2017 13th European Dependable Computing Conference

An Analysis of OpenStack Vulnerabilities

Ivano Alessandro Elia, Nuno Antunes, Nuno Laranjeiro, Marco Vieira
CISUC, Department of Informatics Engineering
University of Coimbra
Polo II, 3030-290 Coimbra – Portugal
ivanoe@dei.uc.pt, nmsa@dei.uc.pt, cnl@dei.uc.pt, mvieira@dei.uc.pt

***Abstract*—Cloud management frameworks provide an effective way to deploy and manage the hardware, storage and network resources for supporting critical cloud infrastructures. OpenStack is used in the context of business critical systems and frequently deals with highly sensitive resources, where a security breach may result in severe damage, including information theft or financial losses. Despite this, there is little information on how much security is a concern during design and implementation of OpenStack components. This work analyses 5 years of security reports on OpenStack and the corresponding patches, with the goal of characterizing the most frequent vulnerabilities, how they can be exploited, and their root causes. The goal is to identify vulnerability trends, characterize frequent threats, and shed some light on the overall security of OpenStack. Special focus is placed on the framework component for virtualization management (Nova), by also analyzing the code of the available patches. Overall results show a preponderance of vulnerabilities that may be exploited to cause DoS and expose sensitive information. Also, 2/3 of the total number of vulnerabilities can be exploited by insider attacks, urging administrators to focus protection efforts on them. Finally, many bugs remain undetected for long periods when most of them are easy to avoid or detect and correct.**

***Index Terms*—Cloud, Security, Vulnerabilities, OpenStack**

I. Introduction

Cloud Management Frameworks (CMFs), such as OpenStack [1], OpenNebula [2] and CloudStack [3], simplify the deployment of cloud infrastructures, particularly in managing hardware pools, storage, and network resources. OpenStack is an open source framework that dominates the market, reaching 69% of adoption rate in some studies [4], being used by many large companies like PayPal, Intel, American Express, Disney, and Walmart, among others [5].

The **adoption of cloud solutions introduces security and privacy concerns** due to the relocation of the resources and data to third-party infrastructures. OpenStack is no exception, and security concerns are present in its lifecycle, as referred in the OpenStack security guide [6], which provides best practices, recommendations and guidelines about securing an OpenStack cloud. The document focuses on threats regarding incorrect or insecure deployment of the cloud systems, components and assets that can be deployed and managed via OpenStack [6]. Conversely, very little attention is given to the possibility of attacks directly targeting the OpenStack platform, rather than the cloud infrastructures managed by it.

In practice, **although OpenStack deals with sensitive resources and is used in business critical scenarios, there is little information regarding how much we can trust on its security**. In fact, despite the numerous vulnerabilities reported over the years, the documentation only briefly addresses the issue [6]. Past works analyzed OpenStack security. However, these studies are in general very superficial, focused only on the features supported and their limitations [7], [8], or in the security of the network configuration [9]. We argue that a deeper analysis is necessary to understand the OpenStack platform from a security perspective.

The analysis of software bugs and vulnerabilities based on repositories is a common and effective strategy [10], [11], [12], [13]. The Common Vulnerabilities and Exposures (CVE) is a public on-line database aggregating the information for the publicly confirmed vulnerabilities, being the *de facto* standard platform for identifying and referencing vulnerabilities. Further information is available in CVEDetails [14]. This is an important source of knowledge about vulnerabilities and respective patches, with large amounts of information that can be analyzed to extract relevant insights. In practice, this analysis may provide answers to the following questions:

- *Which are the most common characteristics of the vulnerabilities affecting the OpenStack platform?*
- *What type of bugs / coding problems are most commonly causing a vulnerability?*
- *What strategies were used to correct the vulnerabilities?*

In this work we present an **analysis of 5 years of OpenStack security reports and patches** and try to understand and characterize the security threats associated with the framework. In practice, we characterize the reported vulnerabilities, identify trends (e.g., vulnerability type, presence across versions) and, for the platform's main component "Nova", study their exploitation through attacks and analyze patches, with the expectation of providing evidences regarding OpenStack's trustworthiness from a security point of view. Such knowledge is of utmost importance not only for cloud providers offering services based on OpenStack, but also for users that intend to contract their services.

