LiveCloudInspector: Towards Integrated IaaS Forensics in the Cloud

Julian Zach and Hans P. Reiser(✉)

University of Passau, Innstr. 43, 94032 Passau, Germany
julian.zach@t-online.de, hr@sec.uni-passau.de

Abstract. Cloud-based systems are becoming an increasingly attractive target for malicious attacks. In IaaS environments, malicious attacks on a cloud customer's virtual machine may affect the customer, who cannot use all diagnostic means that are available in dedicated in-house infrastructures, as well as the cloud provider, due to possible subsequent attacks against the cloud infrastructure and other co-hosted customers. This paper presents an integrated approach towards forensics and incident analysis in IaaS cloud environments. The proposed architecture enables the cloud provider to securely offer forensics services to its customers on a self-service platform. The architecture combines three important analysis techniques and provides significantly better investigation capabilities than existing systems: First, it supports host-based forensics based on virtual machine introspection. Second, it offers live remote capture of network traffic. Third, and most importantly, it provides hybrid combinations of the first two techniques, which enables enhanced analysis capabilities such as support for monitoring encrypted communication.

1 Introduction

1.1 Motivation

The increasing shift of resources towards the cloud makes it necessary to deal with new IT security challenges. As more and more resources are out-sourced into the cloud, these will be a more likely target of malicious activities. Traditional mechanisms for investigating such incidents are, to large extent, insufficient.

A basic problem is the separation between cloud provider and cloud user. In an Infrastructure-as-a-Service (IaaS) cloud, the cloud customer is responsible for all software layers within a virtual machine, but in case of a security incident, the customer cannot apply traditional investigation approaches and tools that require direct access to the physical hardware. On the other hand, the cloud provider has no knowledge about internals of the customer's virtual machines and thus is also in a weak position for an in-depth investigation. A second important challenge is multi-tenancy. Significant efficiency benefits of cloud computing stem from the shared use of resources by multiple customers. A fundamental requirement of cloud infrastructures is the strict separation between multiple tenants using shared physical hardware. An investigation of one customer must not affect the availability, integrity or confidentiality of resources used by other customers.

A. Bessani and S. Bouchenak (Eds.): DAIS 2015, LNCS 9038, pp. 207–220, 2015.
DOI: 10.1007/978-3-319-19129-4_17

1.2 Problem Statement

It is straight-forward to use existing post-incident investigation tools that analyse static main memory snapshots or analyse log files created during system execution in an IaaS cloud environment. Main memory snapshots can efficiently be created from within active virtual machines, and log files can be obtained from the system within the virtual machine as well as from the cloud management system. A fundamental limitations of these approaches, however, is that they enable only static a-posteriori analysis. What is currently missing are appropriate methods for direct *live* investigations on a running system.

In this paper we propose a novel, integrated architecture for live IaaS forensics. The architecture enables the cloud provider to offer forensics services to customers via a secure interface. The architecture, which we implemented in the LiveCloudInspector prototype, makes three important contributions:

- It enables remote host forensics based on virtual machine introspection (VMI) with a self-service interface for customers;
- It enables efficient, transparent remote network forensics in IaaS cloud infrastructures;
- It offers novel analysis capabilities that yield additional insight by combining host and network forensics.

The combined analysis makes important contributions to enhancing the investigation process. Specifically, it enables correlating recorded network traffic with running processes, indicating exactly to which process data has been sent to or received from; it supports dedicated network monitoring of selected processes; and it supports transparent monitoring of encrypted network traffic using VMI-based session key extraction.

This paper is structured as follows: In the next section, we discuss related work. In Section 3, we summarize the network monitoring and virtual-machine introspection mechanisms our work builds upon. Section 4 presents our architecture. Section 5 describes and evaluates our prototype implementation. Finally, we present our conclusions in Section 6.

2 Related Work

The problem of incident investigation in cloud computing environments has gained some attention only in the recent years. Birk et al. [2] state that the ability to perform forensic investigations in the cloud is of high relevance, but seldomly discussed. The authors argue that guidelines and best practices for investigations in the cloud are rare, often outdated, or non-existent. Similarly, Taylor et al. [12] conclude that "currently there do not appear to be any published guidelines that specifically address the conduct of computer forensic investigations of cloud computing systems."

