Automatic Configurator to Prevent Attacks for Azure Cloud System

Chijung Jung[1], Yung Ryn Choe[2], Junghwan "John" Rhee[3], and Yonghwi Kwon[4]
[1]University of Virginia, Charlottesville, VA, USA
[2]Sandia National Laboratories, Livermore, CA, USA
[3]University of Central Oklahoma, Edmond, OK, USA
[4]University of Maryland, College Park, MD, USA
[1]*cj5kd@virginia.edu,* [2]*yrchoe@sandia.gov,* [3]*jrhee2@uco.edu,* [4]*yongkwon@umd.edu*

***Abstract*— Cloud systems are integral for delivering scalable and virtualized resources globally. It also provides security updates and monitoring to keep user data safe. However, the growing complexity of these systems poses significant challenges, particularly in the realm of logging and security. It is difficult to know for users which detail is critical for further security analysis of the resources. Also, external packages used in the cloud system require updates by users to mitigate the vulnerability, but the large number of packages to manage makes them outdated versions.**

This paper shares the weakness of cloud logging systems we observed, which can be exploited by attackers. We propose a tool that configures alerts automatically when commands that have missing details in logs are executed and updates vulnerable versions of packages. Our tool leverages a list that includes the commands with missing details in logs and packages that need to be updated because of the known vulnerabilities. To make the list, we conduct complete enumerating for 1,279 commands in five major resources of Azure to find logs with missing details and search related communities to find vulnerable packages that require the manual update. We evaluate the proposed tool with eight attack scenarios based on real-world cases and the result shows that our tool prevents them successfully.

I. Introduction

Cloud systems such as AWS, Azure, and Google Cloud Platform (GCP) provide scalable and virtualized resources, software, and hardware to customers. It also allows users to use hardware, software, and system maintenance conveniently and securely. As organizations increasingly rely on cloud systems, the complexity of these environments grows in tandem, making them susceptible to vulnerabilities and attractive targets for potential threats [2], [27], [33].

This growing complexity results in increasing weaknesses within the system, particularly in the realms of logging and security. As organizations migrate critical workloads to the cloud, the demand for robust verification mechanisms and comprehensive logging systems becomes paramount. Although tremendous effort has been devoted to detecting/mitigating the vulnerabilities and offering cloud monitoring functions, this issue is not well addressed.

Logging, serving as the backbone for failure analysis, diagnosis, and anomaly detection, is pivotal to maintaining the security and reliability of cloud systems. However, as these systems grow more intricate, the limitations of existing logging mechanisms become increasingly apparent [28], [22]. For example, activities that have logs with missing details lead to inaccurate alerts and anomaly detection. Over half of the reported failures from major cloud providers could not be effectively diagnosed using existing log data [48]. This underscores the urgency to enhance logging mechanisms in cloud environments.

In addition, as the system becomes more complex and the supported technique becomes advanced, the number of required external packages is increasing. Most observed vulnerabilities in the packages are mitigated by patches provided by the package developers. However, those patches require users' updates, and requirements expected from the users for verification purposes are growing hurdles. As a result, such issues can cause the vulnerabilities of outdated external packages.

Unfortunately, the default configuration cannot complement this, and changing the configuration requires a lot of manual effort because the cloud system has a lot of resources and logs. It is hard to find which activity (i.e., command) has an issue (e.g., missing details in logs) causing a lot of time and effort to find potential problematic activities and configure log alerts manually. In addition, as cloud systems leverage many external packages, it is also difficult for the users to know which package has an issue (e.g., known vulnerability) or not.

In this paper, we explore a systematic approach for the automatic configuration of alerts and the updates for external packages for Azure. We compare all activities of five major resources in Azure with their logs to check if there are blind spots. We search related forums and communities to find out packages that are vulnerable but still used. Then, we aggregate them as a list and develop a tool that gets the list as an input. First, our tool generates alert rules when the user is involved in potentially problematic activities of the list. This aims to timely alert management, which is the key to detecting threats early. Second, the tool updates the vulnerable version of packages if they are still used in the user's environment. We create attack scenarios using observed blind spots and show how attackers can exploit them. Also, they show how our tool can help detection and analysis by creating alerts. We evaluate our proposed tool using eight attack scenarios based on real-world vulnerabilities (i.e., CVEs). The result shows that our tool prevents the attacks successfully.

Our contributions are summarized as follows:

- We developed a new configuration tool that creates alert rules and updates outdated packages automatically.
- We list potential problematic activities from five major resources in Azure that have logs with missing details and external packages that are vulnerable unless users update them.
- We evaluate the proposed approach on eight attack scenarios based on CVEs and show this proposal prevents all attacks.
- We publicly release the source code and data of our work on [47].

II. Background & Threat Model

The role of logs and alert systems is important in today's cloud environment. These systems serve as a crucial part of recording events and activities in cloud systems [32], [51], including software applications, networks, and servers. Collected logs are utilized to analyze and identify anomalies or fault conditions in order to provide alerts to system administrators and users. In addition, external package management is also important. Today's cloud system leverages external packages for development productivity, system maintenance, cost, and diversity of services [44], [4].

