Πανεπιστήμιο Πειραιώς
Σχολή Τεχνολογιών Πληροφορικής και Τηλεπικοινωνίων
Τμήμα Ψηφιακών Συστημάτων
Π.Μ.Σ Ασφάλεια Ψηφιακών Συστημάτων

Διπλωματική Εργασία Μεταπτυχιακού Επιπέδου

# Δοκιμές Παρείσδυσης σε περιβάλλοντα Docker Container (Docker Container Penetration Testing)

Επιβλέπον: Γκρίτζαλης Στέφανος, Καθηγητής
Τχης Βάσιος Γεώργιος, ΚΕΠΥΕΣ

Δημήτριος Κούδας dkoudas@ssl-unipi.gr MTE2013

Πειραιάς, 2023

## Περίληψη

Στις μέρες μας τα περιβάλλοντα Docker έχουν μεγάλη ανάπτυξη με πάνω από 7 εκατομμύρια εφαρμογές να τρέχουν σε Docker και πάνω από 18 εκατομμύρια προγραμματιστές να επιλέγουν το Docker για την ανάπτυξη των εφαρμογών τους. Είναι πολύ σημαντικό όμως εκτός από όλες τις ευκολίες που μας παρέχουν τα περιβάλλοντα Docker να λαμβάνουμε υπόψη τα ρίσκα και τις προκλήσεις ασφαλείας που μπορεί να χρειαστεί να αντιμετωπίσουμε σε τέτοια περιβάλλοντα. Σε αυτή την εργασία θα δούμε πως μπορούμε να επιτύχουμε μεγαλύτερα επίπεδα ασφαλείας σε περιβάλλοντα Docker container μέσα από τη σκοπιά της επιθετικής ασφάλειας και ειδικότερα των δοκιμών παρείσδυσης. Θα δούμε τα δομικά στοιχεία των Docker container προκειμένου να γνωρίσουμε την τεχνολογία του Docker ώστε να αποκτήσουμε εφόδια για τα επόμενα κεφάλαια. Επιπλέον, θα δούμε και θα αναλύσουμε την ασφάλεια του Docker από δύο σκοπιές, μέσα από τα container και από τον host προκειμένου να δούμε όλο το φάσμα την ασφάλειας του Docker. Στη συνέχεια, θα δούμε του ελέγχους που πρέπει να πραγματοποιήσουμε κατά τη δοκιμή παρείσδυσης και πάλι από δύο σκοπιές μέσα και έξω από τα Docker containers. Επιπλέον, θα αναλύσουμε χρήσιμα εργαλεία που βοηθούν και αυτοματοποιούν τους ελέγχους που θα χρειαστεί να πραγματοποιήσουμε. Τέλος, θα συγκεντρώσουμε όλους τους ελέγχους και τις ενέργειες που χρειάζεται να γίνουν κατά τη διάρκεια ενός ελέγχου παρείσδυσης σε μια λίστα ελέγχου την οποία θα μπορεί να χρησιμοποιήσει κάποιος κατά τη διάρκεια του ελέγχου.

**Abstract**

Nowadays Docker environments are booming with over 7 million applications running on Docker and over 18 million developers choosing Docker to develop their applications. However, it is very important that in addition to all the conveniences that Docker environments provide, we should consider the risks and security challenges that we may have to face in such environments. In this paper we will look at how we can achieve higher levels of security in Docker container environments from the perspective of aggressive security and in particular penetration testing. We will look at the building blocks of Docker containers in order to learn about Docker technology to equip ourselves for the following chapters. In addition, we will look at and analyze Docker security from two perspectives, from within the containers and from the host in order to see the full spectrum of Docker security. Then, we will look at the checks we need to perform during penetration testing again from two viewpoints inside and outside of Docker containers. In addition, we will analyze useful tools that help and automate the checks we will need to perform. Finally, we will compile all the checks and actions that need to be done during a penetration test into a checklist that can be used during the process.

# Πίνακας Εικόνων

# Πίνακας Πινάκων

# Περιεχόμενα

# 1. Εισαγωγή

Με την συνεχή αύξηση των ψηφιακών υπηρεσιών είναι πολύ σημαντικό οι εταιρείες και οι οργανισμοί να μπορούν να εξασφαλίσουν την ασφάλεια των συστημάτων τους. Ένας τρόπος για την αξιολόγηση της ασφάλειας των συστημάτων και των εφαρμογών μια εταιρείας ή ενός οργανισμούς είναι η δοκιμές παρείσδυσης (penetration tests). Πρόκειται για μια πρακτική η οποία έχει τη φιλοσοφία της ενίσχυσης της άμυνας μέσω της επίθεσης. Στην εργασία αυτή θα επικεντρωθούμε σε δοκιμές παρείσδυσης σε περιβάλλοντα Docker container. Στόχος της εργασίας είναι να αναλύσουμε τα συστήματα Docker τις ευπάθειες που μπορεί να έχουν τέτοια συστήματα καθώς και να παρέχουμε μια μεθοδολογία για την πραγματοποίηση δοκιμών παρείσδυσης σε τέτοιου είδους περιβάλλοντα.