Dykstra and Sherman [3] tried to deploy existing forensics tools such as Guidance EnCase in an IaaS cloud to acquire forensic data remotely over the Internet.

The paper provides an excellent discussion of the limitations of such an approach. In particular, it argues why data acquired that way might not be trustworthy.

Martini and Choo [7] describe a conceptual framework for collecting forensic data from a cloud environment. The paper includes an extensive discussion of related work on digital forensics and cloud forensics on a broader scope than we include here as well as a high-level description of a conceptual framework. To our knowledge, no practical implementation of that framework exists so far.

In practice, the mechanism with most widespread support by existing cloud providers is the export of virtual machine snapshots[1]. Such virtual machine snapshots contain data of the virtual disk, but not live data such as the main memory. They can only be used for off-line forensic investigations. Some providers implement additional features such as Amazon Cloud Watch[2], which provides basic run-time monitoring features that collect various metrics about run-time behaviour, and Amazon Cloud Trail[3], which records AWS API calls and delivers log-files to the customer.

FROST (Forensic OpenStack Tool) [4] has recently been presented as a forensic toolkit within the OpenStack platform. FROST collects data at the cloud provider and host operating system level und makes it available to the customer by additional API methods. These methods allow downloading API logs, firewall logs and retrieving disk images. Similar to our approach, it advocates the idea of integrating forensics tools and interfaces into a cloud platform management infrastructure. What differentiates our work from FROST are enhanced mechanisms for data acquisition and analysis, specifically supporting *live* analysis.

Gebhardt et al. [6] implemented a network forensics tool for the cloud that extends the OpenNebula management platform. This tools allows recording network traffic on demand and delivering network traffic dumps to the customer for further investigation. The LiveCloudInspector includes a very similar approach for network monitoring, but as a main contributions adds additional data acquisition and analysis methods.

Our framework enhances these existing approaches by implementing both network and host forensics based on dynamic run-time introspection in a single integrated platform. The combination of network monitoring with host introspection yields better insights and enables useful additional mechanisms such as automated correlation of network traffic with running processes and automated decryption of encrypted TLS channels based on secret-key extraction.

3 Background

3.1 Virtual Machine Introspection

Virtual machine introspection (VMI) is an established technology in which the virtual machine monitor (VMM) transparently inspects internal data of a running virtual machine. The VMM has full control of all resources of the VM (such

[1] http://aws.amazon.com/ec2/vm-import/ [validated on 2014-09-20]
[2] http://aws.amazon.com/cloudwatch [validated on 2014-09-20]
[3] http://aws.amazon.com/cloudtrail [validated on 2014-09-20]

as main memory, hard disks, and network devices), and thus is in a position for accessing all of them.

Our prototype implementation builds upon the state-of-the-art introspection library LibVMI[4], which supports both Xen and KVM hypervisors. LibVMI requires some knowledge about the OS running within the VM to interpret VM memory correctly. In this paper, we assume that such information exists a priori and can be provided statically to LibVMI. This should, for example, be the case, if the user wants to investigate its own virtual machines. In situations in which such information is not available, our system could be combined with approaches that automatically bridge this semantic gap, such as Insight-VMI [10].

The first goal of our proposed architecture is to enable remote acquisition of memory snapshots using VMI in an IaaS cloud-computing environment.

3.2 High-Level Memory Analysis

While LibVMI offers some low-level API for transparently obtaining data from a running VM, it frequently is desirable to derive more high-level information. Several existing tools such as F-Response[5] and VAD tools[6] support such analysis on the basis of a static main memory snapshot. In our prototype, we use the Volatility framework[7], which can also be combined with live VMI. Volatility has a modular architecture and, for example, includes modules that based on a target system's main memory content enumerate all processes (*pslist*), existing network connections (*connscan*), open files (*filescan*) and registry entries (*hivelist*), or extract sections of main memory of individual processes (*memdump*).

