Optimizing Service Delivery with Minimal Runtimes

Katharina Gschwind[1], Constantin Adam[2], Sastry Duri[2], Shripad Nadgowda[2], and Maja Vukovic[2]

[1] Massachusetts Institute of Technology
gschwind@mit.edu
[2] IBM T.J. Watson Research Center
{cmadam, sastry, nadgowda, maja}@us.ibm.com

Abstract. In this paper, we argue that deploying applications inside minimal runtime environments, which only contain the files necessary and sufficient for the application to run, can cut down the operating costs, specifically the costs for ensuring the application security and compliance. We identify a way to deliver minimal runtimes as Docker containers built from scratch. We describe a use case where minimal runtimes simplify the service maintenance operations, by reducing the number of updates for fixing security vulnerabilities.

1 Introduction

Keeping cloud applications secure is a complex process. Administrators need to check services and their underlying operating systems for vulnerabilities published on a daily basis by sources like Redhat security advisories ([1]), or Ubuntu security notices ([2]). They also need to harden these services and their runtimes against threats using benchmarks such as those defined by the Center for Internet Security (CIS) ([3]). In addition to the deployed application, these processes must be applied to the underlying OS, which in many cases is more complex than the application itself. The OS requires a large amount of configuration and updates, not necessarily related to the deployed application, to ensure that it cannot be used by an intruder to gain access to the Virtual Machine and/or Container and compromise it. Instead of implementing a process that secures the underlying operating system, we propose to eliminate it altogether.

In this paper, we argue that deploying applications inside minimal runtime environments, which only contain the code necessary and sufficient for the application to run, can cut down the operating costs, specifically the costs for ensuring the application security and compliance. We explore ways of automatically building and delivering such minimal runtimes, in the form of Docker containers built from scratch. We build containers from scratch for redis, a popular open-source application, run the original and the minimal redis image through a security vulnerability advisor, and show that minimal runtimes can reduce the number of re-deployments needed to keep up with various published security advisories.

2 Background

To secure an application, one must ensure that any vulnerabilities are remediated, and that the applications, as well as their underlying operating systems are configured properly. Vulnerability remediation and compliance enforcement are complementary actions that must be performed for several layers of software, depending on the type of the underlying runtime of a specific application. Figure 1 provides an illustration of the layers involved for virtual machines (cloud-enabled), regular containers (cloud-native), and minimal containers. Below, we describe in more detail the security and compliance procedures in place for each of these types of runtime.

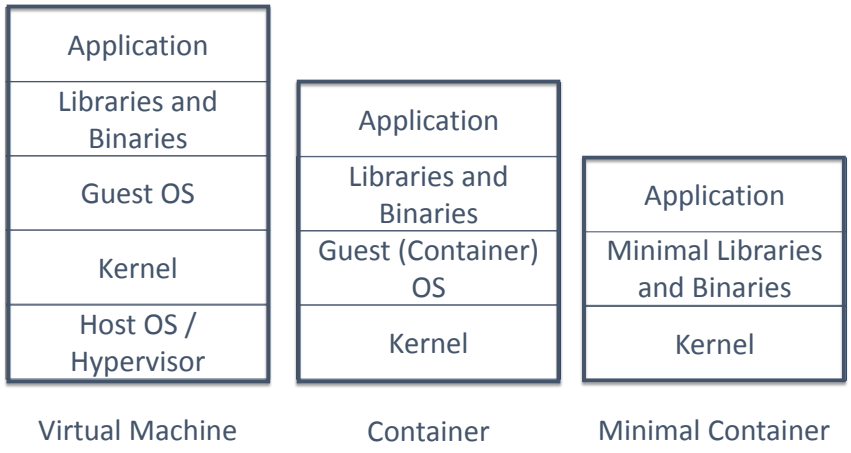

Fig. 1: Comparison of Virtual Machines, Containers, and Minimal Containers.

2.1 Cloud-Enabled Applications

A VM running a cloud-enabled application today contains a full OS image (Linux or Windows) hosting a primary user space application (e.g., MySQL or Nginx), along with secondary services (e.g., SSH, syslog or NTP). These VMs contain multiple packages, services or drivers that are not needed by their primary application, but that increase their attack surface, and reduce their performance. Vulnerability remediations might require a change window, and not happen instantly. Performance-wise, patching can delay the system bootup (this happens frequently on Windows servers). Integration with other systems (user control, monitoring, inventory, patching, etc.) makes the task of enforcing compliance and remediating vulnerabilities on these VMs long and complex.