A. Logs and Alerts

Logs are essential for various security purposes when operating cloud systems. The administrators or users can identify and debug issues that occur during system operation using logs. For security monitoring, logs detect and respond to security threats. Events such as unauthorized login attempts or malicious code execution attempts can be identified in logs. Furthermore, logs are useful for tracking the occurrence and sequence of specific events. For example, a particular process can be traced within logs when a suspicious event happens. Logs are also used to monitor and optimize the performance of applications and systems. Performance metrics such as response times, load distribution, and resource usage are logged.

Alert systems analyze log data to detect anomalies or buggy behaviors and provide alerts to administrators or users. These are several common usages. First, logs are used to promptly identify and alert users to urgent situations like outages or security threats. This enables immediate responses, minimizing system potential damage. Second, alerts offer automated actions to address issues. For example, in response to suspicious login alerts, systems can automatically request enhanced authentication (e.g., two-step authentication). Lastly, the information provided by the alert systems can be used to optimize the performance of the system operations.

B. External Package Management

External package management is important for major security reasons [44], [4]. External packages are essential components of software applications and systems, but they may contain security vulnerabilities. Known vulnerabilities continue to be discovered, and attackers can exploit these vulnerabilities. Therefore, updating external packages to the latest versions is an essential step to keep your system secure and defend against new threats. In addition, timely updates can improve development productivity and allow new features to be added quickly. For this reason, periodic external package updates and management are strongly recommended to users in cloud environments. Various security and update strategies being studied can be applied to maintain and improve the safety and stability of cloud systems.

C. Threat Model

We assume an attacker knows the target system (e.g., VM or specific resources under Azure service) and its package version information. An attacker launches a stealthy attack, exploiting the vulnerability of the installed package in the victim's machine or the blind spots of the logging system, which makes the investigation challenging. For the virtual machine, an attacker cannot compromise the machine or the data directly because the attacker does not have access to the victim's machine. Also, an attacker does not perform zero-day attacks but prefers to perform known attacks (e.g., CVE) on vulnerable software. In addition, for the resources in the cloud system, the attacker does not delete or disable the resource directly because that activity specifies the attacker immediately.

We target a virtual machine or resources under the Azure cloud system. The virtual machine has multiple external software to facilitate the user's work, and they are required to be updated by the user. We focus on making users' software environments in virtual machines more secure through timely updates and informing users of situations that require double-checking by using an alert system. Fixing the vulnerabilities of software or cloud systems is not our scope.

III. Motivating Example

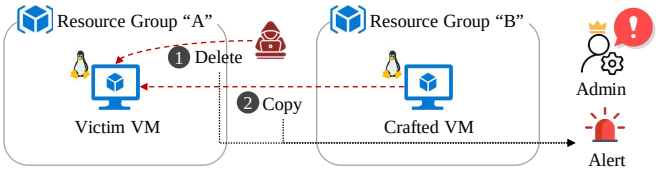


Fig. 1. Attack scenario for indistinguishable logs.

Figure 1 shows the target resource in Azure and the attack scenario used in our motivating example. We use the Ubuntu 20.04 virtual machine, a resource in Azure to illustrate our approach. We assume an adversary wants to deliver a malicious payload to the victim's virtual machine and already takes an account with access (e.g., contributor role) to resource group 'A' in Figure 1. However, the adversary tries to make a crafted virtual machine that already has malicious programs that the adversary wanted and swap the original virtual machine and fake (i.e., crafted) virtual machine rather than direct access because access to the victim's virtual machine may be detected by the log that stores the access history. To do this, the attacker deletes VM (❶) and copies

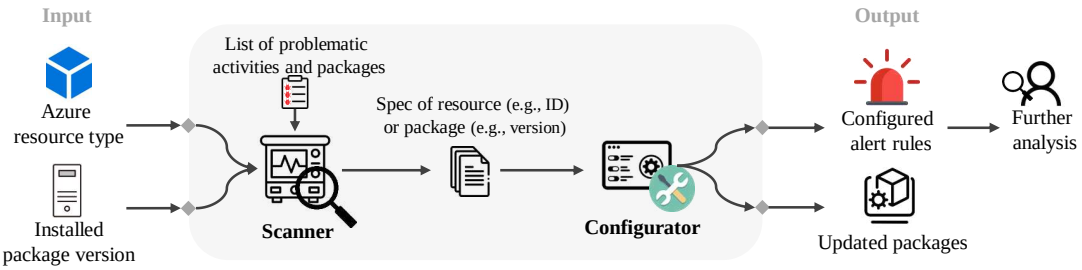


Fig. 2. Overview of the proposed approach.

the crafted VM that has the malicious program from another resource group (❷).

Challenges. Different activities (i.e., different commands and operations) should be distinguishable in logs, but unfortunately, some are logged as the same operation name in logs. In this case, the attacker's behaviors are logged as 'delete' for ❶ and 'create' for ❷, which is indistinguishable from the restore process that is also logged as 'create', meaning that the admin cannot recognize the original virtual machine is swapped by the crafted one. The admin may recognize that the virtual machine has an issue if the user reports it, but it is hard to know the consequence of the attack after time has passed. Also, it is not easy to diagnose the issue such as when it happens or who performs this by diagnosing logs. It requires further investigation with additional log analysis (e.g., activity logs of all other resource groups including resource group "B").

Our Approach. Our tool creates an alert rule when 'delete (❶)' or 'create (❷)' are executed and collects activity logs when alerts are triggered. Our proposed tool is based on the list of commands that are logged with missing details, such as the source of 'created' or 'copied' virtual machine in this example. To make the list, we investigate all commands in major resources in Azure. Note that our tool does not prevent those activities directly because of false positives. As a result, the admin can recognize when the problematic commands are executed and can investigate the context (e.g., what happened and who did it) using them easily.