The second goal of our proposed architecture is to enable secure remote use of this tool in a public cloud environment.

3.3 Network Monitoring Background

According to Garfinkel [5], tools for network forensics operate either host-based (such as *Wireshark*[8]) or network-wide (such as NIKSUN NetDetector). They either capture all network packets and store them for later analysis (*"catch-it-as-you-can"*) or analyse packets directly after reception and store only information produced by that analysis for later further processing (*"stop-look-and-listen"*).

Our third goal is to enable remote packet sniffing in a cloud environment, which means that we aim at designing a service that captures all network packets of a virtual machine and stores them for later analysis by the client. For such a host-based catch-it-as-you-can service, we additionally consider continuous monitoring of virtual machines that are migrated and VMI-assisted filtering of recorded traffic.

[4] https://code.google.com/p/vmitools/ [validated 2014-09-20]
[5] https://www.f-response.com/ [validated 2014-10-20]
[6] http://vadtools.sourceforge.net [validated 2014-10-20]
[7] http://www.volatilityfoundation.org [validated 2014-09-20]
[8] https://www.wireshark.org [validated 2014-09-20]

3.4 TLS Decryption Background

Analysis tools such as Wireshark contain protocol dissectors for hundreds of different protocols. Dissectors are able to automatically interpret and display protocol details. Wireshark is able to dissect TLS traffic, and if provided with the RSA private key, it can decrypt messages used for exchanging the RSA-encrypted session key (*RSA key exchange*). After obtaining the session key, all subsequent TLS traffic can easily be decrypted.

Shamir et al. [11] have shown that RSA private keys can be located in main memory dumps based on algebraic properties. Such approach could be used for our system, but only with significant limitations: It works only for incoming connections (otherwise, the session key is sent to the remote host, encrypted with the remote host's public key, and the corresponding private key of the remote host cannot be retrieved locally), and it does not work with other key exchange mechanisms that offer forward secrecy, such as Diffie Hellman and ECDH.

Our goal is to reuse as much as possible of the Wireshark TLS dissector, but enhance it with VMI mechanisms that retrieve the *session key* from main memory. We want to be able to monitor all TLS traffic, both incoming and outgoing, and independent of the key exchange mechanisms used.

4 Design and Architecture

Our main goal is to design a secure system that enables cloud users and authorized third parties to perform investigations on some target VMs as a self service (i.e., without manual support by the cloud provider). In this section, we first present the role model of our approach and the high-level design, followed by a detailed description of workflows for simple and complex analysis tasks.

4.1 Role Model

Various roles can be differentiated in an IaaS cloud. Figure 1 illustrates the roles that we consider. Our design makes the assumption that the cloud provider itself (including its staff) is trustworthy. We do not consider malicious insiders (such as a malicious administrator of the cloud provider). Such cases might be tackled with trusted cloud computing techniques, for example such as presented by Rocha et al. [9], but this is beyond the scope of the present paper. We consider the following attacks against our architecture:

- Unauthorized malicious third parties that try to attack the forensics system from outside;
- Users and external investigators that are authorized to investigate virtual machines of a specific user, but exploit the forensics system to gain access to or harm other users, violating the separation of tenants;
- Attackers that deploy (as a user) their own virtual machines within the cloud infrastructure and use those as a starting point for attacks against the forensics system.

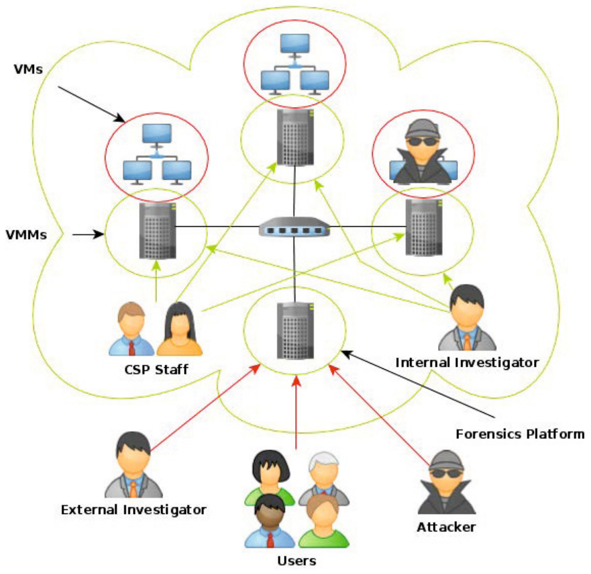

Fig. 1. Trusted and untrusted roles in the LiveCloudInspector architecture: The cloud service provider (CSP) including its staff is trusted, while all other parties (external investigators, users, VMs hosted by the CSP) are untrusted

