Aggregating Vulnerability Information for Proactive Cloud Vulnerability Assessment

Kennedy A. Torkura, Feng Cheng, Christoph Meinel
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany

Abstract

The current increase in software vulnerabilities necessitates concerted research in vulnerability lifecycles and how effective mitigative approaches could be implemented. This is especially imperative in cloud infrastructures considering the novel attack vectors introduced by this emerging computing paradigm. By conducting a quantitative security assessment of OpenStack’s vulnerability lifecycle, we discovered severe risk levels resulting from prolonged gap between vulnerability discovery and patch release. We also observed an additional time lag between patch release and patch inclusion in vulnerability scanning engines. This scenario introduces sufficient time for malicious actors to develop zero-days exploits and other types of malicious software. Mitigating these concerns requires systems with current knowledge on events within the vulnerability lifecycle. However, current threat mitigation systems like vulnerability scanners are designed to depend on information from public vulnerability repositories which mostly do not retain comprehensive information on vulnerabilities. Accordingly, we propose a framework that would mitigate the aforementioned risks by gathering and correlating information from several security information sources including exploit databases, malware signature repositories, Bug Tracking Systems and other channels. These information is thereafter used to automatically generate plugins armed with current information about possible zero-day exploits and other unknown vulnerabilities. We have characterized two new security metrics to describe the discovered risks; Scanner Patch Time and Scanner Patch Discovery Time.

1. Introduction

Cloud security is an active research domain owing to constant emergence of threats and vulnerabilities. Traditional security schemes do not always suffice in the cloud [1]. An example is the case of vulnerability scanners, which provide early detection of security flaws in software and networks. Cloud characteristics such as virtualization require scanners to tackle security issues for both host and guest Operating Systems (OSs). We investigate this specific challenge with OpenStack, since is installed on Linux OSs. By applying quantitative security assessment to vulnerability lifecycles we discover a time lag between patch release and inclusion of patches in vulnerability scanner engines. We note also that patches released for vulnerabilities in OpenStack Kilo and Juno are yet to be implemented in the scanning engine. We believe other vulnerability scanners share similar risks since they limit vulnerability discovery to sources such as National Vulnerability Database (NVD) [2] and Open Source Vulnerability Database (OSVDB) [3]. However, threat detection systems for cloud infrastructure require accurate and current information to stop threats and vulnerabilities. Bug Tracking Systems (BTSs) such as BugZilla [4] and LaunchPad contain highly dependable and timely information on vulnerabilities [5]. Though not common in information security communities, BTSs are critical for collaboration and productivity for developers [6]. Modern BTSs are highly advanced, and are being leveraged upon in software engineering domains for bug prediction [5] and software quality improvements [7]. We leverage on these advancements in this work.

Several analysis of the vulnerability lifecycle reveal that information about vulnerabilities is first released in a BTS or Exploits Database (EDB). Hence, proactive efforts to mitigate the effects of vulnerabilities in applications include information addition of information derived from these other sources. For this reason, in this paper we propose a framework for a proactive vulnerability scanner that acquires and aggregates information from HPI-VDB, OpenStack Launchpad Bug-tracker (OLB), EDB and Symantec Database (SymDB). The system is extensible for other cloud environments and can integrate with other security systems. Our major contributions are as follows:

- We conduct an empirical security study of OpenStack vulnerability lifecycle including patching trends.
- Discover lapses that present opportunities for attackers to compromise the systems with zero-day exploits and other unknown vulnerabilities.
- Propose an extensible system that identifies and mitigates these vulnerabilities.

This rest of this paper is organized as follows: in
next Section, we discuss related works. Section III focuses on the inter-play between bugs and vulnerabilities while Section IV outlines OpenStack architecture and current security structure. We discuss on vulnerability lifecycle, vulnerability scanners and BTSs in Section V. OpenStack vulnerability analysis is in Section VI and we present our proposed framework in Section VII. Highlights of future research directions and our conclusion are summarized in Sections VIII.

