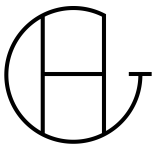

HoGent

Faculteit Bedrijf en Organisatie

Security en Managebility van Docker containers

Tomas Vercautter

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Bert Van Vreckem
Co-promotor:
Bert Van Vreckem

Instelling: —

Academiejaar: 2015-2016

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Security en Managebility van Docker containers

Tomas Vercautter

Scriptie voorgedragen tot het bekomen van de graad van
Bachelor in de toegepaste informatica

Promotor:
Bert Van Vreckem
Co-promotor:
Bert Van Vreckem

Instelling: —

Academiejaar: 2015-2016

Tweede examenperiode

Samenvatting

Voorwoord

In deze bachelorproef wordt er van uitgegaan dat de lezer een basis kennis heeft van docker en linux. De belangrijkste elementen van docker die men best vo

Inhoudsopgave

Hoofdstuk 1

Inleiding

Docker containers is een manier van virtualiseren van services. Hierbij worden services in een

In deze bachelorproef zal ik onderzoek doen naar de security en managebility van Docker conainers en deze vergelijken met de security en managebility van het draaien van de services rechtsreeks op de virtuele machine.

Deze bachelorprief is onderverdeeld in drie grote delen. In de eerste plaats bespreek ik de security zelf. In een tweede deel zal ik het hebben over de managebility van Docker containers. En als laatste onderdeel bespreek ik logging bij docker containers.

1.1 Probleemstelling en Onderzoeksvragen

Hoofdstuk 2

Methodologie

Deel I

Corpus

Hoofdstuk 3

Security

Op vlak van security ben ik gestart van CIS Docker 1.6 Benchmark geschreven door Pravin Goyal (2015). Hierin is in samenwerking met de "Center for Internet Security-onderzoek gedaan naar de beveiliging van docker containers. Dit onderzoek is onderverdeeld in zes grote onderdelen namelijk Host configuration, Docker daemon configuration, docker daemon configuration files, Container images and build file, Container runtime en docker security operations. Hierbij wordt er gekeken welke van deze elementen standaard in docker worden uitgevoerd. In dit hoofdstuk zal ik dezelfde zes onderdelen behandelen en vergelijen. Hierbij zal ik kijken hoe we deze beveiliging aanpakken wanneer we onze services in een Docker container draaien tegenover wanneer we deze native in onze virtuele machine draaien.

Hierbij heb ik ook telkens deze methodes voor Docker containers uitgetest om te kijken hoe makkelijk deze kunnen geïmplementeerd worden in een nieuwe omgevinge of bestaande omgeving. Een grote hulp hiervoor was de github repository "docker-bench-security"van docker een grote hulp. Dit is een script die alle elementen die worden aangehaald in CIS Docker 1.6 Benchmark automatisch getest en feedback over gegeven.

3.1 Host Configuration

Een van de belangrijkste onderdelen in het beveiligen van de Docker stack is het beveiligen van de host machine waarop de docker containers zich zullen bevinden. Dit heeft drie onderdelen namelijk het up to date houden van het systeem, het juist

configureren van het systeem, en het auditten van het systeem.

3.1.1 Up to date houden van het systeem

Om de veiligheid van een systeem te kunnen garanderen moet er ten allen tijde gezorgd worden dat het systeem up to date blijft. Dit zorgt ervoor dat security flaws die gekend en gepatcht zijn ook op het systeem beveiligd zijn. Doordat we Docker gebruiken komt hier nog een extra laag bij. Bij het Updaten van een machine waarop de services rechtstreeks draaien moet enkel het host systeem, de kernel van het host systeem en de services up to date blijven. Doordat Docker een kernel deelt met het hostsysteem wordt het up to date houden van de kernel belangrijker. Naast het onderhouden van de voorvernoemde elementen moet er nu, doordat we Docker gebruiken, nog met twee extra dingen rekening gehouden worden. Enerzijds moet Docker zelf nu ook upgedate worden, maar een onderdeel die soms vergeten wordt is dat elke container nu ook zijn eigen besturingssysteem gebruikt. Hierover wordt er verder bij het beveiligen van Container images meer uitleg over gegeven.