4.2 Design

Figure 2 shows an overview of the architecture of LiveCloudInspector. It distinguishes between two parts of the forensics system. First, a dedicated *forensics platform* implements all supported workflows as well as the public service interface. Workflows are included for low-level direct main memory dumps of virtual machines, for selected memory forensic operations, and for network forensics. The workflow implementations interact with corresponding counterparts that are deployed as a *forensic remote service* on all cloud hosts.

Our design does not integrate the forensics mechanisms deeply into a cloud management platform such as OpenStack[9] or OpenNebula[10]. Instead we aim at proposing a *portable* architecture that can, with little effort, be reused on multiple cloud platforms or different versions of the same platform.

Nevertheless, there are two dependencies on the cloud management system. First, we do not want to have a separate user management system. Instead, LiveCloudInspector will use a platform-specific adaptor to interact with the cloud management system for user authentication and authorization. Second, we depend on the cloud management system for locating the physical host of target virtual machines and for tracking them on migration operations.

[9] http://www.openstack.org [validated 2014-09-20]
[10] http://opennebula.org [validated 2014-09-20]

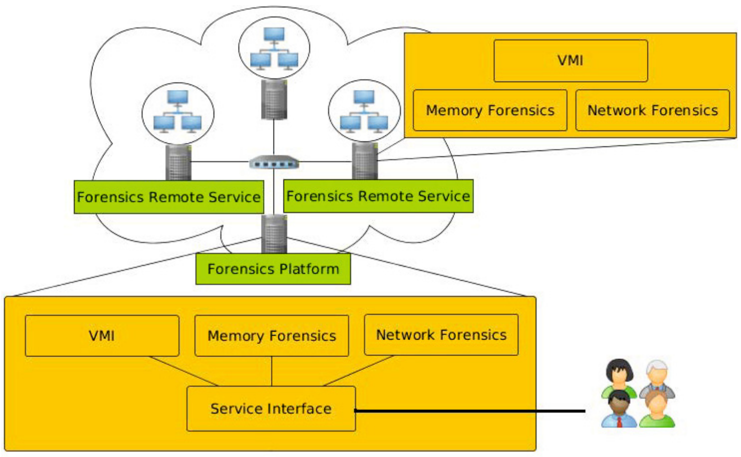

Fig. 2. The LiveCloudInspector architecture adds two components to a cloud environment: A dedicated *Forensics Platform* and decentralized *Forensics Remote Services* deployed on all cloud hosts

The *service interface* of the forensics platform is the public, remotely accessible interface for users and external investigators. After checking the authorization of a client to access the forensics platform via the cloud management system, it accepts commands for *VMI*, *memory forensics* or *network forensics*, retrieves the location of a virtual machine from the platform's management layer, and finally remotely interacts with the corresponding *forensic remote service*.

In the following, we first discuss the details of all *simple workflows*, which are workflows for either recording network traffic or for VMI-based host forensics, followed by a discussion of *complex workflows*, which use a combination of host introspection and network monitoring in order to derive more in-depth insights.

4.3 Simple Workflows

Remote Main Memory Dump. is the first workflow and enables remote snapshots of the *main memory* of a virtual machine. Note that this is fundamentally different from the usual *disk* snapshot generation supported by several cloud providers. Often, relevant artefacts of problems or malicious activities manifest themselves only in main memory, and not on persistent disk storage.