IV. The Proposed Approach

Figure 2 shows the overview of the proposed tool. The tool takes an Azure resource type (e.g., storage account) and an installed package version (e.g., PyYAML 3.13) as input parameters. Then, the Scanner compares the given input with the list of problematic activities and packages. Based on the result from the Scanner, the Configurator creates alert rules or updates packages automatically.

A. Scanner

To check whether the user is involved with the resources that have activities leading to lacking logs and the machine (i.e., virtual machine) has vulnerable packages, it scans the Azure resource and the machine as shown in Figure 3. For example, the Scanner sends a query (❶) to list a storage account, which is a resource where a user may be involved in problematic activities. Then, the Scanner parses the response

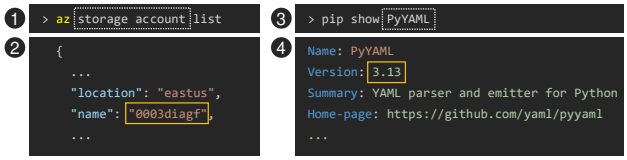


Fig. 3. An example of the Scanner's queries and obtained information.

and obtains the output information (❷), including the storage name (yellow box) or storage ID. When a package version is provided as input, the Scanner executes the command (❸) to obtain the package information and parses the version (❹). After it obtains detailed information about the resource (e.g., Storage account name) or the package (e.g., PyYAML 3.13), it sends this information to the Configurator. However, it is challenging to figure out which resource or package has an issue or not. To solve this challenge, we make a list of problematic activities and packages and leverage this list for the next step.

List of Problematic Activities and Packages To make the list of problematic activities, we first conduct a complete enumeration of 1,279 activities of 5 Azure major resources (i.e., storage account, virtual machine, virtual network, database, web application). We execute each activity and investigate how it works and is logged. If logs from two different activities (e.g., private-endpoint-connection approve and reject) have the same operation name (e.g., write) and other details do not have any difference, we register these two activities in the list.

Figure 4 shows an example. The above log is from 'private-endpoint-connection approve', and below is the log from 'private-endpoint-connection reject'. However, the details of the two logs are indistinguishable except for the

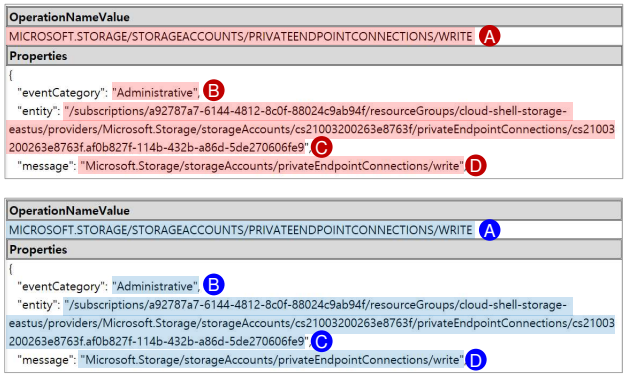


Fig. 4. Example of logs with missing details.

TABLE I
LOGS WITH MISSING DETAILS

Missing details	Operation name	Log name
Policy changes	container policy create	Storage blob log
	container policy delete	Storage blob log
	container immutability-policy create	Storage blob log
	container-rm update	Storage blob log
	blob service-properties delete-policy update	Storage blob log
	table policy create	Storage table log
	table policy delete	Storage table log
	queue policy create	Storage queue log
	queue policy delete	Storage queue log
	share policy create	Storage file log
Updated contents	file resize	Storage blob log
	blob service-properties update	Storage blob log
	container metadata update	Storage blob log
	container restore	Storage blob log
	private endpoint connection approve	Storage blob log
	private endpoint connection reject	Storage blob log
	entity replace	Storage table log
	entity insert	Storage table log

logged time. Operation names (A and A) have the same value, 'write', and the other details of properties are indistinguishable (B~D and B~D). More logs with missing details are shown in Table I.

For the problematic packages, we search issue pages of Azure GitHub repositories [20], [5], [8], [43], [9] and National vulnerability database [35] for CVEs that can still be possible in resources related to Azure. In a case where the vulnerability of specific software is fixed, and a patched version is installed as a default choice by Azure, we do not consider this to be possible to occur again.

B. Configurator

Comparing the list of problematic activities and packages, if a user is involved in a potentially problematic activity, the Configurator creates log alert rules. For example, it creates an alert when 'approve' or 'reject' activities are executed and logged. Also, it collects additional logs, such as activity logs, when the target activities are observed for further analysis. If the user's machine has the vulnerable version of a package, it updates it to the latest version (e.g., update core rule set from 3.0 to 3.2).

C. Indistinguishable Log Events

While we were enumerating the activities from 5 Azure resources, we observed a list of log events that are not distinguishable. We categorize them into four types and create four attack scenarios based on them. Three of them are presented except for the missing details case that is demonstrated in Section III. They show how attackers can exploit indistinguishable log cases as a blind spot and how our tool can mitigate them.