3.1.2 Configureren van het systeem

Bij het configureren van het hostysteem wordt net zoals bij het up to date houden van het systeem dezelfde stappen van een native applicatie op een hostysteem configureren gevolgd. Maar door de extra laag komen daar nog extra stappen bij. Bij een systeem waar Docker op draait wordt er aangeraden om een aparte partitie te creëren voor de containers. Dit is een simpele operatie waardoor alle docker gerelateerde files zich niet meer tussen de files van het hostsysteem bevinden.

Daarnaast moet ook onder controle gehouden worden wie toegang krijgt tot de docker daemon. De Docker daemon heeft 'root' access, gebruikers die worden toegevoegd aan de 'docker' usergroup en geen 'root' privileges hebben kunnen hierdoor 'root' access verkrijgen tot het hostsysteem. Men directories sharen tussen het host systeem en een guest container. Doordat containers standaard altijd runnen als 'root' heeft de container als de '/' directory gemount is op deze container ongelimiteerde toegang tot het volledige host systeem. Hierdoor kunnen zonder ristricties aanpassingen aan het hostysteem gedaan worden. Dit betekend dat een gebruiker die toegang heeft tot de docker daemon verhoogde rechten kan verkrijgen door gewoon een container op te starten.

Zoals bij een virtuele machine waarop de services rechtstreeks wordt ook bij een

systeem waar docker containers op draaien aangeraden overbodige services uit te schakelen of te verwijderen. Moest een gebruiker door een van deze services toegang krijgen tot het hostsysteem. Zou hij makkelijker 'root' rechten kunnen vergrijgen volgens de manier beschreven in de vorige paragraaf.

3.1.3 Auditen van het systeem

3.2 Docker daemon configuratie

Bij het beveiligen van de docker stack is ook de configuratie van de docker daemon een belangrijk element. Dit heeft twee grote onderdelen. Enerzijds de configuratie van de docker daemon zelf, dit houd in dat we de daemon zelf zo gaan configureren dat deze zo veilig en robuust mogelijk wordt. Aan de andere kant moeten we ook zorgen dat de configuratie files zelf de juiste file permissies en owners hebben. Hierbij is de vergelijking met een virtuele machine, waar de services rechtstreeks op draaien, moeilijk aangezien er hier geen docker daemon op te vinden is. Omdat het configureren van de Docker daemon dingen verandert doe voor elke container toegepast worden kunnen we het wel vergelijken met hoe de inviduele services globaal geconfigureerd worden.

3.2.1 Globale configuratie

3.2.2 netwerk configuratie

3.2.3 registry configuratie

3.2.4 Permissies voor Docker config files

Het configureren van de Docker daemon met de juiste instellingen voor zowel networking, registry gebruik en algemene configuratie heeft enkel nut als we ook de configuratie files waarin deze instellingen worden bewaard correct beveiligd zijn.

3.3 Security of Docker images

Als we het hebben over de beveiliging van Docker images zijn er twee grote categoriën die we moeten onderscheiden. Een eerste categorie is wanneer we werken met Docker images die afkomstig zijn van de officiele Docker Hub Registry, een repository waar iedereen zijn images op kan posten en kan distrubueren. Daarnaast hebben we ook de mogelijkheid om onze eigen images te maken.