In this workflow, a cloud user or an authorized investigator requests a memory dump at the LiveCloudInspector platform. The platform checks user authentication and authorization and, if access to the specified VM is granted, retrieves the physical location of the VM from the cloud management platform and requests a memory dump from the forensics remote service at the physical host. Finally, the memory dump is made available for download for the cloud user or investigator.

The advantage of this workflow is that the investigator can use any tool of his/her choice to analyse the memory dump. Under the assumption of a trustworthy cloud infrastructure and forensics service, a digital signature attached to the memory dump by the forensics remote service can guarantee the validity of the snapshot. The disadvantage of this workflow is the cost for transferring the memory dump (potentially several gigabytes for large VM instances). A further limitation of the approach is that information in CPU registers or main memory cache is not recorded in the main memory dump.

It should be noted that in fact main memory forensics in a virtual machine is much easier than main memory forensics on a traditional physical host. The main memory of a virtual machine can easily be extracted using introspection tools such as libVMI, whereas the acquisition of main memory content of physical machines requires dedicated hardware or, alternatively, software running on the host that potentially is subject to (unnoticed) alterations.

Remote Memory Forensics enables remote execution of more complex forensics analysis of VM memory. The data source is the same as in the first workflow (i.e., the main memory of a target virtual machine), but instead of transferring the whole memory dump to the investigator, the analysis is performed directly at the target host. For this purpose, such analysis capabilities are enabled directly as a service. In our prototype implementation, we offer remote execution of *Volatility* commands. Volatility supports many high-level diagnostic operations, as briefly discussed in Section 3, and is able to directly interact with VM memory using libVMI.

The advantage of this workflow is that the memory snapshots do not need to be transferred to the investigator. The main disadvantage is that only those remote forensics tools and operations are supported that have been implemented in the forensics platform. We do not support the execution of arbitrary code selected by the user, because this might raise a lot of security questions.

Network Forensics yields additional information for the analysis of anomalous occurrences and intrusions by observing communication patterns. For example, some malware might be periodically communicating with a command-and-control server. Observing network traffic using physical access to the network is an established approach in forensics. In a public IaaS cloud, the investigator has, in most if not all cases, no direct access to physical hosts. Instead, a mechanism for remote acquisition of network traffic is required.

Multi-tenancy potentially raises additional challenges for such remote acquisition, depending on how multi-tenancy is handled by the network infrastructure used by the cloud provider. If each customer (or each VM) has its own, separate virtual network, the traffic of this virtual network can be used for investigation. If multiple tenants share the same local network, additional filtering needs to be applied to the recorded traffic in order to assure strict separation of tenants.

The network monitoring part of the LiveCloudInspector approach is based on previous work by Gebhardt et al. [6]. Unlike the previous work, which integrated the forensics service deeply into OpenNebula, our focus was put on minimizing

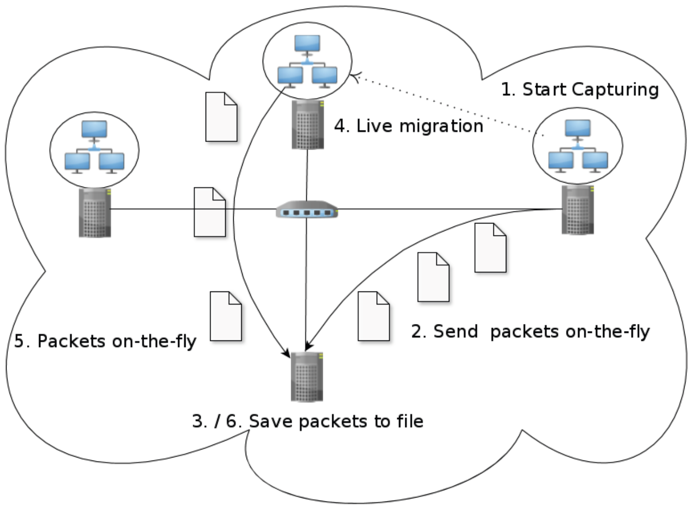

Fig. 3. This image illustrates the high-level interactions between forensics platform (bottom host) and forensics remote service (all other hosts) during VM migration

the dependency on the cloud management system. This makes LiveCloudInspector portable to other cloud management infrastructures.

