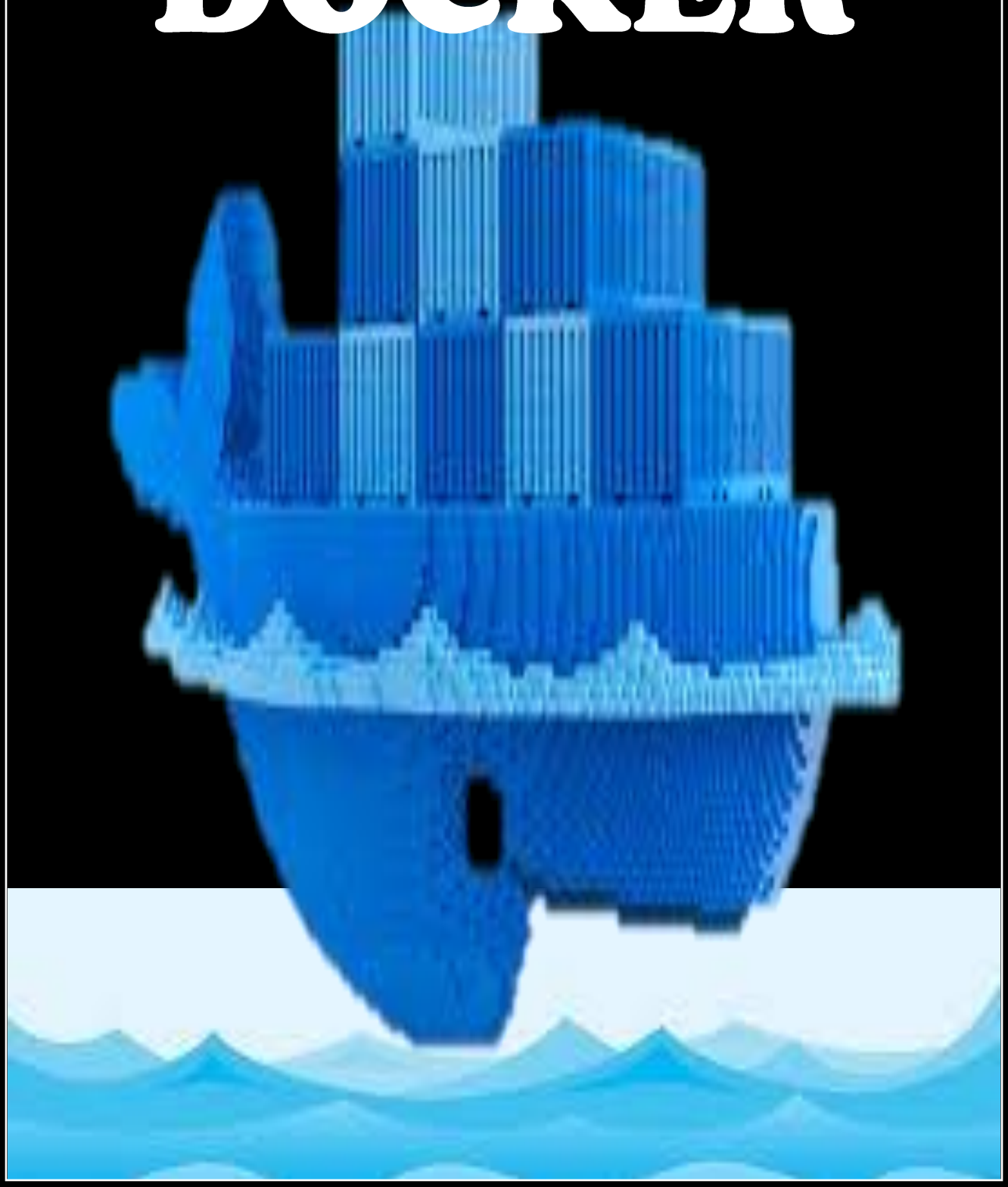
DOCKER



IT17049832 – H.K.N.Dabare

IT17515672 – Senanayaka L.M.K

Table of Contents

Product Overview

Docker is an open source venture to mechanize the arrangement of utilizations as convenient, independent compartments that can run on cloud or on premises. Associations look to Docker to streamline and quicken their application improvement and deployment process. It runs on various stages like cloud, Linux and windows merchants including Microsoft. This helps developers and system administrators to effortlessly run containerized applications. At the core is the Docker Engine, a lightweight application runtime with built-in highlights for coordination, scheduling, systems administration and security highlights to fabricate and send single or multi container applications [1].

Docker was founded by Solomon Hykes and Sebastien Pahl during the Y Combinator summer 2010 startup incubator group and launched in 2011. Hykes started this project in France as an internal project within .dotcloud. It was launched in 2013.

Docker works by giving a standard method to run client code as it is a working framework or OS for holders. This is like how virtual machine virtualizes server equipment.

We can utilize this Docker to dispatch code quicker, normalize application tasks and consistently move codes and also can set aside cash by improving asset use. There is a robust ecosystem of instruments and off the shell applications that are prepared to use with docker. This makes it simple to assemble and run conveyed small scale administrations structures, send the code with normalize consistent joining and make completely oversaw foundation of designers.

There are lots of factors to love Docker. Some of them are,

- **Open source** – as mentioned above it is an open source software which can be use in both linux and windows.
- **Security through sandboxing** - Each application is detached from other application because of the idea of Docker containers, which implies they each run in their own however associated eco system
- **Faster development and Easier iteration**- Since wide scope of conditions can be made, repeated and expanded, API s can be created to work with various frameworks.

- **Lightweight Reduction of redundancy** – By the idea of docker container, API share base kernel. This implies less system assets devoted to excess conditions.

In March 2017, Docker discharged Docker Enterprise Edition (EE), blending their past big business offering of Docker datacenter and remaining their free contribution to Docker community edition (CE).

Docker CE is a free and open source containerization stage. It is rebranded rendition of Docker open source arrangement that has uninhibitedly accessible since the dispatch of Docker in 2013.

Subsequent to discharging Docker CE, Docker chose to change the way they formed the product. Before March 2017, most recent form of docker that was accessible at time was v1.13. After arrival of CE and EE new form was reffered to as 17.03. It appeared that docker has avoided past 16 forms but in really they changed their forming plan to year and month which is like plan use by accepted for Ubuntu.

This can be run on windows 10 and mac, on Azure and AWS just as CentOS, Debian and Ubuntu. This can be download legitimately from Docker Store.

Docker CE accompanies two discharge channels known as "Edge" and "Stable". Edge channel discharges another form in consistently with new functions. Be that as it may, in stable form client need to stand by prolonged stretch of time to get the new features as new discharge accessible in each quarter. This is a lot simpler to keep up than Edge.

Edge channel just discharges bug fixes and security patches during the forms current month. Stable channel discharges patches for security issues and bug fixes for 4 months after introductory discharge.

Anyway Docker CE is as a rule free release, official vendor support isn't accessible as it accompanies network bolster as it were. This isn't quickest technique for help. That said advancement guide of network release additionally originates from the open source network.

Product Asset

Docker containers utilize shared working frameworks. This implies they are significantly more productive than hypervisors in framework asset terms. Rather than virtualizing equipment, containers lay on a single Linux instance. With a perfectly tuned container framework, you can have upwards of four-to-multiple times the quantity of server application cases as you can utilizing Xen or KVM VMs on a similar equipment. Another motivation behind why compartments are famous is they lend themselves to Continuous Integration/Continuous Deployment (CI/CD).

This approach intended to urge developers to incorporate their code into a common vault early and regularly, and afterward to send the code rapidly and productively. Docker empowers developers to effectively pack, ship, and run any application as a lightweight, compact, independent

compartment, which can run essentially anyplace. Security is high in docker network similarly as with significant admonitions, isolating the various segments of a huge application into various containers can have security benefits: in the event that one container is undermined the others stay unaffected.

Docker characterizes an API for mechanizing and automating the creation and sending of containers.The API is typically changed in each arrival of Docker. So API Calls are formed to guarantee that users don't break. For docker 17.06 the API form is 1.30. In past renditions of Docker it was conceivable to get to the API without giving adaptation. Programming interface utilizes an open schema model which mean server may add additional properties to responses.