Careless management of a credential. Figure 5 shows a scenario in which the attacker can access a storage account without permission. Users of web applications can store connection strings as an environmental variable to access the storage account easily (1). However, the other users can also obtain the connection strings by reading the environment variable as long as they are the users of the web application (2). When the attacker targets one of these users, a lateral movement to the storage account is possible (3). Our tool creates an alert rule when access through connection

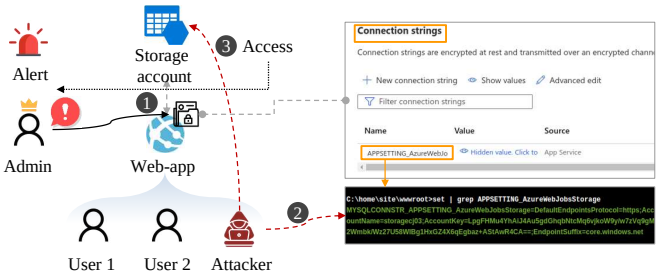


Fig. 5. Attack scenario for the carelessly managed credential.

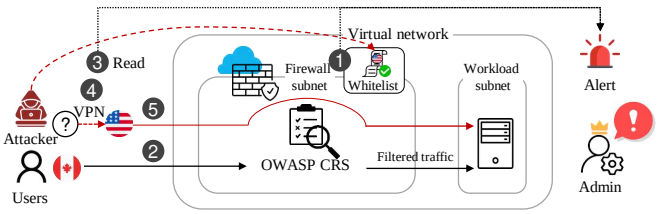


Fig. 6. Attack scenario for whitelist in web application firewall.

strings (3) is observed (i.e., logged) and collects the log of the storage account (e.g., storage blob log) so that the administrator can check if there are malicious behaviors after the access.

Whitelist of WAF ignores CRS. A web application firewall (WAF) has a whitelist. The traffic from every user should be filtered by following the core rule set (CRS) (1). However, if the specific IP address or country is registered in the whitelist, the web application firewall does not block it. Thus it is possible that an attacker registers a specific country from a blacklist without detailed conditions and abuses it.

Figure 6 shows this case scenario. An attacker takes the information of the whitelist (3) and changes the access place to it (4). Then, the attacker can bypass the CRS, and any SQL injection attack is possible (2). Our tool creates an alert rule when the whitelist is created with the condition of the country or someone tries to change (i.e., update) the whitelist so that the administrator can check if there are attack patterns bypassing WAF.

Outdated OWASP CRS. WAF currently supports OWASP CRS 2.2.9, 3.0, 3.1, and 3.2. If the current version of CRS is outdated, the attackers can use the pattern that is detected by a higher version but not by the current version (1) after obtaining the current version by profiling as shown in Figure 7. Our tool updates the components (e.g., OWASP CRS) to the latest version (2) so that the new pattern is also detected and blocked by WAF (3).

V. EVALUATION

A. Experiment Setup

CVE Selection. We use eight CVEs shown in Table II. To select CVEs, we search the Issue pages of the GitHub repository containing Azure modules and the National Vulnerability Database. When we select CVEs, we exclude the CVEs that have already been patched and are, therefore, unlikely to occur. In other words, the selected CVEs can still be possible unless users update them manually. We create

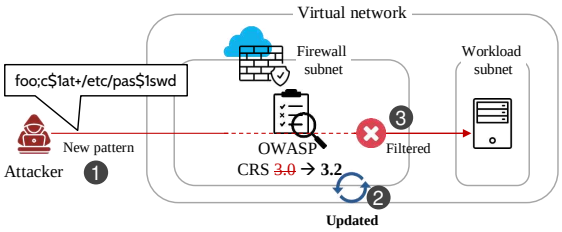


Fig. 7. Attack scenario for outdated OWASP core rule set.

TABLE II
CVEs USED FOR EVALUATION.

CVE	Description	Env.[1]
CVE-2021-38647 [11]	RCE using vulnerability of omi libarary	U20.04[2]
CVE-2022-39327 [14]	RCE using vulnerability of Azure CLI	Win10[3]
CVE-2020-1747 [10]	RCE using vulnerability of PyYAML	U20.04[2]
CVE-2023-23383 [17]	Spoofing through Azure Fabric Explorer	Win10[3]
CVE-2022-24439 [13]	RCE using vul. input validation in GitPython	U20.04[2]
CVE-2022-46169 [16]	RCE after authentication bypass in Cacti	U20.04[2]
CVE-2023-38646 [18]	RCE after unauthorized access in Metabase	Win10[3]
CVE-2023-38831 [19]	RCE when view of malicious rar file	Win10[3]

1: Environment for CVE. 2: Ubuntu 20.04. 3: Windows 10.

eight scenarios by reproducing the selected CVEs to evaluate the proposed approach.

Environment Setup. All experiments are conducted on a machine with an Intel Core i9 3.70GHz processor and 64GB RAM, running Windows 11. We use Ubuntu 20.04 and Windows 10 as the operating systems of the victim machines in attack scenarios.

B. Effectiveness

We reproduce eight CVEs that are mentioned in Azure community or have the possibility of exploitation in a VM environment. Table II shows the description of CVEs and the environment where we reproduce them. The first four CVEs are directly mentioned from the Azure CLI repositories [20], [5], [8], [43], [9] and the second four CVEs are found from the searches from general GitHub repositories or from NVD [35]. Note that every POC used in this evaluation is available on [47].

