifulco R., Canonico R., “integrating a Network IDS into an Open Source Cloud Computing Environment”, Sixth International Conference on Information Assurance and Security, 2010

[83] 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

[84] Alarifi, Suaad S., and Stephen D. Wolthusen. "Detecting anomalies in IaaS environments through virtual machine host system call analysis." Internet Technology And Secured Transactions, 2012 International Conferece For. IEEE, 2012.

[85] http://www.linux-kvm.org/

[86] Sharma, Alok, Arun K. Pujari, and Kuldip K. Paliwal. "Intrusion detection using text processing techniques with a kernel based similarity measure." computers & security 26.7 (2007): 488-495.

[87] Hofmeyr, Steven A., Stephanie Forrest, and Anil Somayaji. "Intrusion detection using sequences of system calls." Journal of computer security 6.3 (1998): 151-180.

[88] Eskin, Eleazar, Wenke Lee, and Salvatore J. Stolfo. "Modeling system calls for intrusion detection with dynamic window sizes." DARPA Information Survivability Conference &amp; Exposition II, 2001. DISCEX'01. Proceedings. Vol. 1. IEEE, 2001.

[89] Kang, Dae-Ki, Doug Fuller, and Vasant Honavar. "Learning classifiers for misuse and anomaly detection using a bag of system calls representation."Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC. IEEE, 2005.

[90] Azmandian, Fatemeh, et al. "Virtual machine monitor-based lightweight intrusion detection." ACM SIGOPS Operating Systems Review 45.2 (2011): 38-53.

[91] Azmandian, Fatemeh, et al. "Securing cloud storage systems through a virtual machine monitor." Proceedings of the First International Workshop on Secure and Resilient Architectures and Systems. ACM, 2012.

[92] Coull, S., Branch, J., Szymanski, B., & Breimer, E. (2003, December). Intrusion detection: A bioinformatics approach. In Computer Security Applications Conference, 2003. Proceedings. 19th Annual (pp. 24-33). IEEE.

[93] Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E. P., & Ioannidis, S. (2008, January). Gnort: High performance network intrusion detection using graphics processors. In Recent Advances in Intrusion Detection (pp. 116-134). Springer Berlin Heidelberg.

[94] https://www.snort.org/

[95] http://cudasw.sourceforge.net/homepage.htm#latest

[96] Wu, Ruoyu, Gail-Joon Ahn, and Hongxin Hu. "Towards HIPAA-Compliant Healthcare Systems in Cloud Computing." International Journal of Computational Models and Algorithms in Medicine (IJCMAM) 3.2 (2012): 1-22.

[97] Liu, Bo, et al. "Thwarting audio steganography attacks in cloud storage systems." Cloud and Service Computing (CSC), 2011 International Conference on. IEEE, 2011.

[98] “Announcing the ADVANCED ENCRYPTION STANDARD (AES)”, Federal Information Processing Standards Publication 197, November 26, 2001

[99] Fabien Petitcolas, “mp3stego” http://www.petitcolas.net/fabien/steganography/mp3stego/

[100] Marinos, L. "ENISA Threat Landscape 2013." Heraklion: European Union Agency for Network and Information Security (ENISA). DOI 10 (2013): 14231.

[101] Prislan, Kaja. "Efficiency of Corporate Security Systems in Managing Information Threats: An Overview of the Current Situation." Varstvoslovje: Journal of Criminal Justice & Security 16.2 (2014).

[102] Bogdanov, A., Khovratovich, D., & Rechberger, C. (2011). Biclique cryptanalysis of the full AES. In Advances in Cryptology–ASIACRYPT 2011 (pp. 344-371). Springer Berlin Heidelberg.

[103] Schneier, Bruce. "New Attack on AES." Schneier on Security, A blog covering security and security technology. Archived from the original on 8 (2010).