Docker give an API to communicating with the Docker daemon called Docker API Engine just as SDK for GO and Python. In Docker community edition has use API rendition 1.26. This is called as Engine API. This was utilized to communicate with the engine by user. So everything that docker customer should be possible with the API.

Managing credentials, access tokens and private keys in an application can be dull. In any event, putting away such things in docker images can be effectively available. In the event that we run picture in interactive container mode all application code is accessible in the container. Therefor docker give docker secret to secure every single sensitive datum. This is mass of information that may comprise of secret phrase and some other sensitive data. Docker secret deal with every one of these information and safely transmit information to containers that need to get to them. This is encrypted over transport and just open to conceded containers. This works just in swarm administrations and not accessible to independent containers. At the point when a Docker secret is made, the sensitive data is transmitted from Docker to its swarm manager where it is put away in a pontoon log, which is scrambled. That encrypted log is then coursed to the next supervisor to have higher availability over the swarm. Docker administration can get to the secret by mounting a encrypted area to it.

Inside the docker container email setups should be possible utilizing host-based-postfix, qmail, Exim and sendmail. There is no simple method to give an email setup in docker other than making a config record and tie it physically. Moviemasher.rb is used to encode mashup recordings, sound, and images in docker.

Keystores are set on an external docker volume so they can be refreshed without altering the docker image. This takes into account refreshing keystores autonomously from docker images in the event of a security break or for an ordinary key/authentication turn/invigorate.

Greater part of the

standard browsers being compromised, featuring that we despite everything have some best approach for a safe browsing experience. It is commonly acknowledged that one of the best approaches to improve the security of how we peruse is utilizing compartmentalisation. To put it plainly, this idea includes segregate different programming parts into isolated containers so that, if one compartment is undermined, it confines the exposure of the penetrate.

One project that has driven the way is arrangement a program in Docker on OS X. Utilizing GUI applications in Docker essentially depends on some way passing the display from the container to the host OS. The standard methods for accomplishing this are sharing the X11 attachment, utilizing X11 Forwarding or utilizing VNC. For this we can utilize one of the numerous pre-assembled image made by the Docker people group. We can utilize sameersbn/docker-program box to open numerous internet browsers simultaneously.

Docker needn't bother with any arrangements to confirm docker condition, just one order line was there for the check. In any case, it isn't prescribed to run "exposed" docker creation. Docker security group together with different organizations discharged CIS Docker 1.11.0 benchmark which is tied in with solidifying and make progressively secure docker CE

Docker Bench for Security is a lot of Bash shell contents, that you should run as a root client on any Docker have and the tool will deliver an extremely decent looking and beautiful arranged yield with all checks. The most recent benchmark for Docker CE 17.06 is (CIS Benchmark v1.1.0). This benchmark is just material to the Docker Community Edition Engine and doesn't include a large number of the security abilities intended to enable organization to fulfill significant consistence prerequisites and that which are given by the total Docker Enterprise Edition stack.

The sandbox permits to arrange and attempt trust activities locally without affecting to creation image. In the event that you are simply utilizing trust out-of-the-crate you just need your Docker Engine customer and access to the Docker Hub. The sandbox copies a creation trust condition and sets up these extra segments.

Container	Description
trustsandbox	A container with the latest version of Docker Engine and with some preconfigured certificates. This is your sandbox where you can use the docker client to test trust operations.
Registry server	A local registry service.
Notary server	The service that does all the heavy-lifting of managing trust

For the sandbox, however, construct own whole, mock production environment.

Inside the trustsand box container, you collaborate with your neighborhood library(local registry) instead of the Docker Hub. At the point when you play in the sandbox, you likewise make root and repository keys. The sandbox is arranged to store all the keys and documents inside the trustsandbox holder. Since the keys you make in the sandbox are for play just, obliterating the container destroy them too.

By utilizing a docker-in-docker image for the trustsandbox container, you don't dirty your genuine Docker daemon reserve with any image you push and pull. The images are store in an unknown volume joined to this container and can be demolished after you destroy the container.

Overview Findings

Docker has a focused approach on security as it directly deals with main computer resources such as memory and processor usage. This also means that most of the functions of Docker need to be run with a high level of
(or the highest

level of) privilege. Thus, intermittent security is a must for such as a software. The security features Docker has is as follows.

1. The embedded security of the kernel as well as the support for namespaces and cgroups.

   Namespaces are initially created when a docker container is run. This creates the first level of security in the form of isolation. This means that containers are independent of each other. Each container also gets their own network stack, further isolating them from attacks. This means that a container does not get **privileged access** to the sockets or interfaces of another. It still can be set to interact with other containers, though.

   The kernel, as well as the concept of namespaces, were introduced in July 2008. Since then, the function has been used and tested on various systems throughout time, and the code has matured well since.

   Cgroups or control groups are another key component of containers. These groups implement Resource accounting and limiting. This allows them to fairly share resource, such as memory, CPU and disk I/O, while making sure no one container has enough resources to bring the system done by totally exhausting a resource. This is essential in defending against Denial-of-service attacks. This code, too, has matured since 2006.

2. Docker Daemon Attack Surface

   The Docker Daemon runs when running docker containers, or even the application itself. This means that the daemon requires root privileges to run many of its functions. Docker has implemented a few security features to protect against abuse of these privileges in regard to the Daemon. One such feature is using chrooted sub-processes when loading or pulling docker containers, which works toward privilege separation. A cryptographic checksum for docker containers were also introduced, which limits the possibility of attacks and increases integrity.

   Other daemon defenses which are used include using a TCP socket bound on 127.0.0.1 instead of a REST API, because REST API are more susceptible to cross site request forgery attacks. This will also allow one to limit access to the control socket. If a REST API is used, it is mandatory to secure the API endpoints with HTTPS and certificates. It is also possible to use SSL instead of TLS, and this allows for an additional layer of security.

3. Linux Kernel Capabilities

   By default, Docker starts containers with a restricted set of capabilities. The privileged accesses are fine grained to a level where only functions that are essential to be run as root are given root level privileges. This is because of the threat of privilege escalation as well as privilege misuse, which docker has taken into consideration. Docker tries to use

the least privilege possible for many of its functions (Least privilege design principle). A few examples are given below.

i. Running **cron** as a user process tailored specifically for certain applications which need scheduling
ii. Graniting **net_bind_service** to bind ports below 1024
iii. Network management is done outside the containers, invalidating commands such as **ifconfig** or **route**, except when explicitly configured to work as a network device.

4. Docker Content Trust Signature Verification

The Docker engine can be configured to run only signed images. This feature is directly built into the **dockerd** library.

Number of Vulnerabilities

Docker is a software which has been there for a long time. As such, there have been many vulnerabilities. The CVE database mentions of many vulnerabilities in various Docker versions which have been mostly patched in the current version. Of these, there have been three vulnerabilities with a Common Vulnerability Scoring System (CVSS 3.1) score of above 9, with one vulnerability reaching the score of 10; this vulnerability was where arbitrary code could be executed with root level privileges in a crafted docker image. These vulnerabilities were recorded until 2019, and more vulnerabilities are being found up to date.

For the purpose of this assignment, we will be looking into 4 vulnerabilities

i. CVE-2019-5736

Runc (The lightweight linux platform for container management) allows attackers to overwrite host runc binary and pbtain host root access by leveraging the ability to execute a command with specific types of containers

ii. CVE-2019-1020014

Docker-credentials-helpers before 0.6.3 has a double free in the List functions.

iii. CVE-2019-15752

Docker Desktop Community Edition before 2.1.0.1 allows local users to gain privileges by placing a Trojan horse docker-credential-wincred.exe file in **%\DockerDesktop\version-bin** as a low privilege user, and then waiting for an admin or service user to authenticate with Docker, restart Docker or run **docker login** to force the command

iv. CVE-2017-14992

Lack of Content verification in the Docker Engine allowed attackers to cause a DOS attack by a crafted image layer payload. This is like gzip bombing

Example Attacks

As said before, Docker deals with container management, and is able to access most of a host system's resources. As such, it is targeted by attackers, and these attackers have found many attack routes. These attack routes can deliver various types of attacks as well.