The basic network capture service implements the recording of network traffic on request by the user. The traffic capture is remotely initiated by remote interaction between forensics platform and forensics remote service. In order to avoid storage overhead on the physical host where the traffic is recorded and also in order to minimize the delay between recording and analysis, each recorded packet is directly sent from the forensics remote service to the forensics platform, where it is made available for download to the investigator.

Virtual Machine Migration is handled by an extension of the network forensics workflow. A cloud management platform may perform live VM migration for purposes such as load balancing or maintenance operations. Many VM managers support not only cold migration (shutting down a VM and relaunching it on a different host), but also hot (live) migration (moving the VM to a different host without shut-down or interruption of client connections).

The LiveCloudInspector supports appropriate coordination mechanisms for handling continuous network capturing during VM migration, as illustrated by Figure 3. For this purpose, it is necessary to receive pre-migration and post-migration events from the cloud management platform, which enable a coordinated activation of traffic acquisition at the migration target and deactivation at the migration source after finishing the migration. Traffic from both the old and the new location are collected by the Forensics Platform and presented to the investigator as a single network capture.

4.4 Complex Workflows

One of the most significant advantages of a hybrid network and host forensics approach is that data of both sources can be combined, which yields several benefits.

Process-Specific Network Monitoring for a single process running within a virtual machine is a first example of a complex workflow. A problem of remote network forensics is that potentially large amounts of recorded data (the whole traffic of a virtual machine) need to be transferred to the investigator.

The LiveCloudInspector allows the investigator to filter the recorded traffic according to running processes within the VM. In order to achieve this, information (IP address and TCP/UDP port numbers) from the captured network is correlated with data about network connections and processes acquired by virtual-machine memory introspection. While this is similar to network forensics workflow, it has the big advantage of requiring less data to be transferred to the investigator.

Correlating Process Names with Network Traffic is a workflow that can be used for monitoring all network communication and correlating each connection with information about the communicating process on the local machine. This might yield only little benefit for incoming connections towards the virtual machine, as usually the target process will be uniquely identified by the destination port and target ports other than the intended services will likely be blocked by a firewall.

Information about corresponding processes can be of valuable benefit for outgoing connections (i.e., originating at the virtual machine under investigation). In this cases, local port numbers usually do not reveal any information about the process the connection originates from.

Monitoring Encrypted TLS Communication is a possibly even more interesting benefit of the combination of host and network forensics. The basic idea is that the session key of a TLS session can directly be extracted using virtual-machine introspection, and then later be used to decrypt all encrypted TLS communication.

The TLS decrypter of LiveCloudInspector can be activated by the network monitor. If a TLS connection is detected, we need to wait for the right point in time for starting the key search. The session key has been calculated by the TLS implementation after the initial TLS handshake has finished and encryption is started using a *ChangeCipherSpec* message. At this point in time, a main memory snapshot of the communicating process within the VM is created and the search for the session key is started concurrent to continuously recording the encrypted traffic.

5 Implementation and Evaluation

5.1 Implementation

We have implemented the proposed architecture in the LiveCloudInspector prototype.

This prototype is designed for working with the OpenNebula cloud management infrastructure. We tried to avoid strong dependency on a specific cloud management product, so we expect that it is easy to port our prototype to other systems. The user interface of LiveCloudInspector is not integrated internally into OpenNebula, but instead a separate web-based interface was created. For this purpose, a front-end running as Java Server Pages on an Apache Tomcat application server were implemented. The front end enables the user to activate several backends for low-level VMI access, for high-level memory introspection, and for network forensics.

The user interface interacts with OpenNebula for authenticating users. The authorization to access virtual machines via LiveCloudInspector is thus not handled with a separate user management system, instead the internal user management of OpenNebula is used. For this purposes we implemented a *Custom Realm* for Tomcat[11] that forwards a user authentication request to OpenNebula authentication core.

