Empirical Analysis Of Containerization and Shift Left

Isaiah Genis
NYU Tandon
NY, USA
isaiah.genis@nyu.edu

***Abstract*— The focus of this work is to perform an empirical analysis of the impact of shift left initiatives and Docker Images for virtualization concerning the impact on security changes. We perform a historical review of system details and recorded vulnerabilities to compare remediation times and effects between shift left and traditional systems showing a significant improvement through these methods. (*Abstract*)**

Keywords— Docker, Patching, Cybersecurity, ShiftLeft

I. Introduction

Maintaining reliable inventories and keeping all systems as up-to-date as possible is a constant challenge for businesses, even with the shift to cloud systems like AWS and containerization. There are a variety of reasons updates and patches are not automatically applied or where there is limited awareness of vulnerabilities in components of the system. Many companies do not run auto-updates on production systems due to fear of downtime or other system impacts. QA and release cycles can cause critical vulnerabilities to stay open even when they are known and fixable. Services Like AWS SSM will often report successful patching despite failing to run restart-required essential updates, allowing vulnerabilities to live on systems longer. Some questions to explore include does containerization leads to faster patching or just less data on a host? How can this be done confidently with items like mission-critical databases and need to meet load standards?

We hypothesize that the migration to container-based images, like Docker, and adoption of shift-left approaches will reduce the duration of time a vulnerability lives on a production system. One of the major benefits of containerized systems, we assert, facilitates more secure systems because the entire OS and all dependencies become a part of the development and release cycle at the developer level, thereby keeping security at the pace of development. Tools like Renovate, Dependabot, SourceClear and more automate these patches and updates part of existing development efforts and run automated checks to incorporate minor changes without the need for extensive QA and configuration work for the majority of changes.

The organization of this paper is as follows. Section II will explore existing research concerning maintaining secure patching and updates of virtualized systems and performing our analysis. In Section III, we provide the empirical results of our efforts. The final section, Section IV, will highlight our conclusions and future work.

II. Related Works And Our Methods

Little has been done to analyze the impact of third-party vulnerability remediation impact from the migration to containerization. Others have explored the security of containerization itself, but this is not what we will focus on for our research. We are focused on the impact of using Docker concerning remediating OS and Package level vulnerabilities, such as done through patching and updating.

Wenhao and Zheng review the security of the Docker Architecture, focusing on four main aspects; file system isolation, device management and resource host constraints, isolation, and secure transmission of images. While they identify valid concerns around Docker's implementation, this differs from our work in that we have a focus on how Docker can alleviate the demands of patching and reduce the number of outdated and insecure packages running within a virtualized system, not the security of the underlying system running itself.

Garg & Garg present an approach on how to deliver containerized systems with security in mind. Their work does not analyze the effectiveness of their strategy or suggest how to address handling vulnerabilities identified within the system packages through the CI/CD process.

Kuacharoen, Akgul, Mooney, and Madisetti, demonstrate a solution to perform kernel updates without needing reboot systems which is one of the challenges I am also looking to address. Still, they have a fundamentally different approach to performing updates in place. In contrast, our approach would be more of a blue/green cutover with a shared file system for application-level program data.

Rather than proposing new methodologies or approaches, we wish to measure the impact of the current trending approach. We performed a comparison of vulnerability and system inventory data across hundreds of VMs, Desktops, and Docker Images to see how impactful each approach was.

We have analyzed the inventory of packages as well as OS vulnerabilities within several Web Application, Database and File systems in the open community as well as at two unnamed private companies including both traditional VM based Web Applications. and Containerized Web Apps to track the rate of remediation and the number of patches/updates that were able to be done in each period relative to the amount of time and human work involved. The

VMs and Databases were configured with auto-updates that do not require restarts enabled. For one of the companies, this was managed through AWS SSM Patch Manager. For another, it was simply done through the system settings. For the Docker-based systems, pull requests were used to modify the dockerfiles. These pull requests were triggered automatically but reviewed by developers and were deployed with the agile release cycle.