Docker is mostly a CLI application. It is also based on the Operating System. As such, it does not have a web interface (though it can be connected to various online third-party web application). This means that attacks through the web is limited, but they do exist. A few examples are

i. Attacks through REST API, TLS socket, or even SSL when Docker is hosted online
ii. Poisoning online containers which could be pulled or loaded into Docker
iii. Attacks through online hosted third-party applications (Jenkins)

These routes can be utilized to launch various attacks on the Docker software. The main security concern in Docker would be using Docker as a route to carry out an attack on the host. There are several vulnerabilities which have shown this is possible. A few possible attacks are as follows.

i. Running commands through Docker on the host (commands to open ports or download malicious software

ii. Running malicious scripts or software within Docker's CLI. This can be done by hijacking specific Docker commands (ex: Setting a script in file which must be read when a certain docker command is run

iii. Hijacking Docker credentials. This will allow an attacker to directly use Docker for malicious activity within a host.

Since Docker has been around for a long time, it has updated its security to patch most vulnerabilities. The large number of vulnerabilities is expected of a sophisticated software. The OWASP top ten vulnerabilities have been of concern for Docker through time, even though it is not a pure web application.

Vulnerabilities through third party software and frameworks are also consistent. RunC is such a library which is constantly exploited. This is also an integral part of the Docker Engine. Docker also supports various other third-party applications and libraries which introduce different vulnerabilities. Jenkins, Pallets-Werkzeug and Gitlab are a few examples.

Vulnerabilities

Note: CVSS vector : CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

First vulnerability

CVE:	CVE-2019-5736
Severity:	8.6 (high) according to CVSS:3.1
Founder:	Dragon Sector
Date:	2019-02-11
Brief Idea:	Runc (The lightweight linux platform for container management) allows attackers to overwrite host runc binary and obtain host root access by leveraging the ability to execute a command with specific types of containers
Bounty:	How exploit works: The host binary is exposed to the container. This means that an attacker could use the container to run commands on the host. This vulnerability presents two different methods of exploit. [2]
i.	The first way is to set a trap through a trojan horse. This trojan horse should start a malicious binary which would listen to user inputs. When the trap is triggered by running **docker exec**, it will trigger the malicious binary which will run as root (because **docker exec** needs root privileges)
ii.	The second way is by creating a malicious Docker image. This will act as the trojan horse, and it only needs to be run to activate the malicious code. (**docker run** must be used to run the image, which will trigger the malicious code within)
How devs fixed it:	The developers wrote a separate file to avoid the exposure of the host binary to the container. This file is a readonly copy of the file which can be exploited. Thus, there is no way for an attacker to write any malicious code on or through this file. [3]
When:	11th February 2019
Patch no:	Docker Community Edition 18.09.2 [4]
Comment:	This issue speaks to the vulnerabilities inherited by using third party libraries. A look into how the issue was found shows that the reason this affected Docker so much was because Docker had utilized the runC code to run an important code such as **docker exec**. Although this is not wrong in any way, more attention should be given when using third party libraries, especially from a security standpoint. [5]

Second Vulnerability

CVE:	CVE-2019-1020014
Severity:	5.5 (medium) according to CVSS:3.1 [6]
Founder:	Docker
Date: -	
Brief Idea:	Docker-credentials-helpers before 0.6.3 has a double free in the List functions
Bounty: -	
How exploit works:	The problem lies with a pointer which has been released twice. This extra free call has a possibility of crashing the program as the memory management data is corrupted. In some circumstances, the extra free releases another malloc (memory allocation). If an attacker is able to identify which memory is free, the system will be vulnerable to a buffer overflow attack.
How devs fixed it:	Controlling the free double. (In a c file, the free was removed, while the free double was set in a conditional within a go file) [7]
When:	July 16th 2019
Patch no:	Docker-credentials-helper v0.6.3 (released with Docker Desktop 2.1.0.1) [8]
Comment:	Security experts working on Docker have swiftly dealt with even farfetched issues such as free pointers. This shows the interest of Docker towards a secure software

Third Vulnerability

CVE:	CVE-2019-15752
Severity:	7.8 (high) according to CVSS:3.1 [9]
Founder:	Morgan Roman [10]
Date:	July 5th 2019
Brief Idea:	Docker Desktop Community Edition before 2.1.0.1 allows local users to gain privileges by placing a Trojan horse docker-credential-wincred.exe file in **%\DockerDesktop\version-bin** as a low privilege user, and then waiting for an admin or service user to authenticate with Docker, restart Docker or run **docker login** to force the command
Bounty:	
How exploit works:	The exploit runs a Metasploit script which puts a trojan horse in a Docker folder. This trojan horse will be executed whenever a user tries to authenticate into Docker. If this authentication is done by users with admin or service level privileges, the script too will be run with such privileges, essentially acting as a privilege escalation.
How devs fixed it:	Adding various security restrictions to the type of files which can be run, specifically to linux tools.
When:	2019 -08 -08
Patch no:	Docker Community Edition 2.1.0.1
Comment:	As identified by the founder of the issue, the line of code responsible for the possible exploit still remains, which could reintroduce a similar attack using a different attack vector (As at April 28th 2020).

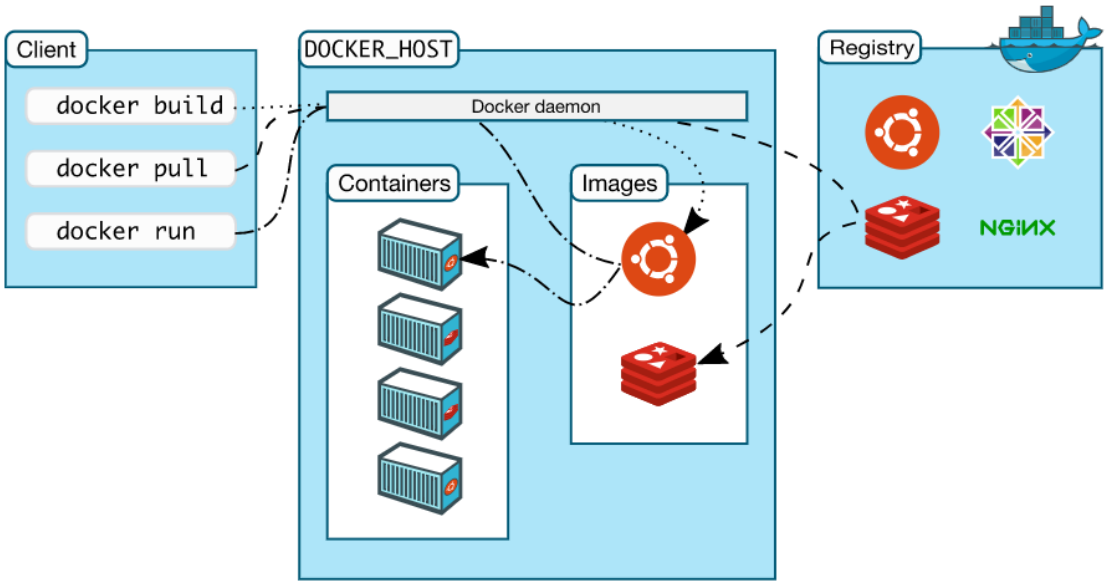
Fourth Vulnerability

CVE:	CVE-2017-14992
Severity:	6.5(Medium) according to CVSS:3.1
Founder:	n4ss
Date:	October 17th 2017
Brief Idea:	Lack of Content verification in the Docker Engine allowed attackers to cause a DOS attack by a crafted image layer payload. This is like gzip bombing.
Bounty:	
How exploit works:	A certain line of code can read any number of end line notations (\0). This can be used by attackers by compressing a large amount of such characters and then pushing or pulling the file as a docker image. When the docker image is imported, it will be decompressed and the characters will take up most of the memory, essentially crashing the device. [11] [12]
How Devs fixed it:	Added tar header validation. The padding (\0s) were checked and proper validation was done. The validation contained an End-of-file check, as well as error logging and stopping the process in case of an attack (essentially stopping any attack from this attack vector)
When:	November 7th, 2017
Patch No:	tar-split v0.10.2 (Docker Engine 17.11) [13]
Comment:	The issue signifies the severity of even one line of code when security has not been applied.