Αρχικά, στο Κεφάλαιο 2 θα δούμε βασικές έννοιες οι οποίες θα μας βοηθήσουν στη συνέχεια. Πιο συγκεκριμένα, θα δούμε τι είναι τα containers, τις διαφορές τους με τα εικονικά μηχανήματα και τι είναι οι δοκιμές παρείσδυσης και πως αυτές πραγματοποιούνται. Στη συνέχεια στο Κεφάλαιο 3 θα δούμε τα βασικά χαρακτηριστικά του Docker και πως αυτό μας βοηθάει στην ανάπτυξη εφαρμογών. Θα δούμε τα δομικά του στοιχεία και ένα πρακτικό παράδειγμα για το πως μπορούμε να αναπτύξουμε και να τρέξουμε μια εφαρμογή σε Docker container. Στο κεφάλαιο 4 θα δούμε την ασφάλεια του Docker τόσο μέσα από τα containers όσο και έξω από αυτά. Πιο συγκεκριμένα, θα δούμε τις πιο γνωστές και συνήθης ευπάθειες που μπορεί να υπάρχουν σε ένα περιβάλλον Docker container, γνωστούς τύπους επιθέσεων καθώς και τα πιο γνωστά CVEs. Έπειτα στο κεφάλαιο 5 θα δούμε πως πραγματοποιούνται οι δοκιμές παρείσδυσης σε περιβάλλοντα Docker container. Πιο συγκεκριμένα, θα δούμε και θα αναλύσουμε ποιους ελέγχους πρέπει να πραγματοποιήσουμε καθώς και πως θα κάνουμε αυτούς τους ελέγχους. Στο κεφάλαιο 6 θα παρουσιάσουμε και θα αναλύσουμε εργαλεία που μπορούν να μας βοηθήσουν κατά τη διάρκεια των ελέγχων. Τέλος, στο κεφάλαιο 7 θα συγκεντρώσουμε όλους τους ελέγχους της διαδικασίας δοκιμών παρείσδυσης και μια λίστα ελέγχου την οποία μπορεί να χρησιμοποιήσει κάποιος κατά τη διάρκεια πραγματοποίησης των δοκιμών παρείσδυσης.

# 2. Βασικές Έννοιες

## 2.1 Λειτουργικό Σύστημα Linux

Το 1969, μια ομάδα προγραμματιστών της Bell Labs άρχισε να εργάζεται σε ένα έργο για τη δημιουργία ενός καθολικού λειτουργικού συστήματος για όλους τους υπολογιστές, το οποίο ονόμασαν Unix. Ήταν απλό και ο κώδικας του ήταν επαναχρησιμοποιήσιμος. Ήταν γραμμένο σε C και όχι σε γλώσσα assembly. Επειδή ήταν επαναχρησιμοποιήσιμο, ένα μέρος του κώδικα, που τώρα είναι γνωστό ως πυρήνας(kernel), χρησιμοποιήθηκε για την κατασκευή του λειτουργικού συστήματος και άλλων λειτουργιών και μπορούσε να χρησιμοποιηθεί σε διάφορους υπολογιστές. Ο πηγαίος κώδικας του έγινε επίσης διαθέσιμος στο κοινό(open source). Πολλές εταιρείες, συμπεριλαμβανομένων των IBM, HP και πολλών άλλων, άρχισαν να αναπτύσσουν τα δικά τους συστήματα Unix τη δεκαετία του 1980. Ως αποτέλεσμα, υπάρχουν ως τώρα μια σωρεία διανομών Unix. Στη συνέχεια, το 1983, ο Richard Stallman δημιούργησε το έργο GNU με σκοπό να δημιουργήσει ένα δωρεάν λειτουργικό σύστημα που μοιάζει με Unix που θα μπορούσε να χρησιμοποιήσει ο καθένας. Η πρότασή του, ωστόσο, είχε απήχηση. Εμφανίστηκαν πολλά περισσότερα λειτουργικά συστήματα παρόμοια με το Unix, αλλά κανένα από αυτά δεν κατάφερε να κερδίσει δημοτικότητα [1].

Ο Linus Torvalds, φοιτητής στο Πανεπιστήμιο του Ελσίνκι στη Φινλανδία, άρχισε να δημιουργεί τον δικό του κώδικα το 1991, ισχυριζόμενος ότι έχει μια δημόσια διαθέσιμη ακαδημαϊκή έκδοση του Unix. Αυτό το έργο τελικά εξελίχθηκε στον πυρήνα του Linux (Linux kernel. Κατασκεύασε αυτήν την εφαρμογή μόνο για τον προσωπικό του υπολογιστή επειδή δεν μπορούσε να αντέξει οικονομικά έναν υπολογιστή Unix 386 Intel. Χρησιμοποίησε τον μεταγλωττιστή GNU C στο MINIX. Ο μεταγλωττιστής GNU C εξακολουθεί να είναι η πιο δημοφιλής επιλογή για τη μεταγλώττιση κώδικα Linux, αλλά χρησιμοποιούνται και άλλοι μεταγλωττιστές, όπως ο μεταγλωττιστής Intel C [1].