Vulnerability in OMI of Azure Monitor. Figure 8-(a) shows the first scenario (CVE-2021-38647 [11]). To reproduce this CVE, we leverage a Proof-Of-Concept (POC) code [12]. Open Management Infrastructure (OMI) is an open-source project [36]. Azure Monitor Agent requires to collect the logs, and it allows the administrator to manage the server remotely (e.g., configuration or monitoring) (❶). However, when the version of the OMI package is lower than 1.6.8.0, it has a vulnerability that allows an attacker to conduct a remote code execution attack without authentication (❷). Unfortunately, this vulnerability still works unless the user has updated it manually. Furthermore, this attack is not detected by Azure because the default log (i.e., syslog) collected by Azure Monitor Agent is not detailed enough to detect this, as shown in Figure 9. For instance, when the user executes a command `logger "This is from me"` (❶), which is logged in the left side of Figure 9, while the right side shows the log for an attack script that execute `logger "Log_attack"` (❶). The benign command is executed with the user name (i.e., spark), which is logged in ❸, and the malicious command is executed by the root user, which is logged in ❸. In addition, logs show the argument of `Logger` in ❷ and ❷. Except for `SyslogMessage` and `ProcessName`, which do not show the attack signature as they can be changed by the user anytime, there are no distinctive differences between these two executions.

To mitigate this, users should configure additional logs for this attack or update the vulnerable package manually. Additional logs may be helpful for the analysis but do not guarantee prevention, and additional information to be collected in additional logs may differ from attacks. When there is an outdated OMI package in VM, our tool automatically updates the OMI package to the latest version that patches the vulnerability without requiring additional effort from users.

Flaw in Azure Command Line Interface. In the second scenario (Figure 8-(b)), when the version of Azure CLI, which is the command line interface, is lower than 2.40, it has a vulnerability for potential code injection (CVE-2022-39327 [14]). We refer to a POC [15] to reproduce this case. The victim executes an Azure command using Azure CLI (❷) following the malicious instruction crafted by an attacker (❶). Then, the arbitrary code (e.g., connection to the attacker's machine) that is intended by the attacker is executed (❸). This vulnerability still works when the user executes the malicious instruction in Windows PowerShell. We analyze the differences between benign behavior and the attack, and they show different process traces. Fortunately, the default logs (i.e., SecurityEvent Log) collected by the Azure Monitor Agent can show it. However, an additional detection rule is required and this may not work well when the attacker uses more complex routes. Our tool prevents this attack by updating the Azure CLI. First, it scans the VM and checks if the Azure CLI is outdated or not. Then, our tool automatically updates it to the latest version.

Flaw in Handling Untrusted YAML Files. Figure 8-(c) shows the third scenario that uses remote code execution vulnerability in versions previous to 5.3.1 of PyYAML (CVE-2020-1747 [10]). We use [41] to reproduce this CVE case. The attacker inserts malicious code in the YAML file (❶), and the victim executes the Python program (❷), loading the YAML file through the vulnerable version of PyYAML (❸). Then, the arbitrary code (e.g., connection to the attacker's machine) that the attacker intended is executed (❹). This can happen when an outdated Python is used. The default log (i.e., syslog) collected from the Azure Monitor Agent does not show the differences between the legitimate connection and the malicious connection executed by attacks. Our tool scans the user's machine and checks if PyYAML needs to be updated or not. Then, our tool automatically updates it to the latest version that patched the vulnerability.

Service Fabric Explorer Spoofing Vulnerability. In the fourth scenario, the Azure Fabric Explorer (version lower than 9.1.1436.9590) has a vulnerability that allows the attacker to launch a remote code execution attack (CVE-2023-23383 [17]) as shown in Figure 8-(d). To reproduce this