Chapter 2 – Design Analysis

Architecture Overview

Figure 2.1: High Level Diagram [14]



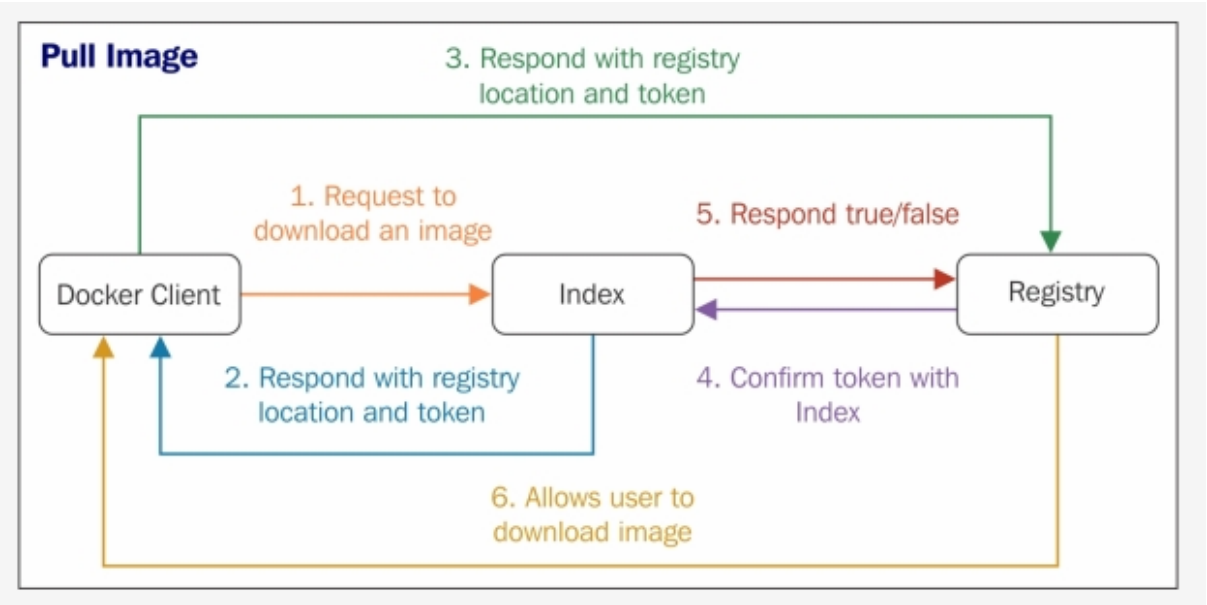
In the above image, Docker is split into three layers; the Client, Docker host and Registry. The Client is the command line interface in which a user can input various commands. The Docker host is the virtualization layer on which containers are run. These containers can be loaded with Docker images. These images can be pulled from the Docker registry, which is the third component. The Docker daemon is the manager of all these components.

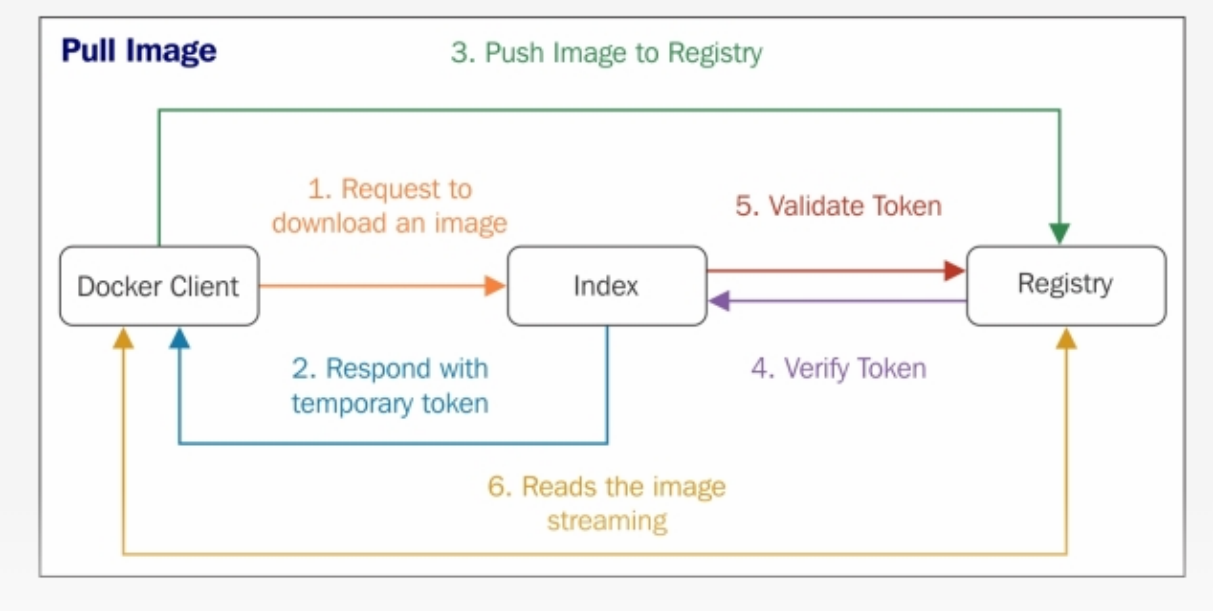
Main functionalities

Docker is a lightweight application virtualization software. The various combinations of applications have increased the demand for certain functionality. Docker currently supports many applications, and comes with various plugins, bringing their own functionality into Docker. Though this is the case, Docker still maintains three main functionalities to run their service, and this can be seen in the high-level architecture. They are

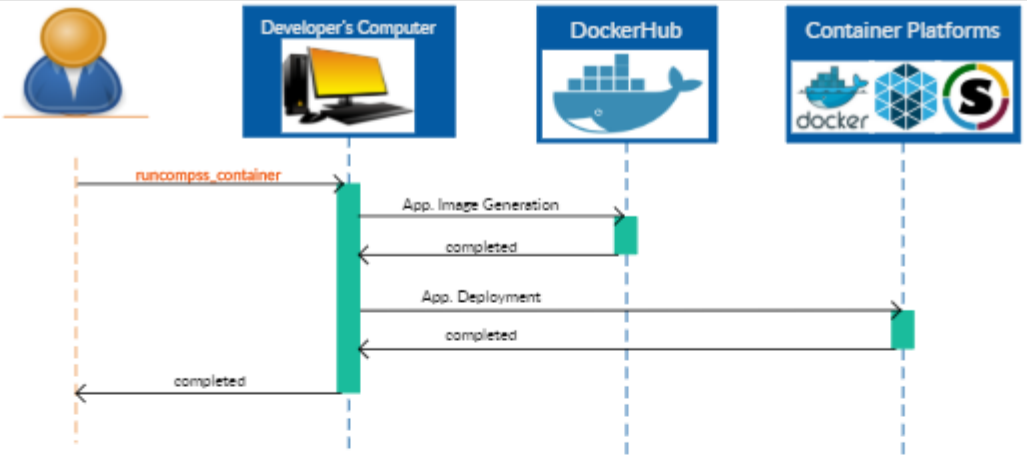
i. Docker build : The functionality to build an image
ii. Docker pull: The functionality to pull an image from a registry
iii. Docker run: The functionality to run the image by activating the container.