2.2 Cloud-Native Applications

For containers built from an operating system image, the inherent complexity of the underlying OS makes it hard for an administrator to know if a container is running compliant configurations, or non-vulnerable code. The OS and other services configuration is scattered across the file system, uses different formats; many packages installed in the container are inactive. The study in [4] shows that, in 2015, 64% of docker images in Docker hub had high profile vulnerabilities. The problem has been solved since then for the latest release of a majority of images, however, as new vulnerabilities appear, not only the latest release needs to be patched, but also all the already existing deployments. This is our main motivation to develop minimal runtimes. There is a common interest to reduce the trusted code base for application runtimes, with unikernels [5] advocating single address programming paradigms, or microcontainers [6] using statically compiled "go" applications or buiding containers from minimal Alpine images.

3 Implementing a Minimal Runtime

DockerSlim ([7]) is open-source software that creates minimal images including only the files necessary and sufficient to run specific applications. For each image, DockerSlim also creates Seccomp and AppArmor security profiles. It takes as input a "full" docker image built on top of an OS, and operates in three phases: initialization, monitoring, and image/security profile generation. During initialization, DockerSlim reverse engineers the dockerfile of the image provided, gathering information about volumes, exported ports, created users, etc. Next, it instantiates a container from the original image, modified to launch the docker-slim-sensor executable within. Finally, DockerSlim establishes communication channels to the instrumented container to send commands to the container, and receive monitored data. During monitoring, DockerSlim runs two sensors in the container: one tracks filesystem events and identifies the files and symbolic links needed to run the container, the other traces system calls and generates security profiles for the image. Finally, DockerSlim generates the minimal image and its security profile by processing the reports generated by the sensors.

4 Case Study

To demonstrate the advantages of a minimal runtime environment, we have downloaded a year-old ubuntu-based Docker image for Redis (version 3.2.2), and generated a minimal image using DockerSlim. We used the IBM Vulnerability Advisor (IBM VA [8], [9]) to analyze both the original and minimal images for package-level security vulnerabilities, published in the Ubuntu security notices.

We had to make two changes for the IBM VA to function with images built from scratch. First, to identify the security advisory service against which IBM VA should check packages, we changed the DockerSlim code to include a file that contains information about the OS for which the files were built. Second, as

the IBM VA takes as input package information, we needed to map individual libraries to packages. With these changes, IBM VA was able to process slim images as though they were full images. Note that since IBM VA works at the package level, and a package consists of multiple files, IBM VA may report a vulnerability for a file that is not used in the minimal image. Further analysis of individual libraries in a slim image is needed to determine whether a package vulnerability is present in it or not.

We found that the original image had 10 vulnerabilities, in 6 vulnerable packages. The vulnerable packages were not necessary for the redis application to function, therefore none of them was included in the minimal image generated from scratch. Deploying a minimal container, at the time of the release of the image, would have avoided dealing with 10 different security vulnerabilities, and potentially as many re-deployments of the redis application.

5 Conclusion and Future Work

We have shown that minimal runtimes can cut down the number of updates that address security vulnerabilities. We will improve upon the DockerSlim method of generating images, and will conduct a large-scale evaluation of its security benefits on the set of Docker images with more than one million downloads. To ensure the completeness of the minimal runtime, we will add static analysis to the dynamic analysis provided by DockerSlim, and will ensure that test suites for the applications are automatically invoked during the building process.

References

1. Red hat customer portal security advisories. https://access.redhat.com/security/security-updates/. Accessed: 2017-08-04.
2. Ubuntu security notices. https://www.ubuntu.com/usn/. Accessed: 2017-08-04.
3. Center for internet security. https://www.cisecurity.org/. Accessed: 2017-08-04.
4. Chris Van Tuin. A security state of mind: Compliance and vulnerability audits for containers. In *2015 Usenix Container Management Summit*, Washington, DC, 2015.
5. Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels: Library operating systems for the cloud. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 461–472, New York, 2013. ACM.
6. https://www.iron.io/microcontainers-tiny-portable-containers/. As of 2017-08-15.
7. Dockerslim (docker-slim): Optimize and secure your docker containers (free and open source). https://dockersl.im/. As of 2017-08-15.
8. Byungchul Tak, Canturk Isci, Sastry Duri, Nilton Bila, Shripad Nadgowda, and James Doran. Understanding security implications of using containers in the cloud. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 313–319. USENIX Association, 2017.
9. Fábio Oliveira, Tamar Eilam, Priya Nagpurkar, Canturk Isci, M Kalantar, W Segmuller, and E Snible. Delivering software with agility and quality in a cloud environment. *IBM Journal of Research and Development*, 60(2-3):10–1, 2016.