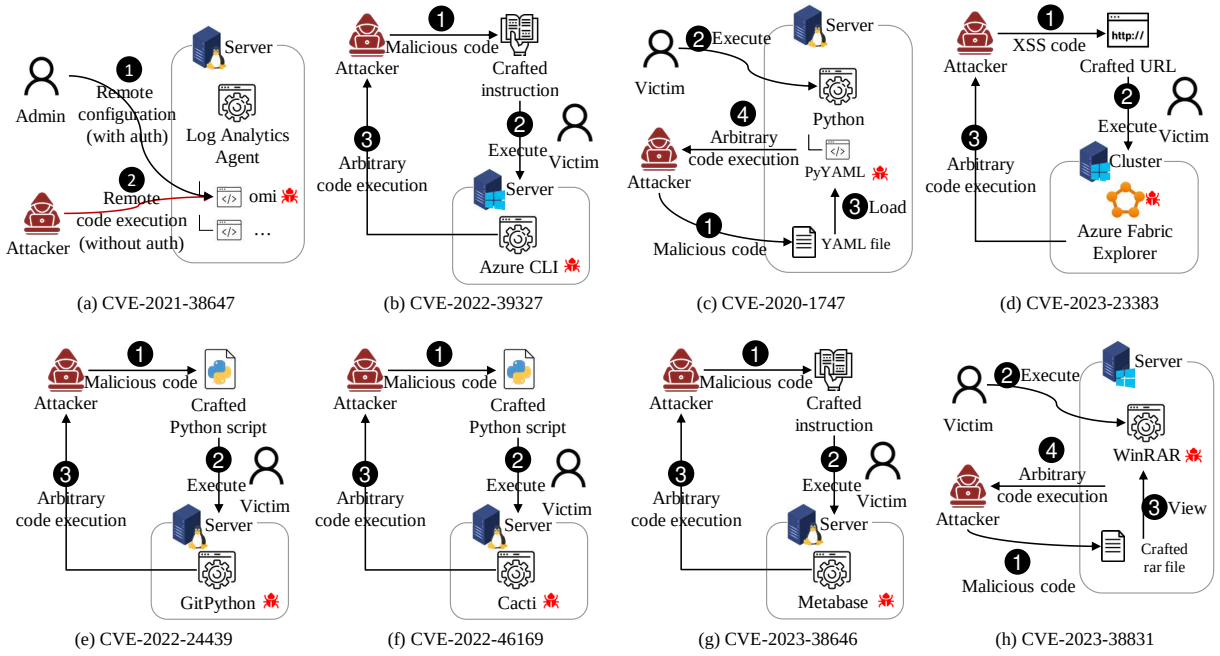


Fig. 8. Attack scenarios based on the real-world vulnerabilities.

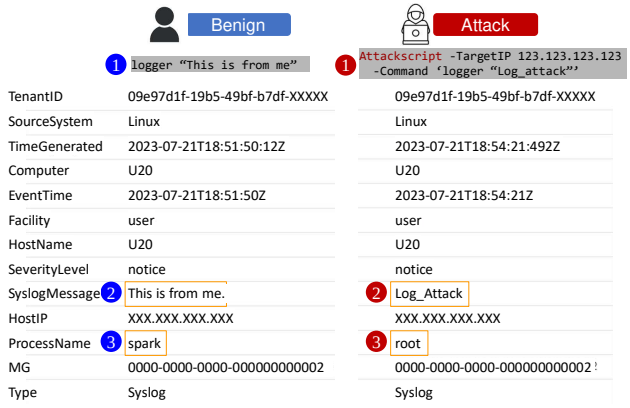


Fig. 9. Log differences between benign and malicious behaviors.

vulnerability in our evaluation, we use a PoC [38]. The attacker deploys the crafted URL, including malicious code for an XSS attack (❶). Victim executes a crafted URL (❷) that executes the arbitrary code (e.g., downloading docker and connecting to the intended website) intended by the attacker through the Azure Fabric Explorer (❸). This attack does not show signatures in the default logs collected by Azure but can be prevented by our tool. The tool scans the user's machine and checks whether the Azure Fabric Explorer needs to be updated. Then, our tool automatically updates it to the latest version that does not have the vulnerability.

Vulnerable User Input Validation in GitPython. Figure 8-(e) shows the scenario in which the attacker exploits the vulnerability of GitPython [25] (lower than 3.1.30). We use a PoC [39] to reproduce this vulnerability (CVE-2022-24439 [13]). The attacker injects a maliciously crafted remote URL into the Python script (❶). In this vulnerable version, executing the command without sufficient sanitization of the input argument is possible due to the vulnerable user input validation (clone command). As a result, when the victim executes the crafted Python script (❷), it executes the arbitrary code intended by the attackerd (❸). As the default log (e.g., Syslog) supported by Azure does not capture the attack signature, it is difficult for the user to know or prevent this attack without manually checking on the obtained script. Our proposed tool scans the server, checking whether the version of GitPython is 3.1.30 or higher, and it automatically updates it to the latest version.

Authentication Bypass in Cacti. Figure 8-(f) shows the scenario in which an unauthenticated user executes arbitrary code on a server running Cacti [6] (lower than 1.2.22) due to the vulnerability of `remote_agent.php`. To reproduce this scenario, we use a PoC [37]. The attacker maliciously crafts the Python script that bypasses the authentication process in `remote_agent.php` (❶). Then the victim executes the crafted script (❷), and malicious code injected by the attacker is executed (❸). Our tool checks the current version of Cacti installed on the user's machine and updates it to the latest version.

Unauthorized Access through Metabase. The attack scenario in Figure 8-(g) shows how an attacker exploits the software security flaw of Metabase (lower than 0.46.6.1 for open source) [34] (CVE-2023-38646 [18]). We reproduce this scenario using a PoC [40]. The flaw originates from the pre-auth API endpoint, which allows the attacker to execute arbitrary code without authentication. In this scenario, the attacker injects the malicious code in crafted instruction (❶) in advance. Then, the victim executes it (❷), and the script executes the command intended by the attacker (❸). Our tool updates the Metabase automatically so that the vulnerability is mitigated.

Maliciously Crafted rar File. Figure 8-(h) demonstrates the

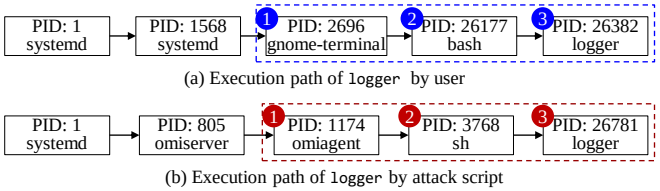
attack scenario in which an attacker exploits the vulnerability (CVE-2023-38831 [19]). We reproduce this scenario using a PoC [42]. WinRAR [45] before 6.23 allows the attacker to execute arbitrary code when a victim tries to view files with a ZIP archive. The attacker injects the malicious code in a crafted `rar` file (❶). Then, the victim executes WinRAR (❷) to view which files are in a zipped file (❸). The malicious code is executed (❹), and the attacker achieves what she wants. Unfortunately, depending on the level of injected code, the security mechanism in Windows may not work properly. Our tool automatically updates the WinRAR to a higher version than the 6.22 version to mitigate this vulnerability before it happens.