Our work may contribute to improve the security of the OpenStack platform by: *i)* providing the developers with systematized information that **prevent** the same errors from happening again; *ii)* helping to foresee the possible weak spots of the platform associated with other currently unidentified vulnerabilities, thus suggesting a **prioritization** for the deployment of defense mechanisms [15]; and *iii)* supporting the development of vulnerability and attack injection techniques for the systematic evaluation of security mechanisms [12].

The results show that 2/3 of the analyzed vulnerabilities require a certain level of access to the platform to be exploited, suggesting that developers may be underestimating the threat represented by internal users. We also discovered that although the majority of vulnerabilities are uncovered within 6 months from their introduction **at least 20% take more than one year to be detected**. The analysis also shows that most vulnerabilities could have been detected with tests that analyze the behavior of the software under situations not foreseen by the developers, e.g. using *stress* or *robustness testing* [16].

The structure of the paper is as follows. Section II presents the methodology followed and Section III presents an overview of the results. Section IV presents a characterization of the vulnerabilities of the Compute (Nova), while Section V discusses the patches for those vulnerabilities. Finally, Section VI systematizes the lessons learned and concludes the paper.

II. Vulnerability Analysis Methodology

The OpenStack community relies on CVE to provide a common reference to all confirmed vulnerabilities. Between 2011 and 2016 there are `162` entries regarding the OpenStack platform, each one corresponding to a vulnerability in one of the OpenStack components. The CVE entries aggregate different types of information, including a textual description for each vulnerability, a set of links related to the discovery of that vulnerability, and the date of creation of the entry.

For the analysis of the security reports, we defined a methodology based on three key phases, as portrayed in Fig. 1. The following paragraphs detail these phases.

As the information available has a few inconsistencies [14], **Phase 1)** is a data preparation phase focused on addressing manually the following issues:

- **Diverse denominations of the components.** The vulnerable components are erroneously referred using multiple names. We classified vulnerabilities using unique names comprising all the vulnerabilities affecting each component, e.g. "Compute", "nova" and "Compute (Nova) Essex" correspond to "Compute (Nova)";
- **Mismatching features of the CVE entry.** Some CVE descriptions state that the vulnerability can be exploited by a "remote authenticated user", but the authentication field of the CVE entry is set to "not required". This was fixed using additional information from the description;
- **Wrong version information.** CVE entries include generic information about the versions of the components affected by each vulnerability (e.g. "Nova before 2015.1.3 (kilo)"). CVE Details extracts this information from the textual description, but in some cases the data is inaccurate. Also fixed using additional description information;

In **Phase 2)** we analyzed the cleaned data of the CVE reports [14] to systematize the `162` OpenStack vulnerabilities. In practice, the vulnerabilities were grouped by type, by affected modules, by source of the threat and by exposure time. The objective was to understand which types of vulnerabilities are most frequent, which are the OpenStack modules that are most affected, and how much time it took be fixed. The results of this analysis are presented in Section III.

In **Phase 3)** we performed an in depth manual analysis of a subset of the vulnerabilities. More specifically, we focused on the 36 vulnerabilities of the Nova component. This choice was dictated by the fact that this is the largest component of the framework in terms of lines of code and also the one with most vulnerabilities reported. Moreover, it is a component that performs one of the most important and sensitive tasks in a IaaS cloud environment: the management of the virtual computational instances. We analyzed all the documents referenced in the CVE entry, including the developers' discussions on internal message boards, and different bug tracking platforms used by the community, like launchpad (https://bugs.launchpad.net) and OSSA (https://security.openstack.org/ossa). This phase was divided in two steps, as follows.

Step 3.A) focused on the analysis of the vulnerabilities to understand how they were discovered, how can they be exploited, and what caused the vulnerability in the first place. The resulting characterization, based on a branching diagram to systematize the vulnerabilities, is presented in Section IV.