A forensics remote service implemented in Java is executed on each cloud host. This service offers a remote interface for interaction with the forensics server. We used SIMON Remote[12] for implementing calls from the server to the remote service, as it is easier to deploy than standard Java RMI. The remote service uses TLS with additional client authentication to make sure that only the forensics server can interact with the remote service.

Our current prototype for the TLS decryption assumes that the session key is directly stored in main memory as a byte array. So far, we successfully validated our key extractor with OpenSSL, JSSE, GnuTLS and Microsoft Schannel implementations. We directly use the decryption functionality of the Wireshark TLS dissector, and thus our implementation works with all TLS ciphers supported by Wireshark.

5.2 Rootkit Case Study

As a first use case we considered a virtual machine infected by the Linux rootkit KBeast. This rootkit is installed as a kernel module and implements features such as key logging and a remote backdoor. The rootkit takes various measures for hiding itself from the user. For example, it hides itself from commands listing the loaded kernel modules (such as *lsmod*), it hides the backdoor process from the process list, and hides network connections of the backdoor from tools such as *netstat*. It is, thus, difficult to detect the rootkit by tools running *within* the virtual machine.

[11] http://tomcat.apache.org/tomcat-7.0-doc/realm-howto.html [validated 2014-10-20]
[12] http://dev.root1.de/projects/simon [validated 2014-10-20]

Using our LiveCloudInspector prototype, we were able to remotely execute volatility functions, e.g., to extract information about kernel modules, network connections, and process lists via VM introspection. As the rootkit is not able to manipulate the information extracted using VMI, we were immediately able to distinguish between an infected and a correct virtual machine.

5.3 Security Evaluation

Our IaaS forensics architecture aims at enhancing the security of cloud computing by enhancing the capabilities for incident analysis, but it also represents an additional component that increases the system's complexity and attack surface and possibly causes additional security risks. We therefore discuss protection mechanisms included in our architecture against the threads described in Section 4.1.

Unauthorized malicious third parties that try to attack the forensics system from outside can interact with the service interface of the forensics platform. Our prototype implements client authentication by delegation to the OpenNebula platform. Assuming a correct implementation of the authorization system in the forensics platform, only users that can access the cloud management platform can access the forensics platform.

Clients that are valid users and thus successfully authenticate to the forensics platform might try to acquire information about other users, violating the strict separation of tenants. In our system, for both network and host forensics, OpenNebula's VM-ID is used to identify the target virtual machine. For all operations, OpenNebula is contacted for all access to that VM-ID, and OpenNebula checks the user's authorization. Only users that are authorized to access a virtual machine via the OpenNebula API are allowed to access it via the forensics platform.

User input is also used in the interaction between the forensics platform's backends and the forensic remote services. In particular, for remote memory forensics, the user has a lot of control over arguments passed to the Volatility tool. Careful input validation is necessary in order to avoid possible injection attacks at this interface.

The *Forensic Remote Service* can also be a target of malicious attacks, originating either from outside or from a virtual machine within the cloud infrastructure. Besides blocking such interaction at the network level using firewall rules we additionally implemented two-way authentication between the *Forensics Platform* and *Forensics Remote Services*, making this kind of attack infeasible.

5.4 Limitations

A frequently discussed limitation of introspection-based tools for malware analysis is split-personality malware. Examples such as RedPill and variants [8] show that it is easy for malware to detect whether it is running in a virtualized environment and thus could behave differently than in a production environment.

This issue is not a problem for our approach, as we apply VMI directly in the production environment.

What is a potential problem are attempts to subvert VMI, as shown for example by DKSM [1]. Our current prototype implementation is based on LibVMI, which relies on the assumption that the provided kernel system map corresponds to the actual kernel in the VM. If the layout of kernel data structures is altered by malware within the virtual machine, VMI can possibly produce wrong data. This is not a direct limitation of our architecture itself, but is an implication of the introspection mechanism in use, and the investigator using our system needs to be aware of this potential problem. The development of more robust introspection solutions that are not vulnerable to such attacks will help to remedy this limitation.