VI. Discussion

Scalability of Our Approach. To support new commands with missing detailed logs or vulnerable packages, two tasks are required: (1) identifying which command execution needs to be monitored or which package needs to be updated, (2) including them in a tool. Identifying problematic commands takes non-trivial effort, but it is required one time for each resource because updates of commands in the cloud system are not frequent. Also, additional work to update this process does not take long because the update is required only for added or modified commands. Incorporating them into a tool takes less than 1 hour (by a graduate student with moderate experience in the target cloud platform). Considering most cloud systems support command line interfaces, adding new features to a tool is not difficult.

Future Directions. There are two aspects of the future directions for our approach: empirical and technical. For the empirical aspect, applying our approach to other resources in Azure after investigating blind spots from them can be the future work. Also, applying our approach to other platforms such as Amazon Web Service [1] or Google Cloud Platform [26] can be an extension of our work in an empirical aspect. In addition, making the manual process in our approach, which compares logs to check whether they are indistinguishable or not due to the missing details, fully automatic can be practical future work in the technical aspect.

Moreover, the enhancement of logs that will be complementary to our tool can be the future work. Figure 10 shows an example. We further analyze the attack signature of CVEs that we reproduced in our evaluation and the logs that Azure can collect additionally. For instance, in the vulnerability of the OMI case (CVE-2021-38647), We observe that the execution paths between benign and malicious behavior (i.e., by attack script) are shown different as shown in Figure 10-(a) and (b), respectively. The user typically executes `logger` in a terminal, which is reflected as a sequence of `gnome-terminal` (❶), `bash` (❷), and `logger` (❸). However, when the attack script exploits the omi vulnerability, the execution path shows `omiagent` (❶), `sh` (❷), and `logger` (❸), which is rare. It means that syslog enriched with process ID and parent process ID can provide critical information that can distinguish suspicious behavior



(a) Execution path of `logger` by user

(b) Execution path of `logger` by attack script

Fig. 10. Execution path of `logger` executed by user and attack script

(red box in Figure 10-(b)) from normal behavior (blue box in Figure 10-(a)).

VII. Related Work

Event Monitoring and Alert Systems. Event monitoring and alert systems have been widely used as essential elements for cloud providers as well as cloud users. They are used to check their system reliability [46], performance optimization [21], security [7], scaling [29], or cost management [23]. In particular, event monitoring and alert systems are helpful in detecting and responding to threats. [32] proposes a framework to automatically detect incidents (i.e., severe enough alerts) based on the incident management platform of Microsoft Azure. [51] focuses on obtaining alerts from textual information to detect security issues in real-time. Our work provides an automated way to make alerts for potentially problematic activities, complementary to the techniques.

Log Enhancement Techniques. When the system fails, logs that include the information of the failure moment are useful evidence for diagnosing the root causes. Hence, log details are critical in terms of the efficacy of logging. There is a line of research [48], [52], [24], [31], [50], [49] about log enhancement that improves log-based debugging by making logs richer and more efficient. [48] showed the logs about the majority of system failures do not have enough details and can be significantly improved by adding more information. Then, they propose ErrLog, a tool that can add more logging statements and achieve less diagnosis time using it. [52] focused on where to place a log printing statement as which information is logged is related to this. They propose Log20, a tool that can optimize the placement of log printing statements with less overhead. Yuan et al. [50] presented LogEnhancer, which has additional variable values as enhanced information in each log printing statement. Our work is complementary to these techniques as it provides timely alerts, which is helpful in diagnosing the event with enhanced logs.

Updates of External Packages in Cloud Environment. The importance of external package updates in cloud computing environments is emphasized in various related studies. These studies discuss why external package updates are important for various aspects such as security [44], [4], stability [30], and performance optimization [3]. In particular, [44] mentions that attackers can exploit known vulnerabilities in packages or containers. [4] also proposes an advanced secure package manager to prevent the system from malicious package developers. Our work is complementary to them as it updates the potentially vulnerable packages automatically.

VIII. Conclusion

In this paper, we propose a tool that configures alerts and updates vulnerable versions of packages automatically in Azure cloud system. This tool is based on the list of potential problematic activities with missing details in logs and vulnerable versions of packages that require manual updates. We evaluate the proposed tool using eight attack scenarios created from real-world vulnerabilities, and the results show the effectiveness of the tool in each attack scenario.

Acknowledgments

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This article describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government. This work was supported through contract #70RSAT21KPM000105 with the U.S. Department of Homeland Security Science and Technology Directorate.

