Poster: Are we reasoning about cloud application vulnerabilities in the right way?

1st Stefano Simonetto
Department of Pervasive Systems
University of Twente
Enschede, The Netherlands
s.simonetto@utwente.nl

2nd Peter Bosch
Department of Pervasive Systems
University of Twente
Enschede, The Netherlands
h.g.p.bosch@utwente.nl

***Abstract*—Enterprises are quickly transitioning to container orchestrators, like Kubernetes, which helps developers and engineers manage a large number of container images, pods, and nodes. However, this new approach does not solve the problem of software vulnerabilities but arguably it makes vulnerability management harder. Most of the time, companies have to deal with thousands of containers in a dynamic environment since they can fail, and be rescheduled in other nodes. All these factors have a great impact on the vulnerability management system because the vulnerabilities and misconfigurations in the system are too many to be manually operated, so we seek a tool to highlight the most dangerous (we need a clear definition of dangerous) to prioritize them.**
This paper wants to emphasize the need for a vulnerability prioritization method and a defense technique improvement.

***Index Terms*—Vulnerability, Prioritization, Container orchestration, Attack kill-chain.**

I. Introduction

Cloud application vulnerabilities have devastating consequences on our digital society, threatening our privacy, finances, and critical infrastructure. CISQ (Consortium for Information and software quality) estimates that the cost of poor software quality in the US has grown to at least $2.41 trillion, but not in similar proportions as seen in 2020. The accumulated software Technical Debt (TD) has grown to roughly $1.52 trillion [2].
In the past few years, the research community proposed sophisticated approaches and techniques to enhance automated security testing and promptly identify vulnerabilities before they can be exploited by malicious attackers.
As a result, today a large variety of tools are available to effectively detect vulnerabilities. However, most organizations do not know how to deal with the tons of vulnerabilities also because these tools are prone to produce false positives. The usual behavior is to patch them based on the score produced by each individual vulnerability.
As if the problem wasn't complicated enough, the container orchestration scenario makes the situation more challenging due to the rapid and continuous deployment of new containers and pods in such environments.
In the real world, attackers put together multiple vulnerabilities to successfully compromise systems as shown in Fig.1 which is taken from a real attack scenario and is related to the ATT&CK framework [5].
Thus, analyzing vulnerabilities in combination with each other represents a fundamental step to obtain a realistic "big picture" of their implications.
Furthermore, most defensive solutions are reactive solutions, like intrusion detection systems, system calls monitoring, etc. Even if these techniques are very well-established, they are affected by scalability problems and are not meant to prevent an attack to happen. This poster's abstract aims to create a discussion on how vulnerabilities are managed in the container orchestration environment and particularly in Kubernetes. More precisely, why don't we prioritize the software patches according to the real attack path instead of focusing on a single score? And if this is possible, can we automate this discovering-fixing process? Can we be more proactive during the defense?

Attack Matrix

Initial Access	Execution	Discovery	Credentials Access	Privilege Escalation	Impact
Exploiting Public Facing Application	RCE	Access to K8S API Server	List All Secrets	Use Admin Secret	Cluster Takeover
	bash / cmd inside containers	Access Instance Metadata API	IAM Role STS Token		

Fig. 1. Real attack path in Kubernetes environment

II. Problem statement

Bug-finding approaches have arguably become too successful thanks to:

1) fuzzers which inject automatically semi-random data into a program/stack and detect bugs,
2) scanners that identify vulnerabilities relying on a database of known vulnerabilities.

Industries are finding more vulnerabilities than they can fix promptly. This leads to a known situation in the security domain, named alert fatigue, where security operators cannot stay on top of the large number of alerts produced by potential security issues, as cited by Sysdig in one of their articles [7]:

”Alert fatigue Syndrome is the feeling of becoming desensitized to alerts, causing you to potentially ignore or minimize risks and harming your capability to respond adequately to potential security threats”.
Only a relatively small share of bugs discovered by fuzzers typically has relevant security implications, while the process of identifying and analyzing bugs to generate and deploy patches remains laborious, expensive, and lacks automation. Therefore, the software industry requires a way to set the priority of fixing the most harmful bugs (e.g. stand-alone or concatenated bugs that lead to a dangerous exploit), reducing costs and manual effort for organizations, and minimizing the time window in which users are exposed to potential devastating cyberattacks.
Existing work does not study a critical aspect: how multiple vulnerabilities can be combined. In fact, when taken individually, certain vulnerabilities do not provide highly dangerous capabilities. However, when combined, even low/mid-severity vulnerabilities can result in high-severity consequences.
Thus, approaches that study vulnerabilities individually, according to their severity, are insufficient as they rely on an unrealistic threat model [1].
Another issue in this field is that most of the defense techniques are meant to react during an attack rather than prevent it. In other words, only a few approaches adopt proactive defenses.