Our monitoring approach for encrypted communication works for traffic that uses standard implementations of TLS. We assume that session keys are stored in main memory (which all popular implementations of TLS that we are aware of do), and that traffic is encrypted according to the specification of TLS. This is most likely true for all software intentionally running within the virtual machine (this is under full control of the user of the VM). Often, even malware uses standard TLS for communication on command-and-control channels, but this could easily be replaced by some other proprietary encryption mechanisms. Our prototype will not be able to decrypt such connections that use encryption methods different to standard TLS.

6 Conclusions

In this paper, we have presented the design and implementation of LiveCloudInspector. LiveCloudInspector enables forensics as a service in public cloud environments. The architecture combines, in a single system, three mechanisms that support the analysis of security incidents:

– It enables transparent live host forensics based on virtual-machine introspection. This feature includes both the possibility of acquiring low-level memory snapshots and the possibility of analysing the running system with high-level Volatility commands.
– It enables remote network forensics by implementing a live network traffic capture mechanism. This mechanism makes sure that captured data is filtered correctly in a multi-tenant environment, and it also supports virtual machine migration during the capture process.
– It combines network monitoring with VMI-based host analysis. This yields interesting new analysis capabilities, such as directly monitoring traffic of specific processes, correlating information about processes with network traffic, and secret key extracting for monitoring encrypted TLS communication. The session key extraction works for both incoming and outgoing connections, it even supports channels established with perfect forward secrecy (i.e., Diffie-Helman based session key establishment), and unlike interception-based approaches it is fully transparent for the communication endpoints.

With these contributions, our architecture enhances the possibilities for investigating security incidents in infrastructure-as-a-service cloud environments.

Acknowledgments. The research leading to these results was supported by the "Bavarian State Ministry of Education, Science and the Arts" as part of the FORSEC research association.

References

1. Bahram, S., Jiang, X., Wang, Z., Grace, M., Li, J., Srinivasan, D., Rhee, J., Xu, D.: DKSM: Subverting virtual machine introspection for fun and profit. In: 29th IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 82–91 (October 2010)
2. Birk, D., Wegener, C.: Technical issues of forensic investigations in cloud computing environments. In: Proceedings of the 2011 Sixth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering, SADFE 2011, pp. 1–10. IEEE Computer Society, Washington, DC (2011)
3. Dykstra, J., Sherman, A.T.: Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. Digital Investigation 9, 90–98 (2012)
4. Dykstra, J., Sherman, A.T.: Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform. Digit. Investig. 10, 87–95 (2013)
5. Garfinkel, S.: Network forensics: Tapping the internet, http://www.oreillynet.com/pub/a/network/2002/04/26/nettap.html (April 01, 2015)
6. Gebhardt, T., Reiser, H.P.: Network forensics for cloud computing. In: Dowling, J., Taïani, F. (eds.) DAIS 2013. LNCS, vol. 7891, pp. 29–42. Springer, Heidelberg (2013)
7. Martini, B., Choo, K.R.: An integrated conceptual digital forensic framework for cloud computing. Digital Investigation 9(2), 71–80 (2012)
8. Paleari, R., Martignoni, L., Roglia, G.F., Bruschi, D.: A fistful of red-pills: How to automatically generate procedures to detect cpu emulators. In: Proceedings of the 3rd USENIX Conference on Offensive Technologies, WOOT 2009. USENIX Association, Berkeley (2009)
9. Rocha, F., Abreu, S., Correia, M.: The final frontier: Confidentiality and privacy in the cloud. Computer 44(9), 44–50 (2011)
10. Schneider, C., Pfoh, J., Eckert, C.: A universal semantic bridge for virtual machine introspection. In: Jajodia, S., Mazumdar, C. (eds.) ICISS 2011. LNCS, vol. 7093, pp. 370–373. Springer, Heidelberg (2011)
11. Shamir, A., van Someren, N.: Playing "hide and seek" with stored keys. In: Franklin, M.K. (ed.) FC 1999. LNCS, vol. 1648, pp. 118–124. Springer, Heidelberg (1999)
12. Taylor, M., Haggerty, J., Gresty, D., Lamb, D.: Forensic investigation of cloud computing systems. Netw. Secur. 2011(3), 4–10 (2011)