2. Related Work

Several research works in the area of cloud security have applied their concept to OpenStack. Fall et al [8] applied Fault Tree Analysis to evaluate the security of OpenStack. In this research paper we focus on the vulnerabilities that can exploit the entire system. Gusev et al [9] studied multi-tenant vulnerabilities affecting OpenStack deployments. Since we employed quantitative security metrics in the analysis of OpenStack vulnerability lifecycle, we mention some work in this direction. In [10], Nappa et al analyzed patch management process for 10 popular client applications using Worldwide Intelligence Network Environment (WINE). Similar to our work, they emphasize the need to aggregate multiple security information sources for effective software security. However, we differ from them since we include BTSs, an approach motivated by Massacci et al's work [11]. In their work, Massacci et al [11] asserted that integrating BTS drastically improves the quality of vulnerability data-sets, however their results were not actively applied to vulnerability detection and mitigation. Our work covers this lapse.

There are existing works focused on the relationship between bugs and vulnerabilities, developers and users. In [34], Breu et al investigated the effect of user involvement in bug resolution by quantitative and qualitative analysis of 600 bug reports from Mozilla and Eclipse projects. Following, they suggested several ways through which BTSs could be better designed. Similarly, Chilana et al [35] proposed approaches for effective developers and user interaction and contribution to bug-fixing in BTSs. Using the Mozilla project, they observed that over the period of 11 years, involving 150,000 users, the bug reports were either redundant with little or new information or requiring normal technical support. Based on this analysis, they opine that an effective approach could be the inclusion of information extracted from BTSs in vulnerability scanners and threat detection schemes. We believe that this active approach could yield more positive and beneficial results to users than previous works. A possible challenge to this approach could be that unwillingness of software vendors to release information about existing vulnerabilities to the public i.e. differences in perception to vulnerability disclosure. However, we observe that opensource software vendors are becoming more supportive of open vulnerability disclosures.

3. Correlating Between Bugs and Vulnerabilities

The current security events demonstrate the fast pace with which vulnerabilities are discovered. Hence, software developers are increasingly under tremendous pressure to develop secure applications and fix discovered vulnerabilities. Security professionals are equally intensifying their efforts at identifying vulnerabilities especially to reduce the number of zero-day exploit. A key factor to the resolution of the afore-mentioned challenges is a clear understanding of bugs and vulnerabilities.

3.1. Differentiating Between Bugs and Vulnerabilities

Software quality assurance is a sub-area of software development focusing on development of defect-free software. This field is more mature than software security, hence several concepts used in security are derived from it. Bugs have been defined as defects within software that could translate to unwanted software behavior such as reduced functionality and excessive memory consumption. On the other hand, vulnerabilities are security-sensitive bugs i.e. “software defect that violates an implicit or explicit security policy” [36].

3.2. Integrating Vulnerability Information into Bug-Tracking Systems

A common approach for developing secure systems is by timely identification and patching of security bugs. In achieving this, several approaches adopted include vulnerability discovery, vulnerability prediction. A commonality in these approaches is the use of information from BTSs since they provide the high quality information about the lifecycle of vulnerability. Information from BTS is correlated with information from public vulnerability information sources like NVD and analyzed. However, while these approaches favor the software developers, users are not adequately catered for. This is because discovery of a bug/vulnerability does not equate to timely development and release of appropriate patches [37]. Some discovered bugs are left un-patched for several reasons including: to maintain software stability, few reported occurrences, economical reasons and lack of adequate skill-sets/manpower. This means the number of zero-day vulnerabilities could be reduced if all discovered vulnerabilities are timely fixed.

4. OpenStack Cloud Operating System

OpenStack is an open-source cloud computing software [12] developed and maintained by a large community of developers. It started originally as a joint project of National Aeronautics and Space Administration (NASA) and RackSpace, but was later out-sourced. Since then, OpenStack has
immensely grown and is currently supported by over 150 companies. In this section, we briefly describe OpenStack’s architecture and the current security structure.

4.1. OpenStack Architecture

OpenStack Liberty is the latest release of OpenStack, several new features were introduced with Liberty include role-based access control at the network level, pluggable IP addressing and image security measures (signing and verification). OpenStack consists of several services but the main ones are: swift (object storage), keystone (identity service), horizon (user interface), nova (compute), cinder (Virtual Machine (VM) storage) and glance (VM catalog service). However, the research in this work focuses on the releases upto Kilo.