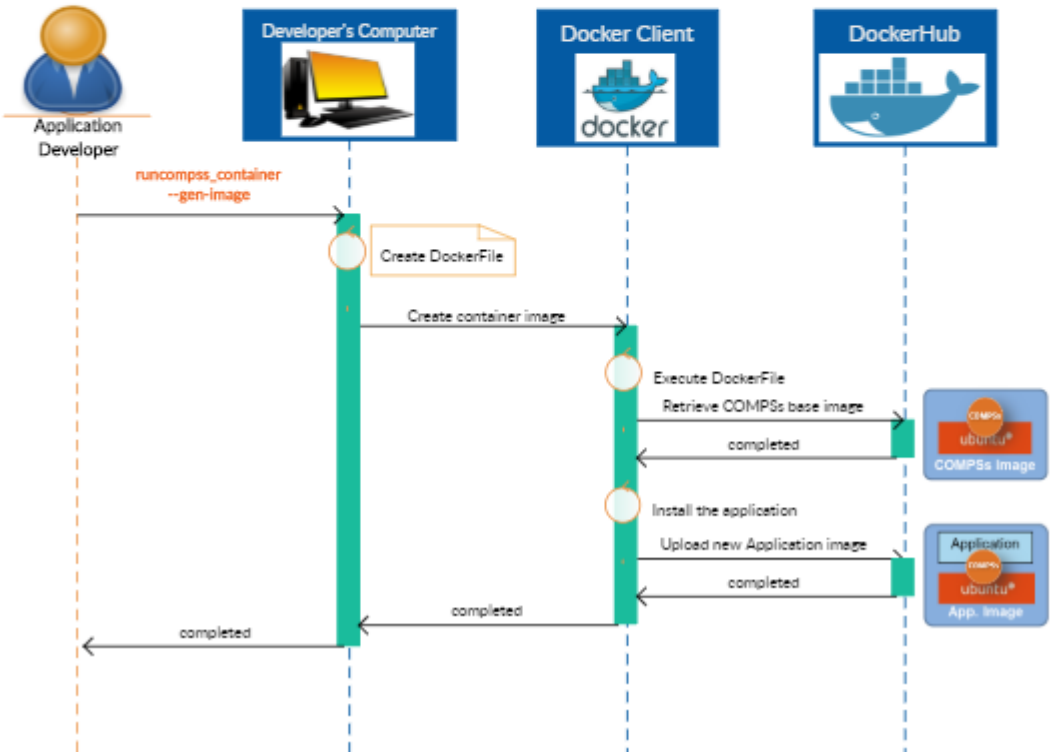
Use Cases


Pull Image
3. Respond with registry location and token
1. Request to download an image
5. Respond true/false
Docker Client
Index
Registry
2. Respond with registry location and token
4. Confirm token with Index
6. Allows user to download image


Pull Image
3. Push Image to Registry
1. Request to download an image
5. Validate Token
Docker Client
Index
Registry
2. Respond with temporary token
4. Verify Token
6. Reads the image streaming


docker


Developer's Computer
DockerHub
Container Platforms
runcompss_container
App. Image Generation
completed
App. Deployment
completed
completed


Application Developer
Developer's Computer
Docker Client
DockerHub
runcompss_container --gen-image
Create DockerFile
Create container image
Execute DockerFile
Retrieve COMPSs base image
completed
COMPSs Image
Install the application
Upload new Application image
completed
Application
App. Image
completed
completed


docker

Basic Level Security and Design principles

In Overview Findings, the security features offered by Docker have been discussed. This section will talk about the intrinsic security features of Docker, as well as the design principles [15] which have brought about these security features. [16]

1. File System Protections

For Docker to run successfully, some linux kernel processes must be loaded onto the container environment. These processes are loaded as 'read-only' mount points, which means no user can write on these processes. This protects the container and host from malicious code execution as well. The design principle behind this is **separation privilege design principle**.

Docker also uses Copy-on-write file system. This means the same image can be used as the base for multiple containers, but when content is written to the image, it is written to a container-specific file. This means there is no sharing of the image file within containers, which is declared as the **least common mechanism design principle**.

2. Capabilities

Capabilities are a method given by Linux to implement access control on processes. This means that certain capabilities can be blocked to stop users from running certain processes. Since Docker uses the certain linux processes, they have removed certain capabilities. These include capabilities such as CAP_SYS_RESOURCE which contain processes for overriding resource limits, CAP_AUDIT_WRITE which allows writing to the audit log, CAP_NET_ADMIN which allows configuration of the network and CAP_SYS_ADMIN which is essentially root privileges. The design principle in accordance with this is **least privilege design principle** which has been used widely in Docker. These restrictions are set for security purposes and do not hinder any functionality. For example, networks can be configured outside the container (from the host) but it is not allowed within the container as the capability has been removed which is in line with **psychological acceptability design principle**.

3. Namespaces and Control groups

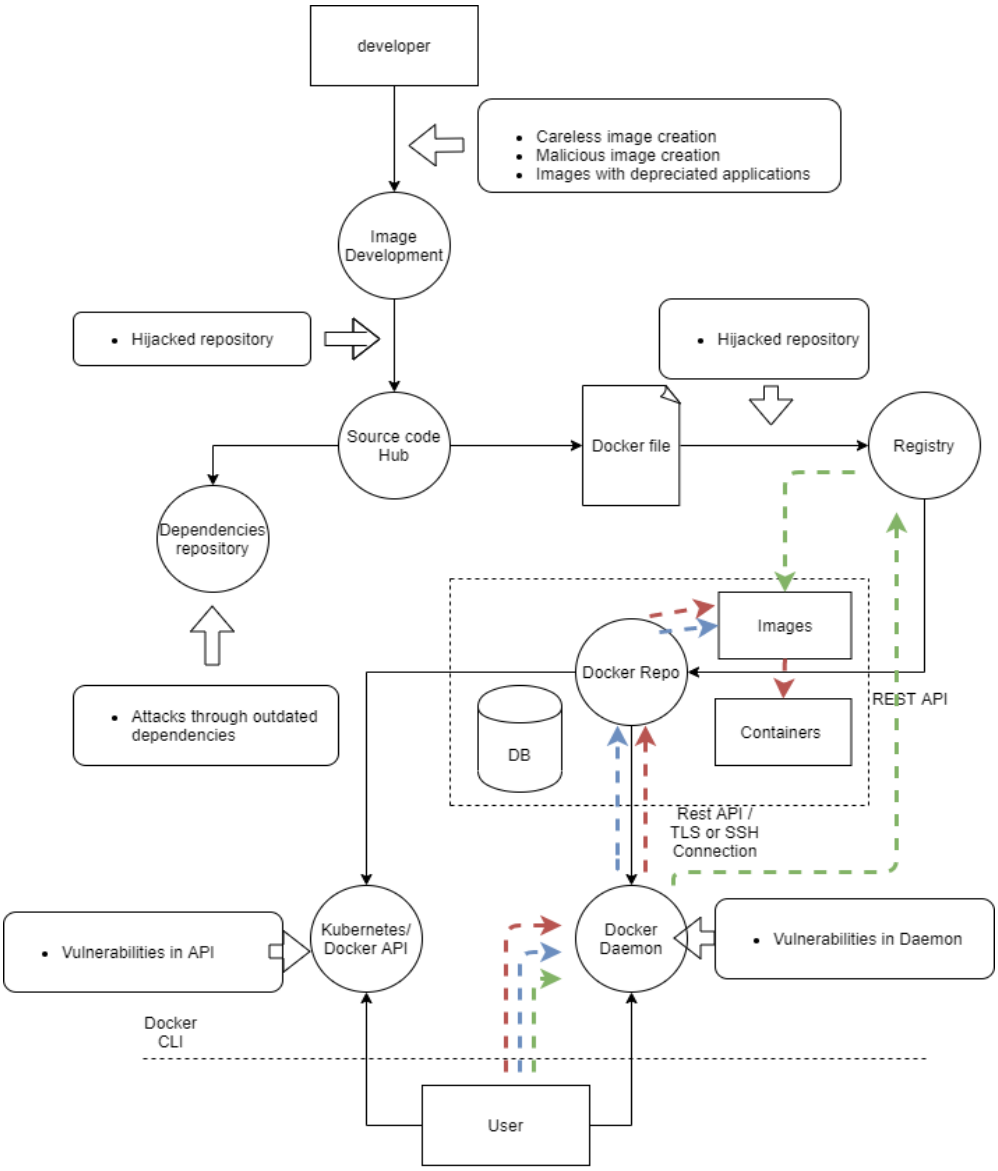
As explained in overview findings, namespaces and control groups (cgroups) are mechanisms given by the Linux kernel to separate containers disallowing them from using up all the resources of the host. Here, again, **separation privilege design principle** is used.