[104] Ferguson, N., Kelsey, J., Lucks, S., Schneier, B., Stay, M., Wagner, D., & Whiting, D. (2001, January). Improved cryptanalysis of Rijndael. In Fast software encryption (pp. 213-230). Springer Berlin Heidelberg.

[105] OpenSUSE, https://en.opensuse.org/Main_Page

[106] “Crypto++”, http://www.cryptopp.com/

[107] 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.

[108] 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.

[109] 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.

[110] Roschke S., Cheng F., Meinel C., “An Advanced IDS Management Architecture”, Journal of Information Assurance and Security 5 (2010)

[111] http://www.linux-kvm.org/page/Main_Page

[112] http://www.kali.org/

[113] http://www.backtrack-linux.org/

[114] http://www.tldp.org/HOWTO/Linux%2BIPv6-HOWTO/x811.html

[115] http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.audit.comp.html

[116] http://linux.die.net/man/1/sed

[117] http://www.opensuse.org/

[118] http://www.tcl.tk/man/expect5.31/expect.1.html

[119] https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Administration_Guide/chap-Virtualization_Administration_Guide-Managing_guests_with_virsh.html

[120] Rawat, Sanjay, et al. "Intrusion detection using text processing techniques with a binary-weighted cosine metric." Journal of Information Assurance and Security 1.1 (2006): 43-50.

[121] Strace, http://unixhelp.ed.ac.uk/CGI/man-cgi?strace+1

[122] Pitropakis N., Yfantopoulos N., Geniatakis D., Lambrinoudakis C., "Towards an augmented authenticator in the Cloud", Proceedings of the 14th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2014), IEEE Xplore, Noida, India, December 2014.

[123] Pitropakis N. , Geniatakis D., Lambrinoudakis C., “Till all are one: Towards a unified Cloud IDS”, submitted to Trustbus 2015

[124] Andreas Westfeld, “Detecting Low Embedding Rates”, Springer, 2003

[125] Pitropakis, N., Anastasopoulou, D., Pikrakis, A., & Lambrinoudakis, C., "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 3.1 (2014): 1-10.

Appendix

Source Code

Smith Waterman Matlab Implementation

```
penalty=1/3;
LA=length(A);
M=LA+1;
LB=length(B);
N=LB+1;
% Initialization
D=zeros(M,N);
for i=1:LA
   for j=1:LB
      if A(i)==B(j)
         D(i+1,j+1)=1;
      else
         D(i+1,j+1)=-penalty;
      end
   end
end
% eo initialization
acc_cost=zeros(M,N);
% start grid processing
for i=2:M %for every row (remember the first row is all zeros and stays like that till the end)
   for j=2:N %for every column (remember the first column is all zeros and stays like that till the end)
      temp_max=D(i,j);
% Diagonal transition
      if acc_cost(i-1,j-1)+ D(i,j)>temp_max
         temp_max=acc_cost(i-1,j-1)+ D(i,j);
      end
% Vertical scan: nodes (1,j),(2,j),...,(i-1,j)
```

```
    if i-Lver>=1
      rowstart=i-Lver;
    else
      rowstart=1;
    end
    for row=rowstart:i-1
      if acc_cost(row,j)-(1+(penalty)*(i-row))>temp_max
        temp_max=acc_cost(row,j)-(1+(penalty)*(i-row)); % the second term is the penalty term for the vertical transition
      end
    end
% Horizontal scan: nodes (i,1),(i,2),...,(i,j-1)
    if j-Lhor>=1
      colstart=j-Lhor;
    else
      colstart=1;
    end
    for col=colstart:j-1
      if acc_cost(i,col)-(1+(penalty)*(j-col))>temp_max
        temp_max= acc_cost(i,col)-(1+(penalty)*(j-col)); % the second term is the penalty term for the horizontal transition
      end
    end
% Finished (i,j).There only remains to store the winner
    if temp_max>0
      acc_cost(i,j)=temp_max;
    end
  end
end
% eo grid processing
maxv=max(max(acc_cost)); % maximum accumulated similarity
if maxv==0
  bp=[];
  return;
end
```

```
[xc,yc]=find(acc_cost==maxv); % where maxv is located
```