4.2. OpenStack Security Structure

The OpenStack security project consists of a OpenStack Security Team and a Vulnerability Management Team (VMT). The security team’s responsibilities include production of security notes and developer guidance documents [13]. The VMT takes care of Vulnerability Management Process (VMP) which is triggered with the filing of a security bug report. Following verification of the bug, an initial impact assessment is conducted and a bugtask is generated in Launchpad BTS. Next, a patch is scheduled for development on the current branch and other affected branches. Once ready, the patch is reviewed by a specific set of developers, at the same time a Common Vulnerabilities and Exposures (CVE) is requested for the vulnerability from NVD. When the patch review is completed, the bug is pushed and committed and the vulnerability is officially announced. However, note that the announcement could happen before patch release. Most of these communications is done using mailing lists on OLB [14].

5. Vulnerability Lifecycle, Vulnerability Scanners and Bug Tracking Systems

Here we describe the vulnerability lifecycle and discuss on the workings of vulnerability scanners. We also explore the current state of art in BTSs how information can be extracted from them to form a security knowledge-base.

5.1. Vulnerability LifeCycle

A vulnerability lifecycle shows the stages vulnerabilities undergo (Fig 1) from the time of injection into a software until complete elimination. More specifically, software vulnerabilities pass through the following stages [15]:

1) Discovery Time ($t_d$): The discovery time is the earliest time a vulnerability is found in a software. Vulnerabilities could be discovered by software vendors, hackers or security researchers. The decision on whether to keep the discovery of vulnerability is open especially in open-source projects, however secrecy is the preferred option.

2) Exploit Time ($t_e$): The exploit time refers to the time which a vulnerability is exploited in the wild. Close monitoring of hacker communities and underground exploit markets is necessary for knowledge on this date, however some public databases such as EDB aggregate and retain these information for easy public consumption.

3) Disclosure Time ($t_d$): The day on which vulnerability is publicly announced is known as the disclosure time. The channels through which such announcements are made include vendor security advisories e.g OpenStack Security Advisories (OSSA), NVD, BTSs and HPI-VDB.

4) Patch Time ($t_p$): The patch time is the time software vendors release fixes to re-mediate existing vulnerabilities. Patches are released before, on or after the disclosure time. Third parties also provide fixes occasionally but these are not always trustworthy and could be malicious. A major source of $t_p$ of information on is at vendor BTSs.

5.2. Risk Levels within the Vulnerability Lifecycle

Several risk levels (Fig.1) are characterized within the vulnerability lifecycle [16] as explained below:

1) Black Risk: This risk level falls between discovery time and disclosure time. Within this period, limited information about the existence of vulnerability is publicly available. Software users are exposed to maximum levels of vulnerability.

2) Grey Risk: Grey risk falls between disclosure time and patch time. Note that, software vendors try to release patches before disclosure time in most cases as this approach offers best protection. But this is not possible in all cases owing to several factors such as economics and availability of skills. We also aim at mitigating the risk exposure within this period.

3) White Risk: The white risk is the only period within users control, patches are released and users are responsible for installation. However, several security breaches occur owing to late or non-application of patches. This is where use of automated tools like vulnerability scanners become handy.

C. Anatomy of a Vulnerability Scanner

Vulnerability scanners are tools used for timely discovery of security flaws in applications and networks [17]. There are currently a large variety of
such scanners including opensource and closed source types. However, systems and networks depend on these scanners for automated scanning for discovery of vulnerabilities and security risks like abuse of password policies, web applications security and insecure database credentials. In order to appreciate the (in) security of vulnerability scanners, an understanding of their operation is important. We use OpenVAS [18] as a case study since it is open-source and all the components can be easily inspected. Vulnerability scanners monitor and acquire security information from vulnerability information providers like NVD and HPI-VDB. From this information, plugins or security checks are developed and included in their scanning engines. Plugins are scripts which scan for specific signatures or patterns using attributes like ports, services and configuration to identify suspicious behavior. Scanning reports are thereafter generated and presented before a human administrator for further action. The report may mention whether or not a vulnerability is found and recommend specific actions if vulnerabilities are identified. However, most scanners are only suitable for mitigating security risks within the white risk period, hence the need for better, proactive scanners that go beyond publicly available information.