III. Discussion and Preliminary results

The main challenge for an effective vulnerability analysis and prioritization strategy is considering multiple vulnerabilities and the capabilities that they enable when combined. In real-world settings, attackers put together (“chain”) multiple vulnerabilities to successfully compromise systems. Thus, ranking vulnerabilities individually, according to their severity, is insufficient and unrealistic.
Most of the work about vulnerabilities kill-chain takes as a baseline the CVSS score which is not a good indicator of how bad the problem is in a particular scenario. For this purpose, the environmental score is introduced but it is implemented in a few papers and when used, most of the time the responsibility is delegated to a security expert. So there is no clear definition of which score is best since we need also to take into account the misconfigurations that are introduced by the infrastructure that allows the programs to work properly, as highlighted in Fig.2.
The literature is also missing a clear definition of what is best to prioritize:

1) single highest CVSS? (We disagree);
2) highest score of a kill-chain? Without taking into account its length;
3) the shortest chain with the highest score?

Finally, there is no clear definition of how to determine the overall kill-chain score. These are critical open problems to be addressed to provide a solid base to automate vulnerability prioritization and patching. The second challenge is changing the defender’s mindset: until today only two big approaches have been proposed in the microservices field about the proactive defense. Moving target defense [3] [4] and Mimic defense [11] leverage the dynamic microservices environment to create uncertainty for attackers, thereby reducing the probability of successful attacks. There are no other real ideas about proactive defenses which instead are highlighted as the leading approaches for the future.

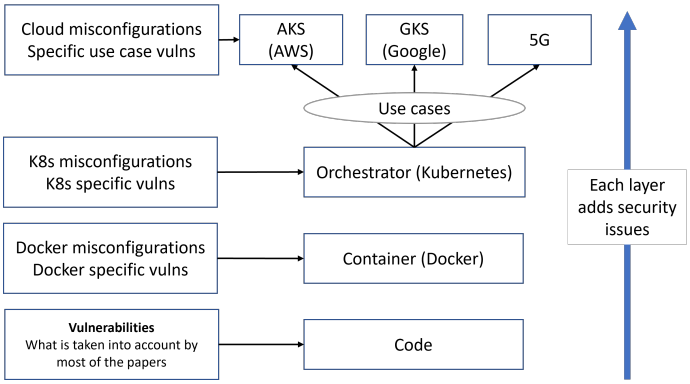

Fig. 2. Each abstraction level adds some vulnerabilities and misconfigurations

IV. Our approach

Our approach to cope with these conceptual problems consist of looking at the scenario from both sides:

1) We want to shed some light on this problem by proposing a method that assigns the proper score to each vulnerability and misconfiguration in a selected environment. We also want to give a clear definition of how to determine the overall kill-chain score and identify the most dangerous ones.
   As soon as it’s done, we aim to automatically suggest patches to fix the problem and automate the process of vulnerability/misconfiguration discovery-fix.
   To give a feeling of the proposed approach:
   a) Discovery: we want to leverage open-source tools like kube-bench [8], kube-hunter [9], and kubeaudit [10] to scan for vulnerabilities/misconfigurations and, based on these findings, start reasoning about the scores and how can they be concatenated.
   b) Fix: this task is very dependent on the problem. For instance, we can try to fix it by applying the new update or release, other times, if the role assigned to a certain user/pod is too permissive, we can try to downgrade it.
   After the patch is applied, start again with this procedure because this last step can lead to a new leak in the system.
   We believe that having a tool that is able to highlight and automatically fix the riskiest path according to the company’s crown jewels, will help to stay on top of the large number of alerts and potential security issues that otherwise could lead to alert fatigue.