Finally, in **Step 3.B)** we examined the code of the provided patches and respective commit messages for each vulnerability in the *Nova* component. The analysis is presented in Section V, together with selected examples of patches for illustrative purposes. In practice, this patch-based analysis provides a more objective understanding of the vulnerabilities, as it happens in defect classification analysis [17].

III. OpenStack Vulnerabilities Systematization

In this section we systematize the vulnerabilities by type, by affected component, by attack source, and by exposure time.

A. Analysis by Vulnerability Type

In terms of type of vulnerabilities, `122` out of `162` CVE have a classification [14], Fig. 2 presents their distribution.

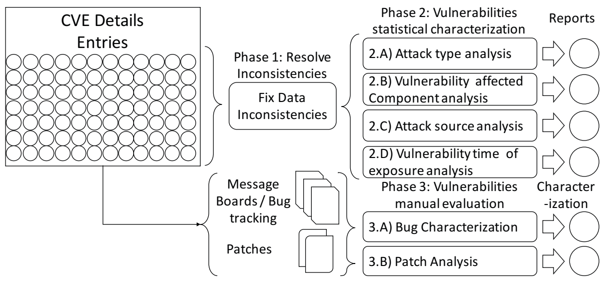
CVE Details
Entries
Phase 1: Resolve Inconsistencies
Fix Data Inconsistencies
Phase 2: Vulnerabilities statistical characterization
Reports
2.A) Attack type analysis
2.B) Vulnerability affected Component analysis
2.C) Attack source analysis
2.D) Vulnerability time of exposure analysis
Message Boards / Bug tracking
Patches
Phase 3: Vulnerabilities manual evaluation
Character -ization
3.A) Bug Characterization
3.B) Patch Analysis

Fig. 1. Schematic representation of the analysis methodology.

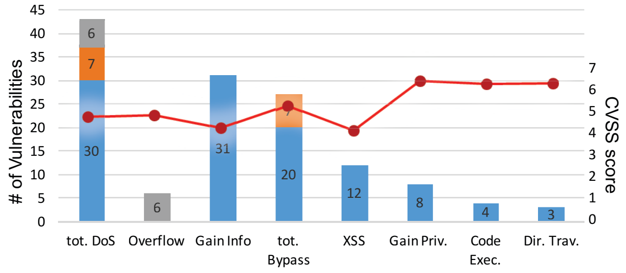

Fig. 2. Distribution by vulnerability type along with average CVSS score.

The analysis shows a prevalence of *Denial of Service (DoS)* related threats (43). Although these vulnerabilities are not as dangerous as privilege escalation or code executions, they affect the availability of the virtualized systems that may be a vital issue for some IaaS platforms (e.g. commercial deployments). The second most common type of vulnerability allows attackers to *gain information* (31). The presence of these vulnerabilities can allow critical problems not only because they tamper with data confidentiality, but they also lead to much more dangerous attacks (i.e. gaining access to users credentials allows other types of attacks).

It is worth noting that a few vulnerabilities are classified in two categories, as highlighted in orange (DoS and Bypass) and gray (DoS and Overflow), as their exploitation leads to both consequences.

B. Analysis by Affected Component

The OpenStack platform is composed of several components, thus it is important to understand which are the most affected ones. Fig. 3.(a) shows the number of vulnerabilities for each of the main components of the OpenStack platform.

As we can see, Nova is the most affected component, with 36 known vulnerabilities. This can be explained by several reasons: it is the biggest component of the project, in terms of lines of code, and it deals with some of the most critical resources (virtual machines). Identity (Keystone) is the second and it is also expected as it provides authentication and authorization services, dealing with delicate functions of the platform where coding errors are more prone to result in vulnerabilities. At the other end of the spectrum, only 4 vulnerabilities were reported for the Orchestration (Heat) component (this is a component that is not very exposed to external attacks).

Fig. 3.(b) shows the distribution of each vulnerability type in terms of components. For DoS, Gain Information and Directory traversal vulnerabilities the most affected component is *Nova*. Also unsurprisingly, most frequent vulnerabilities for *Keystone*, which is in charge of authentication, involve Bypassing security protections and Gaining Privileges.