Smith Waterman Cuda Implementation

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/copy.h>
#include <thrust/fill.h>
#include <thrust/sequence.h>
#include <thrust/transform_reduce.h>
#include <thrust/functional.h>
_global__ void insertKernel(int *a)
{
  int x = blockDim.x * blockIdx.x + threadIdx.x;
  a[x] = x;

}



__global__ void initKernel(int *a, int *b, double penal, double *Di, int column, int laa, int lbb)
{
  int x = blockDim.x * blockIdx.x + threadIdx.x;
  //int x =threadIdx.x;
  int i,j;
  i = x % lbb;
  j = x / lbb;
  if ((i < laa) && (j < lbb))
  {
      if (a[i] == b[j])
      {
          Di[((i + 1)*column) + j + 1] = 1;
      }
      else
      {
          Di[((i + 1)*column) + j + 1] = -penal;
      }
  }
}
__global__ void verticalscanKernel(double *acc_cst, int column, double Lh, double Lv, double penal, int roww, double *temp_maxx, int ii, int jj, int *bestpredii,int *bestpredijj)
{
  int x = blockDim.x * blockIdx.x + threadIdx.x;

  //Vertical scan: nodes (1,j),(2,j),...,(i-1,j)
  //for (row = rowstart; row < i - 1; row++)
  if ((x >= roww) && (x<ii - 1))
      {
          if ((acc_cst[(x*column) + jj]) - (1 + (penal)*(ii - x)) > *temp_maxx)
          {
              *temp_maxx = (acc_cst[(x*column) + jj]) - (1 + (penal)*(ii - x)); //the second term is the penalty term for the vertical transition
              *bestpredii = x;
              *bestpredijj = jj;
          }
      }
}
__global__ void horizontalscalscanKernel(double *acc_cst, int column, double Lh, double Lv, double penal, int roww,int coluu, double *temp_maxx, int ii, int jj, int *bestpredii, int *bestpredijj)
{
  int x = blockDim.x * blockIdx.x + threadIdx.x;

  //Horizontal scan: nodes (i,1),(i,2),...,(i,j-1)
```

```
    //for (col = colstart; col < j - 1; col++)
    if ((x >= coluu) && (x<jj - 1))
    {
        if ((acc_cst[(ii*column) + x]) - (1 + (penal)*(jj - x)) > *temp_maxx)
        {
            *temp_maxx = (acc_cst[(ii*column) + x]) - (1 + (penal)*(jj - x)); //the second term is the penalty term for the vertical transition
            *bestpredii = ii;
            *bestpredijj = x;
        }
    }
}
__global__ void gridKernel(double *Di,double *acc_cst, double *row_pre, double *col_pre, int column, double Lh, double Lv, double penal, int roww)
{
    int x = blockDim.x * blockIdx.x + threadIdx.x;
    int i, j, bestPredi, bestPredj,rowstart, row, col, colstart;
    double temp_max;
    i = x % column;
    j = x / column;
    if ((i >= 1) && (j >= 1) && (i < roww) && (j < column))
    {
        temp_max = Di[(i*column) + j];
        bestPredi = 0;
        bestPredj = 0;
        //Diagonal transition
        if ((acc_cst[((i - 1)*column) + j - 1] + Di[(i*column) + j]) > temp_max)
        {
            temp_max = acc_cst[((i - 1)*column) + j - 1] + Di[(i*column) + j];
            bestPredi = i - 1;
            bestPredj = j - 1;
        }
        //Vertical scan: nodes (1,j),(2,j),...,(i-1,j)
        if ((i - (int)Lv) >= 1)
            rowstart = i - Lv;
    }
        else rowstart = 1;
        for (row = rowstart; row < i - 1; row++)
        {
            if ((acc_cst[(row*column) + j]) - (1 + (penal)*(i - row)) > temp_max)
            {
                temp_max = (acc_cst[(row*column) + j]) - (1 + (penal)*(i - row)); //the second term is the penalty term for the vertical transition
                bestPredi = row;
                bestPredj = j;
            }
        }
        //Horizontal scan: nodes (i,1),(i,2),...,(i,j-1)
        if ((j - (int)Lh) >= 1)
            colstart = j - Lh;
        else colstart = 1;
        for (col = colstart; col < j - 1; col++)
        {
            if ((acc_cst[(i*column) + col]) - (1 + (penal)*(j - col)) > temp_max)
            {
                temp_max = (acc_cst[(i*column) + col]) - (1 + (penal)*(j - col)); //the second term is the penalty term for the vertical transition
                bestPredi = i;
                bestPredj = col;
            }
        }
        //Finished (i,j).There only remains to store the winner
        if (temp_max > 0)
        {
            acc_cst[(i*column) + j] = temp_max;
            row_pre[(i*column) + j] = bestPredi;
            col_pre[(i*column) + j] = bestPredj;
        }
    }
}
```