2) At the same time, we think taking the right countermeasures against the attackers is very useful since most of the techniques available nowadays reason in a reactive way, e.g. IDS, NIDS, and system call monitoring are all used when the victim is under attack. We are wondering if it is reasonable to create a new proactive defense by probing the scenario before the attacker's action, rather than just responding to it after it has happened. Our long shot is to automate the penetration testing process as much as possible. Even if some tools, like Metasploit [12], already exist and are well-established, we noticed two main gaps:
   a) none of them target specifically the Kubernetes architecture or any other container orchestrator structure;
   b) they provide a framework or general reasoning, leveraging the pen-testers/red-teams knowledge and skills.

   We are not introducing a new concept because penetration testing was done before microservices, but not so often because the business architecture changed a little over time. Today we are faced with a very dynamic and versatile architecture and this change must be managed accordingly introducing, perhaps, a service that answers the question: "Can I find paths to break the cloud application?". This is a challenging task because the leading idea is to write a program that is able to automatically attack every single scenario as a pen-tester would do. Since we need to:
   a) gather the pen-testers/hacker behavior and skills;
   b) gather the faulty/vulnerable scenarios;
   c) generalize the techniques for all the possible cases.

   It is easy to realize that the task is very ambitious but, at the same time, it can be a game changer for the security industry.

V. Conclusion

More attention should be granted to this field to reduce the gap between academia and industry in container orchestration security [6]. With this article, we want to emphasize the false feeling of security generated by fixing the individual's most dangerous vulnerability, without taking into account the bigger picture. We need an effective tool that is able to obtain an accurate and comprehensive view of the vulnerabilities, evaluate risks and generate cost-effective patches.

By approaching the problem from the other side, we are thinking about taking the right countermeasures against the attackers by reasoning in a proactive way, trying to make the attack less likely to happen and more challenging for the attackers to execute it.

Finally, to provide a better understanding of our work and to propose our solution in a clear way, we aim to map this flow to the ATT&CK framework [13], which is the most used framework to describe adversary tactics, techniques, and procedures (TTPs).

References

[1] Shemesh D.H. "Schindel A. KubeCon + CloudNative North America 2022". (2023, April). [Online]. Available: https://kccncna2022.sched.com/event/182Jl

[2] Herb Krasner. "NEW RESEARCH: THE COST OF POOR SOFTWARE QUALITY IN THE US: A 2022 REPORT. (2023, April). [Online]. Available: https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2022-report/

[3] Alavizadeh, H., Jang-Jaccard, J., & Kim, D. S. (2018, August). Evaluation for combination of shuffle and diversity on moving target defense strategy for cloud computing. In 2018 17th IEEE international conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE) (pp. 573-578). IEEE.

[4] Jin, H., Li, Z., Zou, D., & Yuan, B. (2019). Dseom: A framework for dynamic security evaluation and optimization of mtd in container-based cloud. IEEE Transactions on Dependable and Secure Computing, 18(3), 1125-1136.

[5] Azarzar O. "Kubernetes Security Webishop" (2023, April). [Online]. Available: https://resources.lightspin.io/kubernetes-security-webishop?submissionGuid=a82ffed0-0e28-43b2-b23b-635c518995ad

[6] N. Alshuqayran, N. Ali and R. Evans, "A Systematic Mapping Study in Microservice Architecture," 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 2016, pp. 44-51, doi: 10.1109/SOCA.2016.15.

[7] Sysdig. "Prioritize Alerts and Findings with Sysdig Secure" (2023, April). [Online]. Available: https://sysdig.com/blog/prioritize-alerts-and-findings-with-sysdig-secure/

[8] Aquasecurity. "Kube-bench". (May 2023). [Online]. Available: GitHub https://github.com/aquasecurity/kube-bench

[9] Aquasecurity. "Kube-hunter". (2022). [Online]. Available: GitHub https://github.com/aquasecurity/kube-hunter

[10] Shopify. "Kubeaudit". (March 2023). [Online]. Available: GitHub https://github.com/Shopify/kubeaudit

[11] YING, Fei; ZHAO, Shengjie; DENG, Hao. Microservice security framework for IoT by mimic defense mechanism. Sensors, 2022, 22.6: 2418.

[12] Metasploit. "The world's most used penetration testing framework". (2023, April). [Online]. Available: https://www.metasploit.com/

[13] MITRE. "Containers Matrix". (2023, April). [Online]. Available: https://attack.mitre.org/matrices/enterprise/containers/