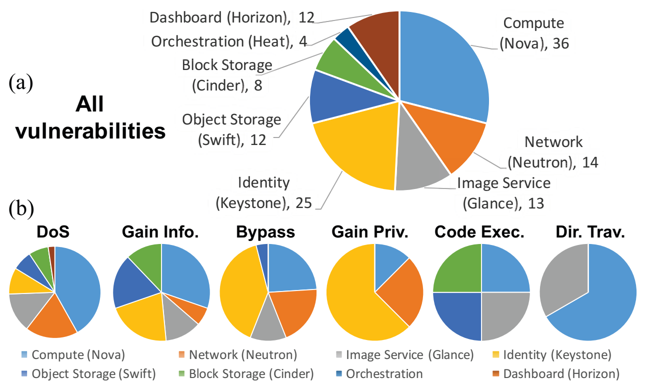

Fig. 3. Distribution of vulnerabilities by affected components.

C. Analysis by Attack Source

It is also necessary to study the sources of the attacks associated with each vulnerability to understand which are the main threats to the security of the platform. In the majority of the CVE entries (≈89%) use three main keywords to describe the possible source of an attack:

- **Remote authenticated user** – remote attacker that has an account in the OpenStack platform (e.g. a cloud tenant);
- **Remote attacker** – attacker that may be located remotely, but does not need to have an account on the platform;
- **Local users** – attacker that does not need to be authenticated in the platform, but needs access to the machine running it (e.g. to read sensitive information leaked);

Fig. 4 represents the distribution of the attack sources, according to (a) Access, (b) Authentication, and whether it is an (c) insider threat. As we can observe in Fig. 4.(a) more than half of the vulnerabilities are exploitable by "remote authenticated users", while the vulnerabilities exploitable by "remote attackers" account for more than one quarter of the total. Also, almost all the vulnerabilities using other keywords for identifying the attackers can be exploited remotely. In practice, these numbers show that the great majority of the vulnerabilities can be exploited remotely (≈87%). Conversely only 13% of the vulnerabilities can only be exploited locally.

An important finding of our analysis is the prevalence of vulnerabilities that can only be exploited by someone with access to the platform (Fig. 4.(c)). Insider threats have been acknowledged since the early 80s [18] and different definitions have been given: "typically attributed to legitimate users who maliciously leverage their system privileges, and familiarity and proximity to their computational environment to compromise valuable information or inflict damage" [18]; "a trusted entity that is given the power to violate one or more rules in a given security policy [...] the insider threat occurs when a trusted entity abuses that power." [19].

In [20] these definitions are applied to the cloud environment identifying two categories: (1) insider threat in the cloud provider, identified as the administrators of the machines running the cloud infrastructure which matches the attackers identified as "local users" (11%) that have legitimate access to machines running OpenStack; (2) Insider Threat in the Cloud Outsourcer where the insider belongs to the organization that uses the services provided by the cloud deployment which matches the definition of "remote authenticated users" (50%) that are users and tenants of the cloud system. According to

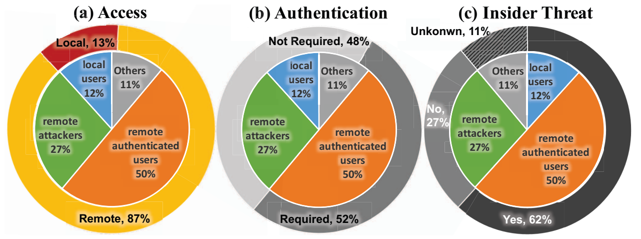

Fig. 4. Vulnerabilities according to the exposure to source of attacks

these definitions, we observe that almost 2/3 of the known OpenStack vulnerabilities are constituted of insider threats.

D. Analysis by Exposure Time

The present analysis covers vulnerabilities involving different components, 14 major releases and numerous component versions over 6 years of development. In this section we analyze the vulnerability exposure time, which is the time between the introduction of a vulnerability (i.e. the release date of the first version that is affected by it) and its correction (i.e. the date of the issuing of the patch). This is the time that the vulnerability is present in the OpenStack official releases and exposed to attackers.