```
int main()
{
    int counter1, patterncounter, i, LA, M, LB, N, size, size2, x, j, bestPredi, bestPredj,rowstart,colstart,row,col;
    int data_to_read;
    FILE *data1; FILE *pattern;
    int *temp_memory; int *temp_pattern; int *device_vector1; int *device_pattern;
    double penalty, Lhor, Lver,maxv,temp_max;
    double *H; double *D; double *temp_acc_cost; double *acc_cost; double *temp_row_pred; double *row_pred;
    double *temp_col_pred; double *col_pred;
    //number initiallizations
    counter1 = 0; patterncounter = 0;
    i = 0;
    maxv = 0.0;

    clock_t tic = clock();

    /*files opening and reading size*/
    if ((data1 = fopen("data1.txt", "r")) == NULL)
    {
        fprintf(stderr, "error in opening file:%s", "data1.txt");
        exit(-1);
    }
    while (!feof(data1))
    {
        fscanf(data1, "%d", &data_to_read);
        counter1++;
    }
    fclose(data1);
    if ((pattern = fopen("pattern.txt", "r")) == NULL)
    {
        fprintf(stderr, "error in opening file:%s", "pattern.txt");
        exit(-1);
    }
```

```
while (!feof(pattern))
{
        fscanf(pattern, "%d", &data_to_read);
        patterncounter++;
}
fclose(pattern);
// temp memory vector has storage for counter1 integers
temp_memory = (int*)malloc(counter1*sizeof(int));
temp_pattern = (int*)malloc(patterncounter*sizeof(int));
std::cout << "temp_memory has size " << counter1 << std::endl;
std::cout << "temp_pattern has size " << patterncounter << std::endl;
//filling temp_memory with systemcall id
if ((data1 = fopen("data1.txt", "r")) == NULL)
{
        fprintf(stderr, "error in opening file:%s", "data1.txt");
        exit(-1);
}
i = 0;
while (!feof(data1))
{
        fscanf(data1, "%d", &temp_memory[i]);
        //std::cout <<temp_memory[i] << std::endl;
        i++;
}
fclose(data1);
if ((pattern = fopen("pattern.txt", "r")) == NULL)
{
        fprintf(stderr, "error in opening file:%s", "pattern.txt");
        exit(-1);
}
i = 0;
while (!feof(pattern))
{
        fscanf(pattern, "%d", &temp_pattern[i]);
        i++;
```