4. Open source code

The **open design security design principle** is clearly seen in Docker as every line of code in Docker is available to any user. This means that Docker is not dependent on secrecy of its design or implementation for the sake of security.

Threat model

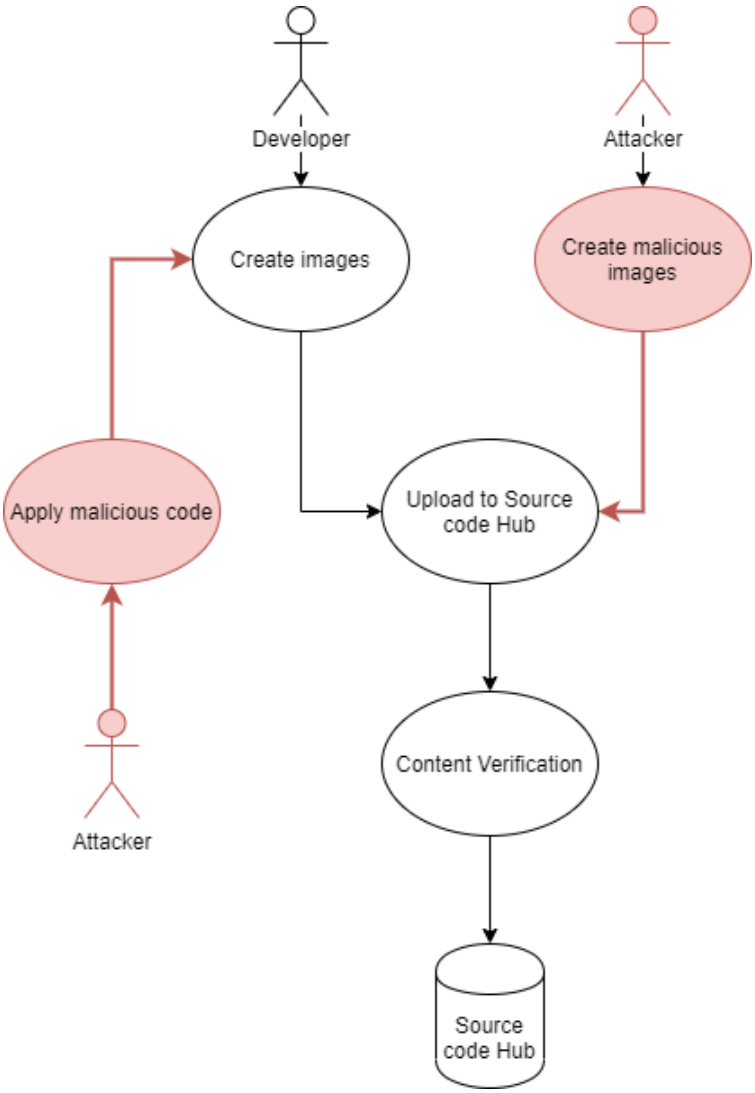
The threat model looks at the high-level architecture and any possible attacks within them. Threats to the main assets of the applications are assessed [17]. The main functionality has also been included in the diagram and shows any possible attacks through the main functionality as well.



Misuse/Abuse cases

Misuse/abuse cases have been identified for the above threat model. Four misuse cases will be modeled using these possible attacks, for specific assets.

1. Process: Image Development
   Asset: Source code Hub
   Threat: Malicious image crafting
   Vulnerability: Source hub has no strict content verification. This is as designed, though, because Docker files are created with various purposes including testing and security purposes, and thus content verification could cause a breakdown in functionality