Unfortunately in the CVE entries information about the components versions affected by each vulnerability is expressed in natural language and is not always very clear or accurate and sometimes not available at all. Only in around 2/3 of the cases (109/162) we were able to extract confirmed information about affected versions. In all the cases where the information about which versions of a component were affected by a specific vulnerability was not completely clear we always favored the most conservative hypothesis making our results a lower bound of the real exposure time.

The histogram in Fig. 5 illustrates the data about the vulnerability exposure time. As we can see only 40% of the vulnerabilities are discovered within a 6 months of their introduction. Moreover more than 20% of the vulnerabilities have been exposed for more than one year with a few cases taking more the 2 years. The fact that the vulnerabilities can be exposed for such extents of time increases the chances for them to be discovered (and exploited) by attackers before the developers. It is also safe to assume that many yet to be discovered vulnerabilities are already present in the current version of OpenStack.

IV. Detailed Analysis of Nova Vulnerabilities

This section presents the classification of the 36 *Nova* vulnerabilities, as depicted in Fig. 6. The leaves of the branching diagram represent the **causes** of the vulnerability (i.e. the bugs that allow the malicious activities). The middle tier presents the **effects** of exploiting the vulnerabilities, and the roots represent the **consequences** of such exploitation. Each node displays the respective number of instances. In some cases, one cause may lead to different effects and therefore be associated with more than one type of consequence. The following subsections present a description of each branch, characterizing a group of similar vulnerabilities and discussing some examples of each group. The classification has been performed based on all the information available in the CVE entries (textual description, links to developer message board discussion, patches).

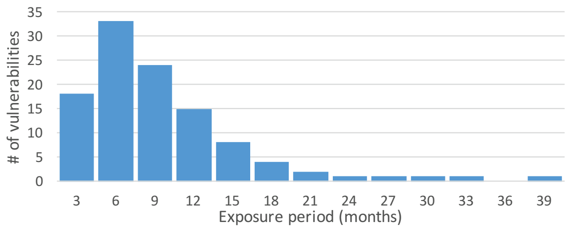

Fig. 5. Number of months the vulnerabilities were exposed in official releases.

A. DoS

The characteristics of vulnerabilities allow us to group them in two groups, according to how the DoS is caused, as follows.

Performance Degradation – In some cases the implementation of an **inefficient information gathering** mechanisms, that unjustifiably consumed an excessive amount of resource resulted in a performance degradation. Other cases are originated by a **vulnerable dependency**, i.e. an external service, library or component.

> ***Load and stress testing** can reveal the system inability to perform the task under the submitted load. Static analyses are ineffective, as the code is functionally correct and issues are only visible during system execution. Issues in OTS components are common and their validation requires effort. Wrapping helps in cases of input/output validation. Security dictates that OTS usage should be minimized, especially new and untested solutions.*

Resources Exhaustion – It can be caused by **unmoderated access to resources** when Nova fails to control access to a limited resource, or when it fails to correctly account the usage of a resource (**unaccounted resources**). It can also be due to unforeseen **concurrent actions**. In several observed cases, if a user issues a delete of a VM undergoing a resize or migration operation, the system loses track of the image and is unable to delete it. Finally, it can be caused when the system fails to correctly account the usage of a resource (**unaccounted resources**). E.g. Nova did not account the virtual disk size in the users disk quota. Thus, when users were allowed to exceed their quota and eventually use all the available disk space.

> *These errors originate from the lack of precaution toward users. From a security perspective, the developers should always use a principle of caution, and consider that any input is possibly malicious. Failing to account for resources poses a problem that is not specifically security-related but rather a functional problem that can be exploited intentionally in the context of a malicious activity. Correctly handling concurrent accesses is a well-known issue. These issues can only be foreseen with a clear view of the use cases and interaction between different tasks.*

B. Bypass

Bypass vulnerabilities are those that involve the circumvention of some type of restriction.

Resources Exhaustion – These vulnerabilities are classified under both Bypass and DoS categories and are similar to the other DoS vulnerabilities described before. In these cases a neglected **concurrent access** issue allows the attackers to bypass a quota restriction that ultimately results in a DoS.