```
	}
	fclose(pattern);
	//device_vector1 for pattern and data_set1
	cudaMalloc((void**)&device_vector1, counter1*sizeof(int));
	cudaMemcpy(device_vector1, temp_memory, counter1*sizeof(int), cudaMemcpyHostToDevice);
	cudaMalloc((void**)&device_pattern, (int)patterncounter*sizeof(int));
	cudaMemcpy(device_pattern, temp_pattern, patterncounter*sizeof(int), cudaMemcpyHostToDevice);
		//smith waterman start
	Lver = 5.0;
	Lhor = 5.0;
	penalty = 1.0 / 3.0;
	LA = counter1;
	M = LA + 1;
	LB = patterncounter;
	N = LB + 1;
	size = N*M;
	size2 = LA*LB;
	H = (double*)malloc(size*sizeof(double));
	cudaMalloc((void**)&D, size*sizeof(double));
	for (i = 0; i < size; i++)
	{
		H[i] = 0;
	}
	cudaMemcpy(D, H, N*M*sizeof(double), cudaMemcpyHostToDevice);
	// print contents of D
	printf("%f \n", penalty);
	//system("pause");
	int threadsPerBlock = 1024;
	int blocksPerGrid = (size2 + threadsPerBlock - 1) / threadsPerBlock;
	initKernel <<<blocksPerGrid, threadsPerBlock >>>(device_vector1, device_pattern, penalty, D, N, LA,
LB);
	cudaMemcpy(H, D, size*sizeof(double), cudaMemcpyDeviceToHost);
	temp_acc_cost = (double*)malloc(size*sizeof(double));
	cudaMalloc((void**)&acc_cost, size*sizeof(double));
	for (i = 0; i < size; i++)
```

```
{
        temp_acc_cost[i] = 0;
}
cudaMemcpy(acc_cost, temp_acc_cost, size*sizeof(double), cudaMemcpyHostToDevice);

temp_row_pred = (double*)malloc(size*sizeof(double));
cudaMalloc((void**)&row_pred, size*sizeof(double));
for (i = 0; i < size; i++)
{
        temp_row_pred[i] = 0;
}
cudaMemcpy(row_pred, temp_row_pred, size*sizeof(double), cudaMemcpyHostToDevice);

temp_col_pred = (double*)malloc(size*sizeof(double));
cudaMalloc((void**)&col_pred, size*sizeof(double));
for (i = 0; i < size; i++)
{
        temp_col_pred[i] = 0;
}
cudaMemcpy(col_pred, temp_col_pred, size*sizeof(double), cudaMemcpyHostToDevice);

//start grid processing
for (x = 0; x < size;x++)
{
        i = x % N;
        j = x / N;
        if ((i >= 1) && (j >= 1) && (i < M) && (j < N))
        {
                temp_max = H[(i*N) + j];
                bestPredi = 0;
                bestPredj = 0;
                //Diagonal transition
                if ((temp_acc_cost[((i - 1)*N) + j - 1] + H[(i*N) + j]) > temp_max)
                {
                        temp_max = temp_acc_cost[((i - 1)*N) + j - 1] + H[(i*N) + j];
                        bestPredi = i - 1;
                        bestPredj = j - 1;
                }

                //Vertical scan: nodes (1,j),(2,j),...,(i-1,j)
                if ((i - (int)Lver) >= 1)
                        rowstart = i - Lver;
                else rowstart = 1;
                cudaMemcpy(acc_cost, temp_acc_cost, size*sizeof(double),
    cudaMemcpyHostToDevice);
                cudaMemcpy(row_pred, temp_row_pred, size*sizeof(double),
    cudaMemcpyHostToDevice);
                cudaMemcpy(col_pred, temp_col_pred, size*sizeof(double),
    cudaMemcpyHostToDevice);
                blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;
                verticalscanKernel << <blocksPerGrid, threadsPerBlock >> >(acc_cost, N, Lhor, Lver,
    penalty, rowstart, &temp_max, i, j, &bestPredi, &bestPredj);
                cudaMemcpy(temp_acc_cost, acc_cost, size*sizeof(double),
    cudaMemcpyDeviceToHost);
                cudaMemcpy(temp_row_pred, row_pred, size*sizeof(double),
    cudaMemcpyDeviceToHost);
                cudaMemcpy(temp_col_pred, col_pred, size*sizeof(double),
    cudaMemcpyDeviceToHost);