5.3. State of the Art in Bug Tracking Systems

BTSs are essential components in software development. They facilitate collaboration among developers and users [19], aid bug tracking/fixing and bug prediction [5]. Modern BTSs correlate information from several sources such as code repositories e.g. GitHub, bug reports and emails systems [6]. This affords quality information suitable for rapid software development which can also be harnessed for security. BTSs contain vital information for vulnerability lifecycles such as vulnerability discovery date, patch time, and disclosure time. This information has not attracted the attention of the information security community. Reasons for this include access restrictions and difficulty of extraction. Though access to BTSs is restricted in commercial software, the case is different in open-source projects. Opensource projects are focused on transparency and community style software development such as Bazaar hence the requirements for accessing BTSs are flexible enough for interested persons. This flexibility could be leveraged on to access bug information for security reasons. On the other hand, it is possible that malicious persons also access this information for malicious reasons like exploit generation. Similarly, modern BTSs such as Launchpad and BugZilla have Representational State Transfer (REST) APIs [20] that enable automated and efficient data access. Moreover, the information required for security as proposed in this paper, forms a small proportion of the overall information, these are usually marked with searchable keywords for example Launchpad uses keywords such as when to indicate vulnerability discovery date and finished or fix committed to indicate patch release date.

6. Quantitative Security Analysis of OpenStack Vulnerabilities

In this section, we explain our data collection methodology and our analysis of lifecycle of vulnerabilities affecting OpenStack. We then characterize two new metrics: Scanner Patch Time and Scanner Patch Discovery Time to measure the risks levels between vendor Patch Time and the actual integration/implementation of patches in vulnerability scanners.

6.1. Data Collection Procedure

Historical data have been a great source for obtaining empirical characteristics of software vulnerabilities [21], [22]. In order to gain insights into the vulnerability trends of OpenStack, we collected vulnerability data from the HPI-VDB [2]. The HPI-VDB is a public database of vulnerability information managed by the Security Team Hasso Plattner Institute (HPI). It consists of several other databases including the NVD and OSVDB [3]. The OSVDB is a third party vulnerability repository. These data sources are commonly used for security analysis hence our choice. We correlated data from these sources in an effort to remove inaccuracies as
mentioned in previous works [10] [23]. In order to gain insight on vulnerability lifecycle, we extract additional data from OLB [14], which is used by OpenStack for bug-tracking. BTSs have not been observed that integrating information from BTSs in vulnerability analysis could drastically increase the quality of data-sets being analyzed.

6.2. Measure of OpenStack Vulnerabilities

OpenStack software development consists of a bi-annual, major release cycles, within which 4-6 versions are also released [24]. We focus on the major releases so as to keep the analysis simple and effective. Moreover, a version-based analysis could not produce results different from ours. We did not analyze the first three OpenStack releases since no records of their vulnerabilities are available. Fig. 2 shows all vulnerabilities discovered in OpenStack, a total of 194 vulnerabilities were discovered and disclosed. We realized that 45.64% of the total vulnerabilities are shared vulnerabilities, inherited from previous releases. A steady rise of vulnerabilities climaxing at Folsom is observed. Folsom, is worst affected with vulnerabilities, having 47 vulnerabilities. However, a downward trend of vulnerabilities follows Folsom, indicating improvement in OpenStack security.

6.3. Correlating OpenStack Patching Trends with Risks in the Vulnerability Lifecycle

Since we are more interested in analyzing the risks from when vulnerability is inserted until when it is completely eliminated, we analyze OpenStack’s vulnerability lifecycle. Figure 1 shows the various risk levels associated with a vulnerability lifecycle while Figure 3 is shows patch distribution across OpenStack. The time between when vulnerability is discovered and when it is patched is imperative for understanding the levels of risks software users are exposed to. Based on our analysis, there is sufficient time for malicious attackers to develop exploits and compromise systems within these periods. In the next section we give details on our approach aimed at countering the effects of black and grey risks in OpenStack.

According to our analysis, 64.95% of vulnerabilities were patched before disclosure. This set of vulnerabilities fall within the grey risk period, while the other set of vulnerabilities that fall within the black risk period is 32.95%. The percentage of vulnerabilities patched on disclosure date is 2.06%, this are also referred to as patch-day [15]. Further analysis of the grey risk shows (Fig. 4) that 37.5% of vulnerabilities are patched within 30 days, 34.34% within 90 days while the balance of 28.13 percent are patched after 90 days. This implies that OpenStack systems could be exposed to a considerable amount of risk even after the vulnerability disclosure.