Security restriction violation – These issues are associated with network security groups rules maintenance. These rules are defined by users to control incoming and outgoing network traffic and can be updated at runtime. In the analyzed

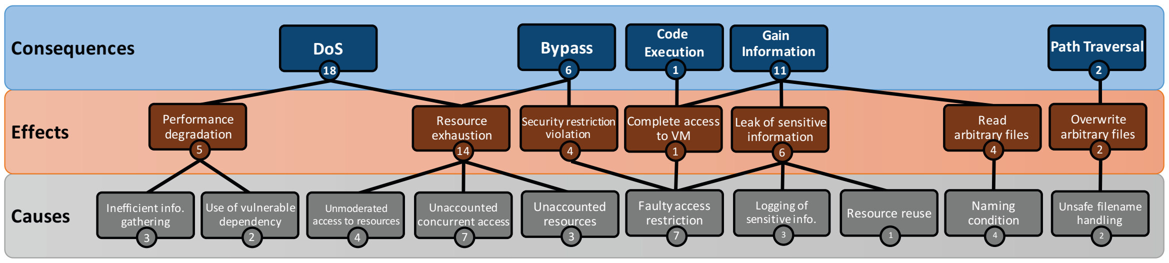
Consequences
DoS 18
Bypass 6
Code Execution 1
Gain Information 11
Path Traversal 2
Effects
Performance degradation 5
Resource exhaustion 14
Security restriction violation 4
Complete access to VM 1
Leak of sensitive information 6
Read arbitrary files 4
Overwrite arbitrary files 2
Causes
Inefficient info. gathering 3
Use of vulnerable dependency 2
Unmoderated access to resources 4
Unaccounted concurrent access 7
Unaccounted resources 3
Faulty access restriction 7
Logging of sensitive info. 3
Resource reuse 1
Naming condition 4
Unsafe filename handling 2

Fig. 6. Security bugs characterization according to common characteristics in terms of root causes, effects and consequences in the security properties.

vulnerabilities under certain circumstances the rules execution or update failed due to a bug. This could be leveraged to bypass such **faulty access restriction** mechanism.

Enforcement and maintenance of security rules, such as network security group rules, is of great importance as their usage may prevent other type of attacks by restricting network access to adversaries. Thus, special care should be posed in enforcing such rules.

C. Gain Information & Code Execution

These vulnerabilities are associated to the disclosure of (or access to) sensitive information or data.

Leak of Sensitive Information – These vulnerabilities could allow attackers to access sensitive information, including VM metadata and credentials. Under certain circumstances (i.e. upon an error) data handled by the system is unduly logged, exposing the data (**logging of sensitive information**). In one case the leaked data included user credentials allowing the attacker to log in and gain **complete access to a VM** of a tenant (Code Execution in Fig. 6).

Developers should always use an extra care when handling sensitive data that should always be filtered when generating an output. The case of users credentials is one of the utmost concern because the leakage of such information may lead to other type of attacks.

The information leek can also be caused by a **faulty access mechanism**. The encountered bugs involved problems like spoofing, accessing a VM by guessing its ID, and brute force attacks (guessing the authorization tokens could be facilitated by measuring response times of failed attempts).

These are among the most classic mistakes in the implementation of security mechanisms. Consulting security experts could have prevented such well-known errors.

Finally, in one case the sensitive information leakage was caused by **reusing a resource** (i.e. physical disk) without wiping the data of the previous user of the same resource.

Read Arbitrary files – these problems involve a violation of the read access rights to files from other users. In the examined cases, these problems were usually caused by a faulty resource tracking mechanisms originated by a complex **naming condition** that involved creation of crafted headers and manipulation of file names that finally resulted in the possibility to read arbitrary files.

These vulnerabilities are quite difficult to detect as they involve complex interactions between different parts of the system and need more than one step to be exploited.

D. Path Traversal

The attackers uses characters as "`../`" to tamper paths and reach files or directories he should not be allowed to access.

Overwrite arbitrary files – These vulnerabilities were caused by **unsafe handling of filenames** and paths, which allowed the modification of legitimate paths in order to read and overwrite arbitrary files of other tenants.