                //Horizontal scan: nodes (i,1),(i,2),...,(i,j-1)
                if ((j - (int)Lhor) >= 1)
                        colstart = j - Lhor;
                else colstart = 1;
                cudaMemcpy(acc_cost, temp_acc_cost, size*sizeof(double),
    cudaMemcpyHostToDevice);
                cudaMemcpy(row_pred, temp_row_pred, size*sizeof(double),
    cudaMemcpyHostToDevice);
                cudaMemcpy(col_pred, temp_col_pred, size*sizeof(double),
    cudaMemcpyHostToDevice);
                blocksPerGrid = (size + threadsPerBlock - 1) / threadsPerBlock;
                horizontalscalscanKernel << <blocksPerGrid, threadsPerBlock >> >(acc_cost, N, Lhor, Lver, penalty, rowstart, colstart, &temp_max, i, j, &bestPredi, &bestPredj);
                cudaMemcpy(temp_acc_cost, acc_cost, size*sizeof(double), cudaMemcpyDeviceToHost);
                cudaMemcpy(temp_row_pred, row_pred, size*sizeof(double), cudaMemcpyDeviceToHost);
                cudaMemcpy(temp_col_pred, col_pred, size*sizeof(double), cudaMemcpyDeviceToHost);

                //Finished (i,j).There only remains to store the winner
                if (temp_max > 0)
                {
                        temp_acc_cost[(i*N) + j] = temp_max;
                        temp_row_pred[(i*N) + j] = bestPredi;
                        temp_col_pred[(i*N) + j] = bestPredj;
                }
            }
        }
        //eo grid processing
        maxv = temp_acc_cost[0];
        for (i = 1; i < size; i++)
        {
            if (temp_acc_cost[i] > maxv)
                maxv = temp_acc_cost[i];
        }
        //printf("gia na doume an ftanei mexri edw \n");
        printf("the similarity is %f \n", maxv);
        clock_t toc = clock();
        printf("Elapsed: %f seconds\n", (double)(toc - tic) / CLOCKS_PER_SEC);
       system("pause");
        // empty the vector
        cudaFree(device_vector1);
        cudaFree(device_pattern);
        free(temp_memory);
        free(temp_pattern);
```

```
  cudaFree(device_vector1);
  cudaFree(device_pattern);
  cudaFree(acc_cost);
  cudaFree(col_pred);
  cudaFree(row_pred);
  free(temp_acc_cost);
  free(temp_col_pred);
  free(temp_row_pred);
          return 0;
}
```

TSL Script

```
#!/usr/bin/expect -f
set timeout 10
set machine kali
set attack <attack string>
set terminator   <terminator string>
system {mkdir -p metriseis/first/unclean}
system {mkdir -p metriseis/first/clean}
system {rm --f /var/log/audit/audit1.log}
system {rm --f metriseis/first/unclean/file1.txt}
system {rm --f metriseis/first/clean/clean_file1.txt}
spawn virsh
expect "virsh #"
send "console $machine\r"
expect "Escape character is ^]\r"
send "\r"
expect "#"
system {rcauditd start}
sleep 2
system {auditctl -e 1}
send attack
expect terminator
system {auditctl -e 0}
sleep 2
```

```
system {rcauditd stop}
system {mv /var/log/audit/audit.log /var/log/audit/audit1.log}
system {sed -n "s/.* \(syscall=[0-9]*\).*/\1/p" /var/log/audit/audit1.log >metriseis/first/unclean/file1.txt}
system {sed 's/syscall=//g' metriseis/first/unclean/file1.txt > metriseis/first/clean/clean_file1.txt}
```

Authenticator Windows Implementation