3.3.1 Images van de Docker hub

Wanneer we Docker images gebruiken van de officele Docker Hub Registry is een handig hulpmiddel het commando ´docker search zoekterm´. Dit geeft ons een lijst van alle docker images die voldoen aan deze zoekterm. Een voorbeeld hiervan vindt u hieronder.

```
$ docker search busybox
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
busybox	Busybox base image.	316	[OK]	
progrium/busybox		50		[OK]
radial/busyboxplus	Full-chain, Internet enabled, busybox made...	8		[OK]
odise/busybox-python		2		[OK]

Een van de dingen die hierbij opvallen is de kolom 'OFFICIAL'. Een image die als öfficialïs aangeduid, is door het approval proces van docker zelf gegaan. Dit houd onder andere in dat de images onderhouden worden, bij herhaaldelijk builden altijd hetzelfde resultaat bekomen wordt, consistent maar toch duidelijk zijn en goed bevijligd zijn. Meer hierover in het deel over eigen docker images beveiligen.

De officiële images zijn in het algemeen veiliger om te gebruiken dan niet officiële images. Dit wil niet zeggen dat niet officiële images niet veilig kunnen zijn, maar deze zijn niet gecontroleerd geweest door docker zelf. Doordat we bij het downloaden van een Docker image niet direct kunnen zien wat deze zal doen wordt het dus sterk afgeraden om Docker images van niet vertrouwlijke bronnen te gebruiken.

Van elke image die gevonden wordt met ´docker search´ en dus op de officiele repositories staat (ervan uitgaand dat er manueel geen andere repositories zijn toegevoegd) is online op de docker hub webpagina een Dockerfile te vinden. Met deze Dockerfile kunnen we ook zelf kijken hoe de image precies is opgebouwd. Dit kan wel snel een tijdrovend proces worden aangezien docker images kunnen gebaseerd zijn op

andere images. Hierdoor kan voorkomen dat er meer dan een paar Dockerfiles moeten bekeken worden voordat men weet wat er precies allemaal inbegrepen zit in de onderzochte image.

Een ander belangrijk element dat met de officiële docker images probeert opgelost te worden is dat iedereen zijn eigen manier heeft om de dockerfiles op te bouwen en manieren van software te installeren. Door een methodiek aan te bieden die gevolgd moet worden voordat de image kan opgenomen worden in de officiële docker hub zorgen ze voor een gelijkaardige opbouw van Dockerfiles bij officiële docker images.

De vergelijking met een virtuele machine zonder docker kan gemaakt worden door de officiële docker hub te vergelijken met een package archive. In beide gevallen kan zowel van de officiële repositories afgehaald worden als zelf niet officiële repositories toegevoegd. Het verschil ligt dan uiteraard in de aangeboden software, Docker images t.o.v. packages.

3.3.2 zelf Docker Images maken

Als we zelf Docker images willen maken is goede referentie om aan te voldoen de eisen die gesteld worden voor officiële docker images. Uiteraard zijn er hier elementen die voor eigen gebruik niet uiterst noodzakelijk zijn maar deze vormen een goede basis om van te vertrekken.

Als eerste belangrijk punt moeten we rekening houden met welke packages we willen gebruiken in onze images. Hierbij zijn dezelfde risico's aan verbonden als moesten we een gewone virtuele machine gebruiken. Enkel packages installeren die gebruikt worden en het goed configureren (indien nodig) van packages is hier even belangrijk als bij een gewone virtuele machine. Dus bij het installeren van packages is het aangewezen om de fingerprint the controleren.

Bij het maken van onze docker images kunnen we ook files copiëren in onze container. Dit wordt best zo veel mogelijk vermeden maar indien toch vereist is het best om de file specefiek te copiëren en niet een hele map. Bij het copieren van een volledige map kunnen bij herbuilden ongewenste files in de map bijgekomen zijn waardoor er een onverwacht of ongewenst restultaat kan bekomen worden.

Een belangrijk element om bij stil te staan als de gemaakte Docker image robuust wenst gemaakt te worden is het definiëren van vaste package versies. Hierdoor blijven onze images bij het herbuilden van dezelfde versie altijd hetzelfde. Extra beveiliging kan hierdoor verkregen worden, door het gebruiken van specifieke versies van packages

weten we ook altijd exact welke versies van deze packages er in onze image gebruikt worden. Een nadeel hieraan is dat bij het uitkomen van nieuwe updates en eventuele security updates en bugfixes we onze docker images moeten aanpassen. In het algemeen wordt aangeraden om voorgedefineerde versies te gebruiken, nieuwe versies van software worden best eerst uitvoerig getest vooraleer in gebruik genomen te worden. Door het definiëren van de versie dat we willen gebruiken hebben we hierover volledige controle.