Path traversal is a well-known problem that can be solved by input validation. Although often overlooked, it may provide relevant information to attackers.

V. Patch Analysis

This section provides an overview of the *Nova* patches analyzed in our study. We highlight the main strategies adopted to fix the coding errors described in the previous section and in a selected set of cases we provide concrete examples.

Our analysis revealed that in the majority of the cases the patches are quite small, affecting only a limited portion of code and involving a limited number of files. Fig. 7 presents two examples. Fig. 7.(a) shows CVE-2013-2096, a case of **DoS** vulnerability where the images virtual size was **not accounted** allowing malicious users to saturate the disk by creating images with a large virtual size (**resource exhaustion**). The patch updates only a single file adding 8 lines for the missing virtual size check upon image creation.

The majority of the security patches examined are quite straightforward. For instance, Fig. 7.(b) shows the patch for a **path traversal** vulnerability where an **unsafe filename handling** in the original version permitted "`..`" to be contained in the path thus allowing the attackers to **overwrite arbitrary files**. The patch simply forbids it by raising an exception.

As in the presented examples, the majority of the vulnerabilities were due to simple oversights on the programmers side involving issues like erroneous accounting of a resource, or

```
if size and size < disk.get_disk_size(base):
    LOG.error('%s virtual size larger than flavor root disk
          size %s' % (base, size))
    raise exception.ImageTooLarge()
```

(a) CVE-2013-2096 – https://review.openstack.org/#/c/28717
file: nova/virt/libvirt/imagebackend.py

```
absolute_path = os.path.realpath(os.path.join(fs, *args))
if not absolute_path.startswith(os.path.realpath(fs)+'/'):
    raise exception.Invalid()
return absolute_path
```

(b) CVE-2012-3360 – https://review.openstack.org/#/c/9268
file: nova/virt/disk.py

Fig. 7. Examples of code snippets from the vulnerability patches.

not considering a possible malicious abuse of a use paradigm, or wrong handling of sensitive information. Many of these errors could have been avoided by a more attentive planning of development and a more security oriented design.

However, we also found cases where the patching process is much more complex like the **gain information** vulnerability CVE-2015-7548 where **naming conditions** allowed attackers to **read arbitrary files**. Exploiting this vulnerability requires a much more elaborated procedure involving different steps (creation of a crafted image, request of a snapshot, and downloading the snapshot) and different issues (bugs) are involved in the process mainly related with wrong file type detection and naming conditions, which allow the malicious users to read arbitrary files by smuggling them. To fix this, three patches were issued affecting collectively more than 100 lines over 5 different core *Nova* files (see OSSA-2016-001). Cases like these are much harder to be foreseen given the complexity of the multi-step exploit.

The issued patches do not always provide a definitive solution for the vulnerabilities. In four of the analyzed reports, the patch was not completely effective (*incomplete fix*), causing at a later point the discovery of other six vulnerabilities (e.g. CVE-2013-4463). In another case (CVE-2016-7498) a vulnerability was at later point reintroduced (*regression*) by an update of *Nova* leading to the issuing of a new CVE entry. Finally, for CVE-2016-2140, the first patch introduced some bugs that had to be fixed via additional patches originating a regression in the process (OSSA-2016-007).

VI. Conclusions and Lessons Learned

Our analysis revealed that in OpenStack around 2/3 of the vulnerabilities are insider threats where the source of attack is not located outside the perimeter of the cloud platform but rather inside itself. Differently from other networked systems as web applications and services, where the defenses are deployed at the border to protect from breaches from the outside, in cloud systems like OpenStack, some of the **security resources should be directed towards the inside of the platform**, e.g. by monitoring the behavior of its users.

We also observed that it is common for vulnerabilities to be discovered only after a long period of time. In more than 20% of the cases, vulnerabilities have gone undetected for more than one year, which leaves attackers with plenty of time to exploit OpenStack systems. Moreover, it also suggests that there are **a *non-negligible* number of vulnerabilities still to be discovered** in the platform.