III. Empirical Evidence

Analyzing the vulnerability data for over 400 hosts and Docker images over the past six months, we observed average remediation of OS and Package vulnerabilities from VMs hosted on AWS within a mean time to remediate (MTR) of 109 days even when configured with AWS SSM Patch Manager consistently reporting successful weekly compliant runs. Alternatively, we saw remediation of upgradeable/patchable container-based vulnerabilities within around one month. We also noticed that of those vulnerabilities on hosts or in containers for more than 30 days, over 70% were weaponized in some form of malware. We did observe some vulnerabilities the remained open through the observation window

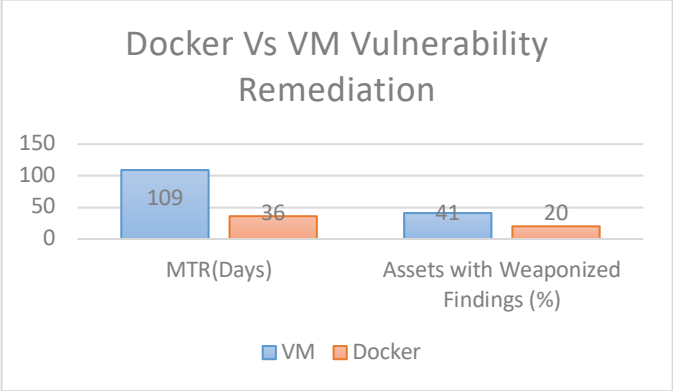

Figure III-1

As shown in Figure III-1, the remediation time for shift-left and container-based systems was around one-third of that for traditional systems and had half as many vulnerable assets.

IV. Conclusions and Future Goals

While there were fewer vulnerabilities, and they tended to be resolved more promptly in containerized platforms. I think this is due to the shift to stateless architecture designs that can allow for resource cutovers. The biggest delay for many VM resolutions was that the restart requirement required an approved maintained window to be performed in production. These were at fixed intervals, and given that the vulnerabilities were not critical, the teams likely felt comfortable waiting for an upcoming period. Given that so many of these CVEs were open for longer than a month are associated with malware, it may help emphasize the compounding interest that vulnerabilities have over time. By making pull requests in the development process for system changes, teams were able to ensure that their changes meet all the QA requirements as software changes would while keeping system security as close to the pace of development as possible. These updates take around 2 to make it to production, which may seem like much time for a single line change in a file. However, it is significantly faster than was observed for more traditional infrastructure and development models. Another benefit of implementing system-level changes through developer-involved pulled requests is that developers become more aware of their practices.

Moving forwards, we will explore if this approach is specific to containerization, and the Dockerfile or similar "shift left" approach could be used with other system types.

References

[1] Agirre, "Safe and secure software updates on high-performance embedded systems," 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2020, pp. 68-69, doi: 10.1109/DSN-W50199.2020.0002

[2] J. Wenhao and L. Zheng, "Vulnerability Analysis and Security Research of Docker Container," 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), 2020, pp. 354-357, doi: 10.1109/ICISCAE51034.2020.9236837.

[3] S. Garg and S. Garg, "Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security," 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), 2019, pp. 467-470, doi: 10.1109/MIPR.2019.00094.

[4] S. Potter and J. Nieh, "AutoPod: Unscheduled System Updates with Zero Data Loss," Second International Conference on Autonomic Computing (ICAC'05), 2005, pp. 367-368, doi: 10.1109/ICAC.2005.16.

[5] P. Kuacharoen, T. Akgul, V. J. Mooney and V. K. Madisetti, "Adaptability, extensibility and flexibility in real-time operating systems," Proceedings Euromicro Symposium on Digital Systems Design, 2001, pp. 400-405, doi: 10.1109/DSD.2001.952348.

[6] WhiteHat, "Renovate Docs"