6.4. Scanner Patch-Time and Scanner Patch Discovery Time

In order to understand the risk window between Patch Time and inclusion of patch in vulnerability scanners we characterize two new metrics: Scanner Patch Time and Scanner Patch Discovery Time represented below:

\[ S_t = S_d - t_p \]  

where \( S_t \) represents the Scanner Patch Time, \( S_d \) is Scanner Patch Discovery Time and \( t_p \) is the Patch Time.

We easily get \( S_t \) for each vulnerability by inspecting its plugin in OpenVAS plugin repository [25] using its CVE identifier as search parameter. We selected OpenVAS since it is open-source, thereby affording sufficient investigation and because it is popular in the information security and research community. Figure 5 summarizes our findings for vulnerabilities that have affected OpenStack Juno and Kilo releases, the two most current releases. Out of 25 vulnerabilities that have affected both releases, 9 vulnerabilities do not have OpenVAS plugins while one was a shared vulnerability.

![Figure 4. OpenStack’s Patch Trends Across Diablo to Kilo](image)

![Figure 5. Graph showing the number of days before plugins are created in OpenVAS Scanner for OpenStack Juno and Kilo Vulnerabilities](image)
We then computed the difference between the patch time OLB bug-tasks and the date of plugin creation, to understand the risk windows. The three highest risk windows were 299 days (CVE-2014-3473, CVE-2014-3474), 298 days (CVE-2014-3520) and 340 days (CVE-2014-4167). The fastest fix was within 11 days (CVE-2014-5253, CVE-2014-5252, CVE-2014-5251).

7. Proposed Framework

Here we introduce our proposed framework (Fig. 6) to mitigate the risks characterized in Section V-D. We aggregate and correlate information from HPI-VDB, OLB, EDB and SymDB. EDB [26] is an open repository for proof-of-concept exploits while SymDB [27] is an open database of vulnerabilities exploited in the wild. EDB and SymDB have been used for quantitative security assessments [11], [28] but we apply these to create a knowledge base for generating vulnerability signatures/plugins. The first three databases were already introduced in Section IV-A. We leverage on OpenVAS [18] scanning engine for the actual vulnerability scanning. OpenVAS is a popular open source vulnerability scanning framework consisting of several tools used for security assessment. At the lower level, identification of software flaws is done by the NVTs or plugins. Open Vulnerability Assessment System (OpenVAS) currently employs about 40,627 plugins. These plugins are developed using Nessus Attack Scripting Language (NASL), a scripting language originally developed for Nessus vulnerability scanner.

7.1. Vulnerability Information Gatherer

The Vulnerability Information Gatherer collects information from the HPI-VDB, OLB, EDB and SymDB. This information is initially retained in separate MongoDB databases. We prefer MongoDB since it supports a schema-less database which is suitable for working with vulnerability databases [29]. HPI-VDB and OLB provide REST Application Programming Interfaces (APIs) for HTTP requests. SymDB and EDB support xml parsing and archive downloads.

7.2. Vulnerability Information Processor

The Vulnerability Information Processor is an important part of the proposed framework. It handles information collation, normalization and correlation. Basically, we are interested in the following information: vulnerability description, CVE, Common Vulnerability Scoring System (CVSS), discovery date, patch date, exploit availability and exploit signature/behavior which we extract as explained below:

Collation: Information retrieved from the various sources are initially retained in separate databases.
Normalization: Here information is restructured into specific format as defined in our database. Errors are removed and information is cleaned.
Correlation: We derive a new correlation algorithm to handle the correlation of information from various stages.

7.3. Plugin Generator

OpenVAS plugins are developed specific to vulnerability and a software application. For example CVE-2014-6271, the ShellShock vulnerability [30] has 26 OpenVAS plugins, each major Linux distribution had at least one of these and other applications also had their specific plugin. However, for the purpose of simplicity we choose to provide plugins for Ubuntu Linux because is the most used OS for OpenStack deployment [31]. We observed that the OpenVAS plugins are not timely developed for the disclosed vulnerabilities, for example only one of the disclosed vulnerabilities for OpenStack Kilo has a plugin. The plugin Generator automatically generates plugins. Vulnerability signature generation [32] has been used in the concept of Intrusion Detection Systems (IDS) and anti-virus systems, we gain from these research efforts but apply similar approaches to vulnerability scanning.