```
<Window x:Class="First_Step.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Key generator" Height="350" Width="525" Icon="/First_Step;component/Images/key2.png">
  <Window.Background>
    <ImageBrush ImageSource="/First_Step;component/Images/music.jpg" />
  </Window.Background>
  <Grid>
    <Grid.Background>
      <ImageBrush />
    </Grid.Background>
    <Label Background="White" Content="Please insert the input and the output filename for the text encryption or decryption" FontWeight="Bold" Height="25" HorizontalAlignment="Left" Margin="10,10,0,0" Name="label1" VerticalAlignment="Top" Width="494" />
    <Button Content="Encryption" Height="23" HorizontalAlignment="Left" Margin="271,42,0,0" Name="button2" VerticalAlignment="Top" Width="75" Click="button2_Click" />
    <TextBox DataContext="{Binding}" Height="23" HorizontalAlignment="Left" Margin="154,217,0,0" Name="textBox3" VerticalAlignment="Top" Width="120" TextChanged="textBox3_TextChanged" />
    <Label Background="White" Content="Welcome to your key generator! Insert  your user id and password!" FontWeight="Bold" Height="25" HorizontalAlignment="Left" Margin="10,10,0,0" Name="label2" VerticalAlignment="Top" Width="494" />
    <Button Content="Decryption" Height="23" HorizontalAlignment="Left" Margin="19,172,0,0" Name="button5" VerticalAlignment="Top" Width="75" Click="button4_Click" />
    <Button Content="Browse Input File" Height="23" HorizontalAlignment="Right" Margin="0,217,392,0" Name="button6" VerticalAlignment="Top" Width="106" Click="button6_Click" />
    <TextBox DataContext="{Binding}" Height="23" HorizontalAlignment="Left" Margin="99,42,0,0" x:Name="textBox1" VerticalAlignment="Top" Width="137" TextChanged="textBox1_TextChanged" />
    <TextBox DataContext="{Binding}" Height="23" HorizontalAlignment="Left" Margin="99,87,0,0" x:Name="textBox2" VerticalAlignment="Top" Width="137" TextChanged="textBox4_TextChanged" />
```

```
            <Label Background="White" Content="User Id" FontWeight="Bold" Height="25" HorizontalAlignment="Left" Margin="19,40,0,0" x:Name="label2_Copy" VerticalAlignment="Top" Width="56" />
            <Label Background="White" Content="Password" FontWeight="Bold" Height="25" HorizontalAlignment="Left" Margin="19,85,0,0" x:Name="label2_Copy1" VerticalAlignment="Top" Width="68" />
            <Button Content="Steganography" Height="23" HorizontalAlignment="Left" x:Name="button2_Copy" VerticalAlignment="Top" Width="97" Click="button1_Click" Margin="271,85,0,0" />
            <Button Content="Steganalysis" Height="23" HorizontalAlignment="Left" x:Name="button2_Copy1" VerticalAlignment="Top" Width="75" Click="button3_Click" Margin="304,217,0,0" />
    </Grid>
</Window>



using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Diagnostics;
using System.Windows.Forms;

namespace First_Step
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
```

```
{
    string filename;
    public MainWindow()
    {
        InitializeComponent();
    }
    private void textBox4_TextChanged(object sender, TextChangedEventArgs e)
    {

    }

    private void textBox3_TextChanged(object sender, TextChangedEventArgs e)
    {

    }
    private void textBox1_TextChanged(object sender, TextChangedEventArgs e)
    {