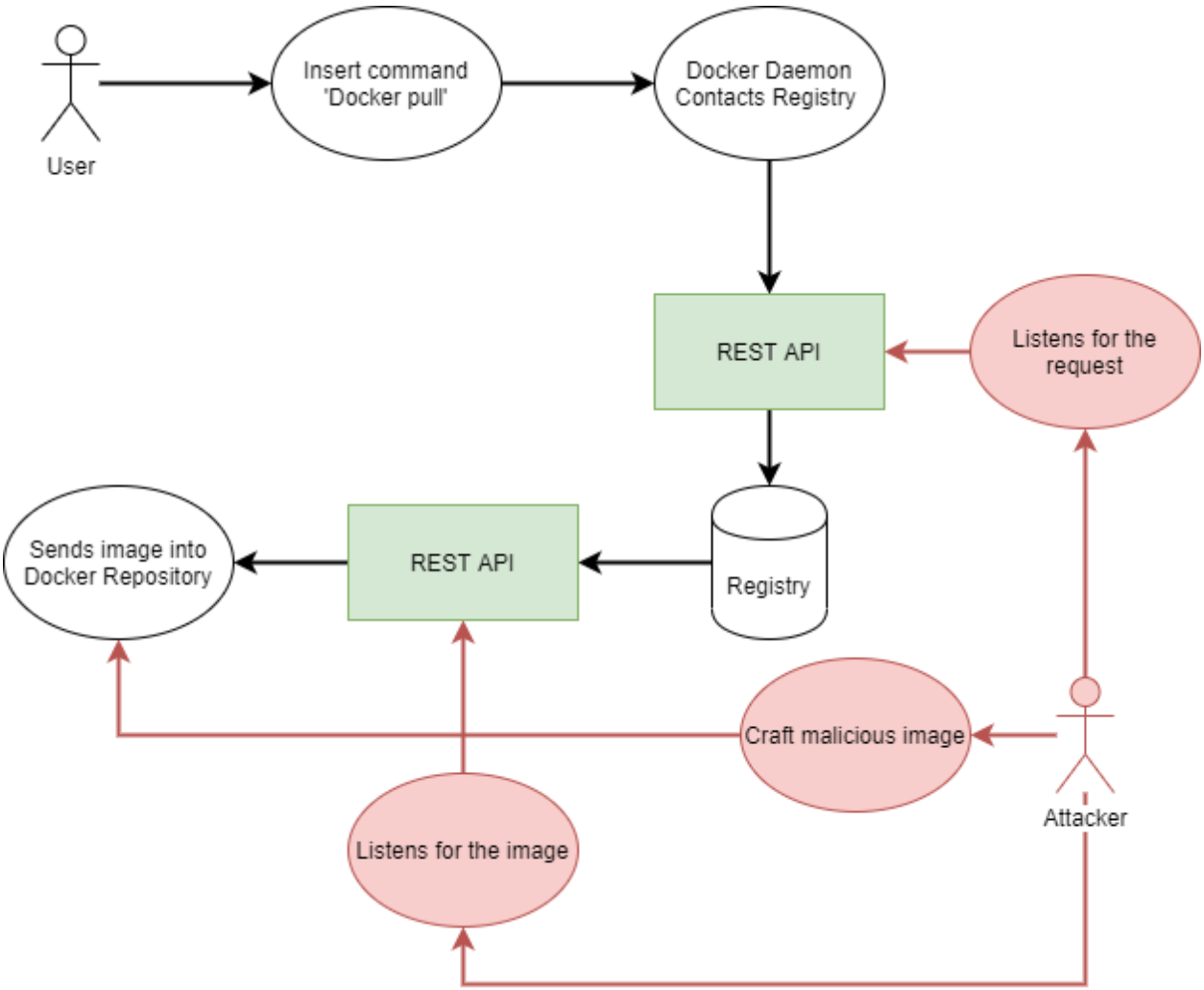


2. Process: Docker Pull

   Asset: Registry

   Threat: Man-in-the-middle attack when pulling images

   Vulnerability: Vulnerabilities in the REST API used

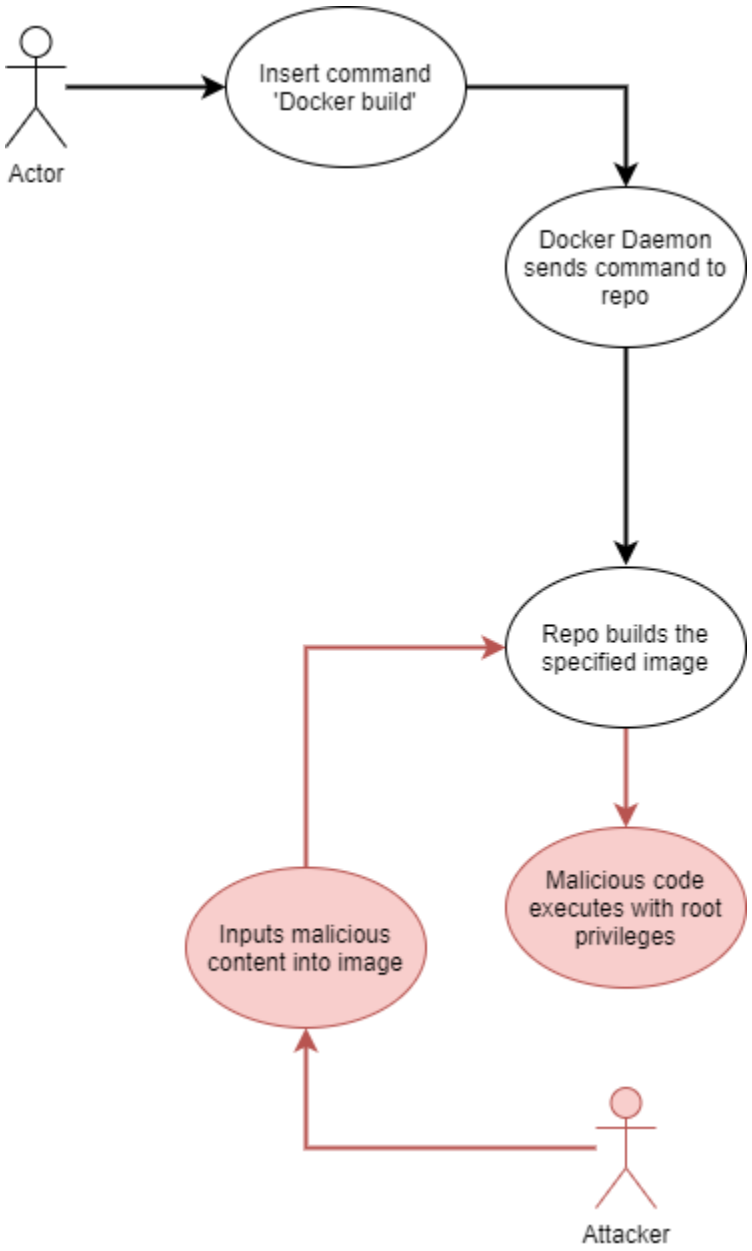


3. Process: Docker build

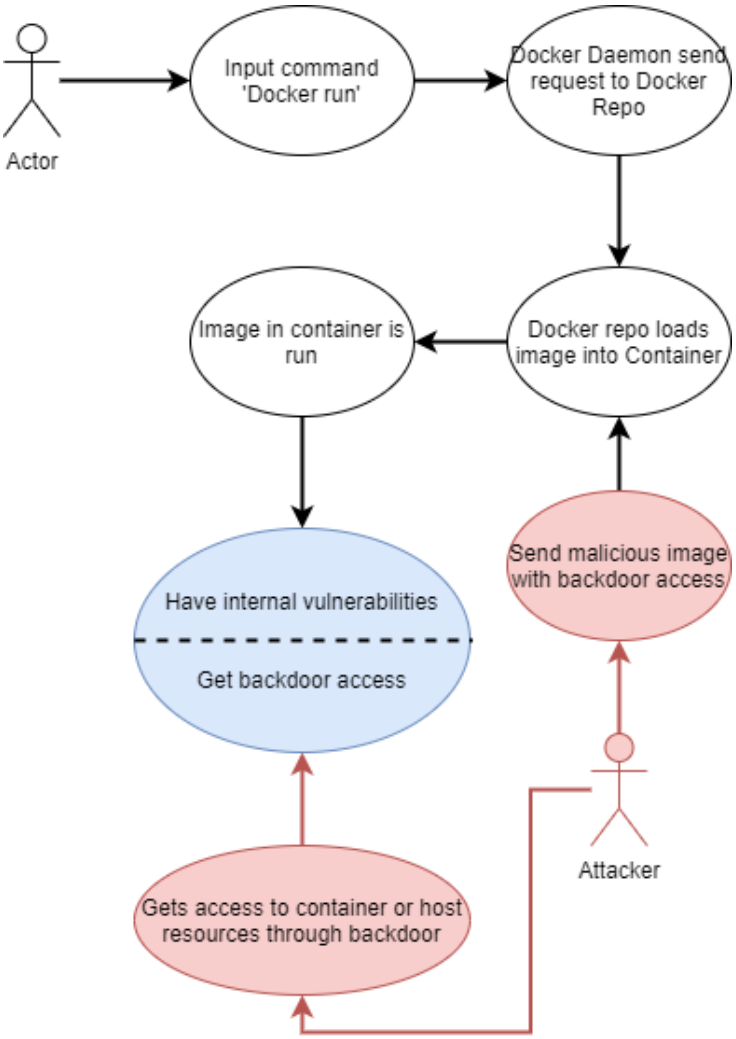
   Asset: Docker Repo (Conrainers) and Host

   Threat: Unauthorized access to processes which need root/high-level privileges

   Vulnerability: Docker build need certain root privileges to build the image.



4. Process: Docker run
   Asset: Docker Daemon
   Threat: Attacks through internal libraries
   Vulnerability: Internal library vulnerabilities



Chapter 3 – Code inspection

One

Selected File:	osxkeychain_darwin.c
Folder path:	docker/docker-credential-helpers/osxkeychain (github path)
Lines of Code:	227 as at April 30th, 2020
Reason:	Osxkeychain is the macOS mechanism used to store passwords and account information. The reason for selecting this file was because of the importance of account information control.
Basic Function:	The main functions used in the page are as follows. I. Keychain_add II. Keychain_get III. Keychain_delete IV. Keychain_list A few other functions are present in the page and are used in the above functions. It is evident that the above functions are the CRUD functions, except update, because from a security standpoint, updating these keychains could lead to a leak in confidentiality. One important note is the use of dynamic memory allocation, and the possible errors with no properly freeing this data. This code also uses server and host information as well, and developers have done a good job of not letting any data be free after they are used. Usage of error logging in a relatively small code page can also be seen as another good practice. Functionality wise, most code works with third party classes in creating and deleting objects (SecKeyChain related objects).
Security related functionality:	Since this code is related directly to software security, all functionality needs to be secure. Using the data dynamically and freeing the data after they've been used is a good security practice. This way, there is no remnant data, and it, on a small scale, may prevent DOS attacks.
History of	

Vulnerabilities:	This code had one vulnerability
Line no:	227(as at 16th July 2019)
Severity:	5.5 (medium) according to CVSS 3.1
Description:	Docker-credentials-helpers before 0.6.3 has a double free in the List functions
Error/Bug:	No issue reported specifically (merged along with the update)

Second

Selected File:	disassemble.go
Folder path:	Docker/engine/vendor/github.com/vbatts/tar-split/tar/asm
Lines of Code:	154 as at 30th April 2020
Reason:	This file has code pertaining to tar unzipping which is an area which should be looked in security. This was selected as an optimal file for code inspection because of this.
Basic Function:	The whole functionality of this code is to disassemble a tar file
Security related functionality:	In this code, security related functionality is mainly around controlling the tar unzipping without enabling any leaks in security.
History of Vulnerabilities:	One vulnerability was found in this file Line no: 124 as at October 4th 2017 Severity: 6.5 (medium)
Description:	There was a lack of tar header verification in the Docker Engine. This allowed attackers to cause a DOS attack by a crafting a tar zip with a lot of unwanted end-of-file characters.
Error/Bug:	No issue reported specifically (merged along with update)