7.4. Central Management Server

The Central Management Server (CMS) is the hub of the proposed framework. All activities including target selection, scan configuration, scheduled scanning and scan reports generation will be managed at the CMS. The CMS can employ OpenVAS Management Protocol (OMP) for internal communication with OpenVAS scanner. OMP provides an XML-based communication platform for integration with third party systems.

7.5. Scanner Module

We leverage on OpenVAS scanner since it adequately supports third party integration [33].
OpenVAS has a modular structure including OpenVAS scanner, OpenVAS manager and OpenVAS Command Line Interface (CLI) aiding independent use of modules. The scanner synchronizes plugins from OpenVAS central repository and the local plugin cache.

7.6. Information Correlation

Correlation of vulnerability information into information security systems such as Security information and Event Management (SIEMs) is a rapidly maturing field. Such systems aim at leveraging on historical data such as logs generated from security tools to investigate a security incident or even detect an on-going attack. Though the results from this approach are very important, they tend to be reactive rather than proactive. However, a proactive system could prevent the cost of recovering from a malicious attack which amount majorly from costs of investigation or the replacement of services and other damaged IT assets. We present in this paper a proactive approach aimed at overcoming some of the above mentioned lapses of reactive security systems. Our approach is distinguished in the sense that we integrate the availability of vulnerabilities regardless of patched availability. Most vulnerability scanners are based on availability of patches published at NVD but the NVD has been shown to be incomplete and sometimes information provided could be late enough for malicious actors to act against systems. In order to provide precise information, we verify availability of exploits for specific vulnerabilities by correlating our information with information derived from EDB. Based on this effort, appropriate warnings are included in the scan report. We opine that such warnings could be integrated into security tools such as IDS to actively stop an attack that employs such exploits. In order to apply information extracted from these sources, we employ a new correlation algorithm that searches for and adds specific vulnerability attributes. The stages of the algorithm are detailed below:

Stage 1: Basic vulnerability information is extracted from OLB through the provided REST API. These include vulnerability description, references, CVE and CWE.

Stage 2: Information collected in Stage 1 is correlated with information from HPI-VDB. Additional information like CVSS and Common Weakness Enumeration (CWE) are added if not already obtained during the previous stage.

Stage 3: Availability of exploit is checked in the EDB using already extracted CVE as a basic identifier.

Stage 4: In order to gain more understanding of the exploits detected in the previous step, we map CVEs of exploits and vulnerabilities in the previous steps to corresponding signatures in the SymDB. This step provides useful information for the identification of discovered vulnerabilities such as attack patterns.

8. Conclusion

The need to integrate information for BTSs or secure information security has been explored in several approaches. Approaches which involve development teams and software users have been favored since they afford developers the ability to have feedback from the users and improve software security. This work follows this approach, however rather than just registering bugs at the BTSs, we aim at affording the users and opportunity to play a more active role through the use of vulnerability scanners and other security tools/threat mitigation strategies. We opine that this approach is especially in the current dispensation considering the rapid pace of software development, types of application developed and the advent of cloud computing, which could overwhelm the reaction time of software development teams. Several security approaches in traditional environments are not effective in the cloud, e.g. vulnerability scanners have been used to timely identify flaws in software applications and networks. However deploying them in the cloud requires more dynamic approaches considering the peculiar characteristics of cloud computing such as virtualization. In this paper, we conducted a quantitative security assessment of OpenStack and identified lapses in the vulnerability lifecycle and a time lag between Vendor Patch Time and inclusion of patches in vulnerability scanners. We therefore introduce two new metrics to characterize these risks and propose a proactive framework that monitors and correlates vulnerability and exploit information from various sources including BTSs. We discuss on a new correlation algorithm for effectively extracting this information which is thereafter used to automatically generate plugins capable of identifying existing vulnerabilities and zero day exploits. This system is applicable to even traditional computing environments and could be integrated with other security tools such as IDS and SIEMs. In the future, we would implement the system and measure its applicability. Besides normal inclusion of vulnerability information, we aim at applying machine learning approaches in the vulnerability scanner in order to obtain intelligent performance. Machine learning is current favored as a viable approach in information security e.g. malware detection.

9. References