Het maken van onze eigen Docker images kunnen we dus volledig vergelijken aan het configureren van een virtuele machine. Hierbij moet er ook gekeken worden naar welke files we willen gebruiken op deze machine alsook welke packages er op geinstalleerd moeten staan. Dezelfde veiligheids voorschriften die gebruikt worden bij het configureren van een virtuele machine zijn dus ook van toepassing op het maken van een eigen Docker image.

3.4 Container configuration

Als we Docker containers gebruiken moeten de containers zelf ook goed geconfigureerd zijn. Een groot deel van de veiligheid van de containers zelf hangt af van hoe we ons ´docker run´ commando gebruiken. Daarnaast zijn er ook nog een aantal punten waarvoor we moeten oppassen of kunnen optimaliseren op de container zelf.

3.4.1 docker run commando configureren

Elke container word opgestart met het commando ´docker run [OPTIONS] IMAGE [COMMAND] [ARG...]´. Met dit commando kunnen aan de hand van onze zelfgemaakt Docker image of een gedownloade Docker image een container aanmaken en direct opstarten. Doordat dit een van de belangrijkste commando's is die bij Dokcer hoord is het uiteraard ook een van de krachtigste, dus moeten we wel beter enkele dingen in gedachten houden wanneer we het uitvoeren.

Bij het uitvoeren van het standaard ´docker run´ commando namelijk ´docker run IMAGE´ zijn er veel dingen die standaard in orde zijn. Uiteraard zijn er dingen waar we normaal geen problemen mee mogen hebben maar soms toch problemen kunnen veroorzaken.

Een van deze dingen zijn de standaard Linux Kernel Capabilities die de container heeft. Over Linux Kernel Capabilities zou kunnen een Bachelorproef op zich geschreven

worden en gaat valt hierdoor niet binnen de scope van deze Bachelor proef. Maar sterk vereenvoudigd is dit de basis instructieset die een linux besturingssysteem bezit, deze instructies zijn als gewone gebruiker niet allemaal toegankelijk maar de root user kan elke instructie uitvoeren. Docker heeft uit voorzorg hiervoor deze lijst met Capabilities al sterk verminderd in vergelijking met de gewone Linux Kernel Capabilities, dit zorgt ervoor dat de root user in een container veel minder kan dan een root user op een gewoone machine. De meeste use cases hebben containers ook geen echte root rechten nodig. Hiervoor kunnen we dus de Linux Kernel Capabilities per container gaan tweaken en enkel privileges toewijzen aan de containers die ze echt nodig hebben. Om deze beperking te omzeilen kunnen Docker containers ook met de optie ´–privileged´ opstarten. Dit geeft de container alle Kernel Capabilities die het hostsysteem heeft en wordt dus best buiten in enkele zeer specifieke use cases vermeden.

Naast de mogelijkheid om de standaard Kernel Cabapilities aan te passen van de containers hebben we ook de mogelijkheid om het processor en geheugen gebruik te limiteren. Docker containers kunnen zonder verdere configuratie het complete geheugen gebruiken van de host, hierdoor kunnen andere containers zonder geheugen geraken en dit kan voor problemen zorgen. De mogelijkheid een container het complete geheugen van het hostsysteem in beslag neemt is zeker iets waar rekening mee gehouden moet worden. Hiervoor gebruiken we de optie ´-m´. Hiermee kunnen de maximale hoeveelheid geheugen die een container mag gebruiken bepalen. Daarnaast hebben we ook CPU gebruik. Elke container gebruikt standaard een gelijke hoeveelheid van de cpu. De optie die gebruikt wordt om het cpu gebruik te bepalen is de ´-c´ of de cpu-shares optie. Standaard heeft deze een waarde van 1024, als alle containers deze waarde hebben, krijgen ze allemaal gelijke prioriteit op de cpu. Door deze waarde te verhogen of verlagen kunnen we desgewenst een hogere of lagere prioriteit verkrijgen voor onze container.

Het is maar op het moment dat we dit standaard run commando beginnen aanpassen met opties dat er enkele dingen zijn waarvoor we best opletten. Wanneer een docker run commando wordt uitgevoerd zijn er twee dingen die we in opties kunnen toevoegen waar we toch iets voorzichtiger mee moeten zijn. Namelijk netwerk configuratie en mounts van host directories in de container.

Netwerk configuratie van docker is als er niets aan is aangepast tameliljk robuust. Maar als we hierin dingen beginnen aanpassen is het mogelijk om deze rubuustheid wat te ondermijnen. Een van de elementen hierbij is poorten. Bij het connecteren naar buiten de machine waarop een container opereert moet een zowel een poort op de host als op de container worden opengesteld, waarna deze poorten "verbonden worden"waardoor het netwerkverkeer die op de gedefinieërde poort van de host toekomt wordt doorgezonden naar de gekozen poort op de container. Een best practice hierbij