References

[1] Amazon, "Amazon web services," 2023, https://aws.amazon.com/.
[2] "Azure status history," 2023, https://azure.status.microsoft/en-us/status/history.
[3] R. Bacci di Capaci and C. Scali, "A cloud-based monitoring system for performance assessment of industrial plants," *Industrial & Engineering Chemistry Research*, vol. 59, no. 6, pp. 2341–2352, 2020.
[4] F. Brown, A. Mirian, A. Jaiswal, A. Notzli, and D. Stefan, "Spam: A secure package manager," *USENIX HotSec*, vol. 2017, p. 7, 2017.
[5] "Bundled python version has multiple vulnerabilities," 2023, https://github.com/Azure/azure-cli/issues/26740.
[6] "Cacti," 2023, https://www.cacti.net/.
[7] Z. Chen, G. Xu, V. Mahalingam, L. Ge, J. Nguyen, W. Yu, and C. Lu, "A cloud computing based network monitoring and threat detection system for critical infrastructures," *Big Data Research*, vol. 3, pp. 10–23, 2016.
[8] "Critical cve found in pcre2 cve-2022-1586/cve-2022-1587," 2022, https://github.com/Azure/azure-cli/issues/22862.
[9] "Cve-2020-14343 - upgrade pyyaml from 5.3.1 to 5.4," 2023, https://github.com/Azure/azure-cli/issues/16650.
[10] "CVE-2020-1747," 2020, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-1747.
[11] "CVE-2021-38647," 2021, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-38647.
[12] "CVE-2021-38647," 2021, https://github.com/AlteredSecurity/CVE-2021-38647.
[13] "CVE-2022-24439," 2022, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-24439.
[14] "CVE-2022-39327," 2022, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-39327.
[15] "POC-CVE-2022-39327," 2022, https://security.snyk.io/vuln/SNYK-PYTHON-AZURECLI-3063430.
[16] "CVE-2022-46169," 2022, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-46169.
[17] "CVE-2023-23383," 2023, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-23383.
[18] "CVE-2023-38646," 2023, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-38646.
[19] "CVE-2023-38831," 2023, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-38831.
[20] "Cve in python package dependency invoke," 2020, https://github.com/Azure/azure-cli/issues/14553.
[21] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E. C. Zermeno, E. Rubow, J. A. Docauer *et al.*, "Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization," in *USENIX NSDI 2018*.
[22] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "Metric selection and anomaly detection for cloud operations using log and metric correlation analysis," *Journal of Systems and Software*, vol. 137, pp. 531–549, 2018.
[23] T. Forell, D. Milojicic, and V. Talwar, "Cloud management: Challenges and opportunities," in *IPDPSW 2011*.
[24] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 24–33.
[25] "GitPython," 2023, https://pypi.org/project/GitPython/0.3.2/.
[26] Google, "Google cloud platform," 2023, https://cloud.google.com/.
[27] "Google cloud status dashboard," 2023, https://status.cloud.google.com/summary.
[28] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *ISSRE 2016*.
[29] C.-L. Hung, Y.-C. Hu, and K.-C. Li, "Auto-scaling model for cloud computing system," *international journal of hybrid information technology*, vol. 5, no. 2, pp. 181–186, 2012.
[30] R. Koller, C. Isci, S. Suneja, and E. De Lara, "Unified monitoring and analytics in the cloud," in *HotCloud 2015*.
[31] H. Li, W. Shang, Y. Zou, and A. E. Hassan, "Towards just-in-time suggestions for log changes," *Empirical Software Engineering*, vol. 22, pp. 1831–1865, 2017.
[32] L. Li *et al.*, "Fighting the fog of war: Automated incident detection for cloud systems," in *USENIX ATC 2021*.
[33] "Aws post-event summaries," 2023, https://aws.amazon.com/cn/premiumsupport/technology/pes/.
[34] "Metabase," 2023, https://www.metabase.com.
[35] "National vulnerability database," 2023, https://nvd.nist.gov/vuln.
[36] "Open management infrastructure," 2023, https://github.com/Microsoft/omi.
[37] "CVE-2022-46169 PoC," 2023, https://github.com/ariyaadinatha/cacti-cve-2022-46169-exploit.
[38] "Super fabrixss (cve-2023-23383)," 2023, https://orca.security/resources/blog/super-fabrixss-azure-vulnerability/.
[39] "GitPython-remote code execution," 2023, https://security.snyk.io/vuln/SNYK-PYTHON-GITPYTHON-3113858.
[40] "Metabase pre-auth rce (cve-2023-38646) poc," 2023, https://github.com/m3m0o/metabase-pre-auth-rce-poc.
[41] "Deserialization of untrusted data in pytorchlightning," 2020, https://huntr.com/bounties/31832f0c-e5bb-4552-a12c-542f81f111e6/.
[42] "CVE-2023-38831 (POC)," 2023, https://github.com/MorDavid/CVE-2023-38831-Winrar-Exploit-Generator-POC.
[43] "Support python 3.11," 2022, https://github.com/Azure/azure-cli/issues/24494.
[44] B. Tak, C. Isci, S. Duri, N. Bila, S. Nadgowda, and J. Doran, "Understanding security implications of using containers in the cloud," in *USENIX ATC 2017*.
[45] "RARLab WinRAR," 2023, https://www.win-rar.com.
[46] Y. Xu, K. Sui, R. Yao, H. Zhang, Q. Lin, Y. Dang, P. Li, K. Jiang, W. Zhang, J.-G. Lou *et al.*, "Improving service availability of cloud systems by predicting disk error," in *USENIX ATC 2018*.
[47] "Automatic configurator repository," 2023, https://github.com/ezazconf/src.
[48] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: Enhancing failure diagnosis with proactive logging," in *OSDI 2012*.
[49] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *ICSE 2012*.
[50] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," *TOCS 2012*.
[51] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang *et al.*, "Real-time incident prediction for online service systems," in *ESEC/FSE 2020*.
[52] X. Zhao, K. Rodrigues, Y. Luo, M. Stumm, D. Yuan, and Y. Zhou, "Log20: Fully automated optimal placement of log printing statements under specified overhead threshold," in *SOSP 2017*.