    }

    private void button2_Click(object sender, RoutedEventArgs e)
    {
        //creation of passphrase
        int userid = Convert.ToInt32(textBox1.Text);
        int year = DateTime.Today.Year;
        int month = DateTime.Today.Month;
        int day = DateTime.Today.Day;
        int result = (userid + year) % month;
        string charresult = result.ToString();
        string passphrase = charresult + textBox2.Text;
        System.IO.File.WriteAllText(@"code.txt", passphrase);
        //creation of encryption passphrase
        result = (userid + month) % day;
        charresult = result.ToString();
        passphrase = charresult + textBox2.Text;
        Process proc = new Process();
        proc.StartInfo.FileName = @"cryptest.exe";
        proc.StartInfo.Arguments = "e " + "code.txt" + " " + "enc_code.txt " + passphrase;
        proc.Start();
    }

    private void button6_Click(object sender, RoutedEventArgs e)
    {
        //Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
        //System.Windows.Forms.OpenFileDialog dlg = new System.Windows.Forms.OpenFileDialog();
        OpenFileDialog dlg = new OpenFileDialog();
        dlg.Filter = "All files (*.*)|*.*";
        dlg.InitialDirectory = "tests";
        dlg.Title = "Select file to steganalyse";
        dlg.ShowDialog();
        //filename = dlg.FileName.ToString();
        filename = dlg.SafeFileName.ToString();
        textBox3.Text = filename;
        //MessageBox.Show(dlg.FileName.ToString());
        //MessageBox.Show(textBox3.Text);
        }




    private void button4_Click(object sender, RoutedEventArgs e)
    {
        //creation of passphrase
        int userid = Convert.ToInt32(textBox1.Text);
        int year = DateTime.Today.Year;
        int month = DateTime.Today.Month;
        int day = DateTime.Today.Day;
        int result = (userid + month) % day;
        string charresult = result.ToString();
        string passphrase = charresult + textBox2.Text;
```

```
                    //creation of encryption passphrase
                    Process proc = new Process();
                    proc.StartInfo.FileName = @"cryptest.exe";
                    proc.StartInfo.Arguments = "d " + textBox3.Text + " " + "code_decoded.txt " + passphrase;
                    proc.Start();
                }

                private void button1_Click(object sender, RoutedEventArgs e)
                {
                    //creation of passphrase
                    int userid = Convert.ToInt32(textBox1.Text);
                    int year = DateTime.Today.Year;
                    int month = DateTime.Today.Month;
                    int day = DateTime.Today.Day;
                    int result = (userid + day) % month;
                    string charresult = result.ToString();
                    string passphrase = textBox2.Text + charresult;
                    //pick a random number of wav file from 1-20
                    Random rnd = new Random();
                    int number_of_wav = rnd.Next(1, 20);
                    //make it string
                    string wavnumber = number_of_wav.ToString();
                    string song = wavnumber + ".wav";
                    Process proc = new Process();
                    proc.StartInfo.FileName = @"encode.exe";
                    proc.StartInfo.Arguments = "-E " + "enc_code.txt" + " -P " + passphrase + " " + song + " " + wavnumber + "stega.mp3";
                    proc.Start();
                }

                private void button3_Click(object sender, RoutedEventArgs e)
                {
                    //creation of passphrase
                    int userid = Convert.ToInt32(textBox1.Text);
                    int year = DateTime.Today.Year;
```

```
            int month = DateTime.Today.Month;
            int day = DateTime.Today.Day;
            int result = (userid + day) % month;
            string charresult = result.ToString();
            string passphrase = textBox2.Text + charresult;
            //System.Windows.MessageBox.Show(textBox3.Text);
            Process proc = new Process();
            proc.StartInfo.FileName = @"Decode.exe";
            //
            //System.Windows.MessageBox.Show("-X -P " + passphrase + " " + textBox3.Text);
            proc.StartInfo.Arguments = "-X -P " + passphrase + " " + textBox3.Text;
            //string arguments="-X -P " + passphrase + " " + textBox3.Text;
            //System.Windows.MessageBox.Show(arguments);
            //System.Diagnostics.Process.Start("Decode.exe",arguments);
            proc.Start();
        }




    }

}
```