Third

Selected File:	exec.go
Folder path:	engine/daemon/exec
Lines of Code:	147 as at 30th April 2020
Reason:	This code contains the implementation for 'docker exec', which essentially opens a new shell to be used. As such, the input and output must handled as it could be used by attackers to execute malicious scripts. The reason this was selected for a security code inspection is to learn how a software such as Docker will sanitize command execution.
Basic Function:	The main functionality pertains to execution, logging and input validation of 'docker exec' commands. Handling the input out streams is done concretely in this file (by controlling the stdio stream). This is done by holding a set of configurations, as seen in the 'Config' structure. A new struct object is created whenever 'docker exec' is run and is stored for logging purposes. The stream is also a part of the struct and is only open momentarily.
Security related functionality:	The design principles **Economy of Mechanism** and **Defense in depth** can be seen here as there are many layers of validation. The code works by opening the input stream to record the command and is closed immediately afterwards (using CloseStreams()). This will stop any attackers from adding any malicious code. These exec functions are also archived for auditing purposes.
History of Vulnerabilities:	None reported

Fourth

Selected File:	pull.go
Folder path:	engine/distribution
Lines of Code:	196 as at 1st May 2020
Reason:	This page acts as an intermediary when 'docker pull' command is executed. Since this is one of the main functionalities, it was thought of as a good file for code inspection.
Basic Function:	The functionality of this page is to facilitate pulling images. It is done by using an interface 'Puller' which will abstract pulling for different API versions (v1 or v2) . The function 'Pull' is the main function of this page and is responsible for pulling the Docker image from a registry.
Security related functionality:	The security related functionality in this page is the function for adding cryptographic checksums, or in this case, the digest using 'addDogestReference'. What the function does is check the store for a digest. If so, it would add it to the pulled image so that it can be verified.
History of Vulnerabilities:	None reported

Misuse and Abuse cases: Updating of digests is not supported, and by editing the checksum, an attacker could make it so that some images cannot be pulled.

Product specific Checklist

- Domain specific concerns

  Docker is a lightweight virtualization application which uses a linux kernel. This linux kernel uses a library name runC which constitutes of the internal implementation of Docker. RunC is widely used in other virtualization platforms as well and is the default runtime in Docker. Because of this, any vulnerabilities in runc directly affect the Docker runtime. As such, it is important to keep up to date about runc vulnerabilities.

- Coding mistakes

  Docker has a large code base, and a large pool of developers as well. These developers are from various backgrounds and specialize in various fields, as expected of an open source software. In such an environment coding mistakes are bound to happen. Sometimes, the code is not updated properly, as most developers are not looking into the Docker project only. A few coding mistakes identified are as follows.

i. Using depreciated functionality from libraries. The cause for this is because developers have forgotten to update the code. This can be seen in the service.py file at line 1788 (tempfile.mktemp()) which is depreciated. Instead, python recommend the use of mkstemp(), NamedTemporaryFile() and mkdtemp() which are much more secure. [18]

ii. Not using values which are assigned to local variables. This takes up unwanted memory, and since it is never used, an attacker could possibly use this variable to store malicious code and it will go undetected as it is not used in the application. This is the case with the variable 'version' in line 121 of the release.py file in the root/script/release folder.

iii. Incorrect conditioning. This is usually the result of a miscode and could possible lead to a security issue. This happens when an array index is compared with the length of an array, which, if not tested properly, might cause problems (because first element of array starts with 0, but length calculation starts from one and this may lead to an operation which is out of bounds)

- Design Concerns

  A few design concerns too exist in Docker. These might be slightly insecure but are essential for the application to work properly.

i. XSS vulnerabilities can be seen in a few places, especially http.go and blobwriter.go in the Docker distribution package. This is because the application directly writes into the HTTP response. Unfortunately, both are essential to the application. Http.go deals with all http request used in Docker distribution and Blobwriter.go handles operation with uploading BLOBs (binary large objects)

ii. Overflow possibilities are also present in a few areas. This occurs when a certain function takes many large inputs. Since the code present in the mfs.go is essential to work with docker node, the function cannot be changed.

Summary

Docker is an open source application which can be used on many platforms to deliver applications to clients. It uses a concept of virtualization for this purpose. Docker is widely used in the industry as well. Since Docker utilizes host processes and is used in many organizations, it has become a target for attackers. Developers of Docker have always considered security when developing, but attackers have found many attack vectors to attack Docker, as well as to use Docker as an attack vector to attack a host device. The large number of internal and external libraries used by Docker does not help the cause. In such a premise, Docker has done well to fend off attacks and patch vulnerabilities as soon as possible.

This report has investigated various aspects of the Docker application and code with the objective of understanding the security level of Docker. During the investigation, it was brought to light that Docker (since 2016) haven't had many vulnerabilities directly attached to the Docker code. Most vulnerabilities were due to internal or external libraries. It is also evident that good security practices were put into place when the code was written. The handling of issues pointed out that code was reviewed by a senior developer before merging the code.

Thus, the conclusion of this report is that Docker is secure application, but, as with every other application, vulnerabilities will continue to be found.

References

[1] [Online]. Available: https://www.docker.com/.

[2] Frichetten, "CVE-2019-5736-PoC," github, 19 February 2019. [Online]. Available: https://github.com/Frichetten/CVE-2019-5736-PoC. [Accessed 28 April 2020].

[3] [Online]. Available: https://github.com/opencontainers/runc/commit/0a8e4117e7f715d5fbeef398405813ce8e88558b.

[4] [Online]. Available: https://github.com/docker/docker-ce/releases/tag/v18.09.2.

[5] A. Iwanuik and B. Poplawski, "CVE-2019-5736: Escape from Docker and Kubernetes containers to root on host," Dragon Sector, 13 February 2019. [Online]. Available: https://blog.dragonsector.pl/2019/02/cve-2019-5736-escape-from-docker-and.html. [Accessed 28 April 2020].

[6] N. V. database, "CVE-2019-1020014," National Vulnerability database, 29 July 2019. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-1020014. [Accessed 28 April 2020].

[7] [Online]. Available: https://github.com/docker/docker-credential-helpers/commit/1c9f7ede70a5ab9851f4c9cb37d317fd89cd318a.

[8] [Online]. Available: https://github.com/docker/docker-credential-helpers/releases/tag/v0.6.3.

[9] N. V. Database, "CVE-2019-15752," National Vulnerability Database, 28 August 2019. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-15752. [Accessed 28 April 2020].

[10] M. Rmoan, "elevation of Privilege in Docker for windows," medium, 24 August 2019. [Online]. Available: https://medium.com/@morgan.henry.roman/elevation-of-privilege-in-docker-for-windows-2fd8450b478e. [Accessed 28 April 2020].

[11] [Online]. Available: https://github.com/moby/moby/blob/f2614f2107c838d014d31b806e3b8a9f1395cb2b/vendor/github.com/vbatts/tar-split/tar/asm/disassemble.go.

[12] [Online]. Available: https://github.com/moby/moby/issues/35075.

[13] docker, "Docker Engine 17.11 releease notes," Docker, 20 November 2017. [Online]. Available: https://docs.docker.com/engine/release-notes/17.11/. [Accessed 29 April 2020].

[14] Docker, "Docker Overview," Docker, [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed 29 April 2020].

[15] T. Meritt, "9 Software Security design Principles," DZone, 15 January 2013. [Online]. Available: https://dzone.com/articles/9-software-security-design. [Accessed 30 April 2020].

[16] D. J. Walsh, "Bringing new security features for Docker," Opensource, 03 September 2014. [Online]. Available: https://opensource.com/business/14/9/security-for-docker. [Accessed 30 April 2020].

[17] "Container Security: Examining Potential Threats to the Container Environment," Trned Micro, 14 May 2019. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/security-technology/container-security-examining-potential-threats-to-the-container-environment. [Accessed 2 May 2020].

[18] [Online]. Available: https://docs.python.org/2/library/tempfile.html.