Finally, the analysis shows that, in many cases, the bugs leading to the vulnerabilities are not very complex. There are cases where the attack pattern involves several steps and leverages on different bugs or uses specific concurrent access to resources that may be hard to foresee. But the majority of vulnerabilities originate from trivial issues, that could be prevented following simple best practices as input validation and others could be detected with better testing procedures.

Future work includes studying in detail other components of the OpenStack framework. This will further enhance the understanding of security issues and support the development security evaluation techniques such as vulnerability injection, which is the best suited technique to evaluate the defense-in-depth strategies that are key in systems of such complexity.

Acknowledgment

This work has been partially supported by the project **EUBra-BIGSEA** (www.eubra-bigsea.eu), funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement no 690116.

References

[1] OpenStack Foundation, "OpenStack - Open source software for creating private and public clouds." [Online]. Available: https://www.openstack.org/

[2] OpenNebula Community, "OpenNebula - Flexible Enterprise Cloud Made Simple." [Online]. Available: https://opennebula.org/

[3] Apache Software Foundation, "Apache Cloudstack - Open Source Cloud Computing." [Online]. Available: https://cloudstack.apache.org/

[4] Zenoss, "2014 State of the Open Source Cloud Report," Zenoss, Austin, TX, USA, Tech. Rep., 2014. [Online]. Available: https://tryit.zenoss.com/asset-lp-2014-state-of-the-open-source-cloud-report/

[5] "User Stories - OpenStack Open Source Cloud Computing Software." [Online]. Available: https://www.openstack.org/user-stories/

[6] OpenStack Foundation, "Openstack docs : OpenStack Security Guide." [Online]. Available: http://docs.openstack.org/security-guide/

[7] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2012, pp. 2457–2461.

[8] H. Albaroodi, S. Manickam, and P. Singh, "Critical review of OpenStack security: Issues and weaknesses," *Journal of Computer Science*, vol. 10, no. 1, pp. 23–33, 2014.

[9] S. Ristov, M. Gusev, and A. Donevski, "Openstack cloud security vulnerabilities from inside and outside," in *The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization*, Valencia, Spain, 2013, pp. 101–107.

[10] J. A. Duraes and H. S. Madeira, "Emulation of Software Faults: A Field Data Study and a Practical Approach," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 849–867, 2006.

[11] J. Stuckman and J. Purtilo, "Mining security vulnerabilities from linux distribution metadata," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2014, pp. 323–328.

[12] J. Fonseca, M. Vieira, and H. Madeira, "Evaluation of Web Security Mechanisms Using Vulnerability & Attack Injection," *IEEE Trans. on Dependable and Secure Computing*, vol. 11, no. 5, pp. 440–453, 2014.

[13] A. Milenkoski, B. D. Payne, N. Antunes, M. Vieira, and S. Kounev, "Experience Report: An Analysis of Hypercall Handler Vulnerabilities," in *2014 IEEE 25th International Symposium on Software Reliability Engineering (ISSRE)*, 2014, pp. 100–111.

[14] "CVE security vulnerability database. Security vulnerabilities, exploits, references and more." [Online]. Available: https://www.cvedetails.com/

[15] M. Howard and D. E. Leblanc, *Writing Secure Code*, 2nd ed. Redmond, Washington: Microsoft Press, 2002.

[16] P. Koopman and J. DeVale, "Comparing the robustness of POSIX operating systems," in *Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, 1999. Digest of Papers*, 1999, pp. 30–37.

[17] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[18] R. Chinchani, A. Iyer, H. Q. Ngo, and S. Upadhyaya, "Towards a theory of insider threat assessment," in *International Conference on Dependable Systems and Networks (DSN'05)*, Jun. 2005, pp. 108–117.

[19] M. Bishop and C. Gates, "Defining the Insider Threat," in *4th Annual Workshop on Cyber Security and Information Intelligence Research: Developing Strategies to Meet the Cyber Sec. and Inf. Intelligence Challenges Ahead*. New York, USA: ACM, 2008, pp. 15:1–3.

[20] M. Kandias, N. Virvilis, and D. Gritzalis, "The insider threat in cloud computing," in *International Workshop on Critical Information Infrastructures Security*. Springer, 2011, pp. 93–103.