is om altijd de volledige connectie te definiëren, dit wil zeggen dat we zowel de de enxterne interface, de externe host poort als de interne container poort meegeven met ons run commando. Hierbij worden op de container best enkel poorten geopend die effectief gebruikt worden door deze container. Dit verkleind het aanvalsvlak via netwerk op deze container. TCP/IP poorten onder 1024 op de host machine worden best ook vermeden om op te binden. Uiteraard zijn er services die hier op moeten binden zoals een http proxy moet luisteren op poort 80 om te kunnen functioneren en zijn ook veilier als ze hierop luisteren. Een goede maatstaf om te weten als een container op een privileged port gebonden moet worden is kijken indien men de service die in de container draait ook wanneer die op een virtuele machine draait op een poort onder 1024 zou binden. Daarnaast hebben we net zoals bij Kernel Capabilities ook bij netwerking ook een manier om het intern Docker netwerk te omzeilen. Een container gebruikt standaard een bridged netwerk, hierdoor wordt deze container in een aparte netwerk stack gestoken. Als we de optie ´–net=host´ meegeven overriden we dit en geven we de container toegang tot alle netwerk interfaces van de host machine. Dit moet tenzij in zeer specifieke toepassingen niet gebrijkt worden.

3.4.2 container configuratie

Om extra beveiliging te voorzien voor de docker containers is het mogelijk (indien ondersteund door de host OS) om AppArmor en SeLinux te gebruiken. Met AppArmor is het mogelijk om voor elke container appart een AppArmor profiel te maken. SeLinux voor docker moet aangezet worden bij het opstarten van de Docker Daemon, dit kan simpel gedaan worden door de Daemon bij het starten de optie ´–selinux-enabled´ mee te geven. Hierna kunnen we SeLinux security opties meegeven aan containers. Zowel AppArmor en Selinux zorgen ervoor dat we elke container individueel kunnen configureren op vlak van security. Het individueel configureren van SeLinux en AppArmor voor containers kan vergeleken worden met het configureren voor processen op een niet-Docker systeem. Dit is buiten de scope van deze Bachelorproef en hierom gaan we hier niet verder op in.

Daarnaast is een element die we moeten bekijken wat we nu precies in een container willen draaien qua processen. Het idee achter docker is om één enkele applicatie per container op te delen. Docker luistert standaard enkel op één hoofdprocess, dus indien we meerdere processen willen uitvoeren op 1 container die niet gemaakt zijn om vanuit 1 hoofdprocess op te starten en informatie terug door te geven via dit hoofd process moeten we een process manager gaan bijvoegen in de container. Dit zorgt er onder andere voor dat een container die algemeen gezien een simpel iets is weer zeer complex wordt. Anderzijds zorgt dit ervoor dat je de processen binnen de container niet meer op

een optimale monier kunt monitoren met docker. Een voorbeeld hiervan zou zijn indien we zowel onze applicatie als de database op eenzelfde container plaatsen. Inden we deze configuratie zouden gebruiken merken we al snel dat het heel wat makkelijker zou zijn op elke vlak om gewoon de database uit de applicatie container te halen en in zijn eigen container op te starten, dit heeft ons naast een robuuster systeem (als de applicatie crasht zou het kunnen de database mee doen crashen indien het op dezelfde container draait) ook een meer schaalbaar systeem door gewoon meer applicatie containers te linken met dezelfde database container. Een ander voorbeeld is het gebruik van SSH, container hebben in de meeste gevallen geen ssh nodig. Alle containers zijn toegankelijk via het hostsysteem met het commando ´docker exec CONTAINER´ dus is daar de meest aangewezen plaats om SSH te installeren.

Hoofdstuk 4

Managebility

Hoofdstuk 5

Logging

Logging is een zeer belangrijk onderdeel van docker containers. Omdat we onze docker containers veel meer gaan isoleren en onze containers veel vluchtiger zijn dan gewone services op een virtuele host wordt loggen van

Hoofdstuk 6

Conclusie

Bibliografie

Anderson, C. (2015). Docker [Software engineering]. *IEEE Software*, 32(3):102–c3.

Brockmeier, J. (2014). Is It Safe? A Look at Docker and Security from LinuxCon — Project Atomic.

Docker (2015). Introduction to Container Security. Technical report.

Docker (2016). Docker security.

Ewindisch (2014). On the Security of Containers.

Jessie Frazelle (2016). Docker Engine 1.10 Security Improvements.

Lenny Zeltser (2015). Security Risks and Benefits of Docker Application Containers.

Mónica, D. (2015). Understanding Docker Security and Best Practices.

Mouat, A. (2015). *Docker Security*.

Petazzoni, J. (2013). Containers & Docker: How Secure Are They?

Pravin Goyal (2015). CIS Docker 1.6 Benchmark v1.0.0. Technical report.

Toni de la Fuente (2015). Docker Security Tools: Audit and Vulnerability Assessment.

Voelker, A. (2015). Why I don't use Docker much anymore - Abe Voelker.

Lijst van figuren

Lijst van tabellen