Τα Linux είναι ένα σύνολο από λειτουργικά συστήματα τα οποία βασίζονται στον πυρήνα Linux (Linux kernel). Είναι βασισμένα στην οικογένεια λειτουργικών συστημάτων Unix και είναι ανοιχτού κώδικα (open source). Τα Linux αναπτύχθηκαν αρχικά για προσωπικούς υπολογιστές με βάση την αρχιτεκτονική Intel x86, αλλά έκτοτε έχουν μεταφερθεί σε περισσότερες πλατφόρμες από οποιοδήποτε άλλο λειτουργικό σύστημα [2].

Σήμερα τα Linux βρίσκονται παντού καθώς αποτελούν βάση για το λειτουργικό σύστημα Android που χρησιμοποιούν τα περισσότερα smartphones σήμερα. Επιπλέον, τα Linux χρησιμοποιούνται και από υπερυπολογιστές, server, προσωπικούς υπολογιστές, IoT συσκευές κ.α.

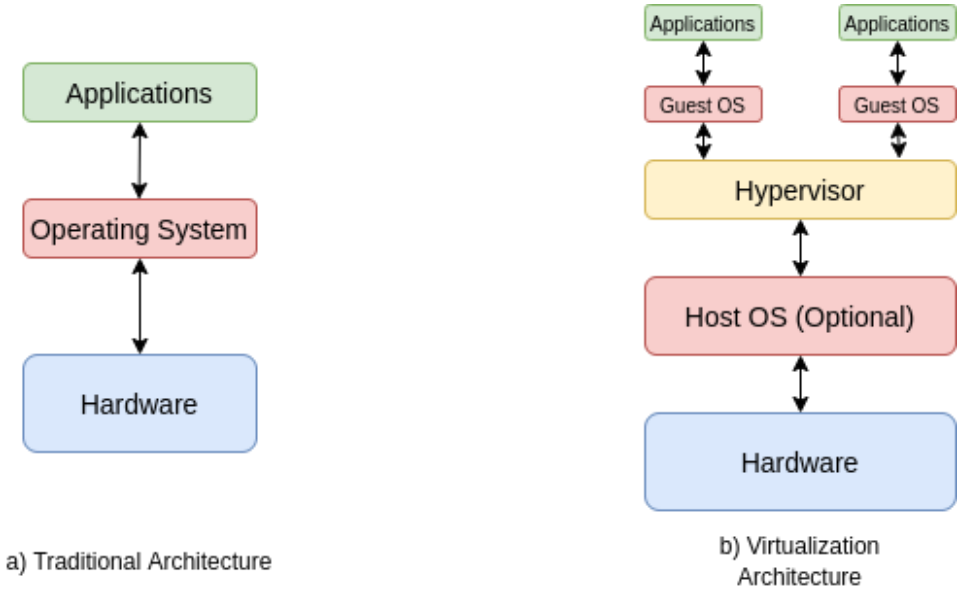
## 2.2 Virtualization

### 2.2.1 Εισαγωγή

Virtualization ονομάζεται η τεχνική με την οποία μπορούμε να εκτελούμε εικονικά instances ενός υπολογιστικού συστήματος σε ένα επίπεδο πάνω από το φυσικό hardware. Συνήθως, το virtualization χρησιμοποιείται για να τρέξουμε πολλαπλά λειτουργικά συστήματα σε ένα φυσικό μηχάνημα, δηλαδή στο ίδιο hardware. Η τεχνολογία του virtualization χρονολογείται πίσω στη δεκαετία του 1960, όμως από τις αρχές του 2000 ξεκίνησε να χρησιμοποιείται ευρέως. Η χρήση της τεχνολογίας αυτής εκτοξεύτηκε όταν οι περισσότερες εταιρείες που διέθεταν servers ξεκίνησαν να χρησιμοποιούν το virtualization προκειμένου να μπορούν να τρέξουν διάφορες εφαρμογές στο ίδιο φυσικό μηχάνημα. Σήμερα η τεχνολογία αυτή χρησιμοποιείται ευρέως σε παρόχους υπηρεσιών φιλοξενίας καθώς φυσικά και στις υπηρεσίες cloud.

Η τεχνολογία Virtualization χρησιμοποιεί ένα λογισμικό που ονομάζεται hypervisor το οποία διαχωρίζει τους φυσικούς πόρους του μηχανήματος από το εικονικό περιβάλλον. Οι hypervisors μπορούν να τοποθετηθούν απευθείας στο hardware ή πάνω από ένα λειτουργικό σύστημα. Οι hypervisors διαχωρίζουν τους φυσικούς πόρους σε εικονικά περιβάλλοντα ώστε να μπορούν να χρησιμοποιηθούν. Οι πόροι

κατανέμονται ανάλογα με τις ανάγκες από το φυσικό περιβάλλον στα εικονικά περιβάλλοντα. Οι χρήστες αλληλεπιδρούν και εκτελούν εργασίες μέσα στο εικονικό περιβάλλον που συνήθως ονομάζεται επισκέπτης (guest) ή εικονική μηχανή (virtual machine). Ένα Virtual Machine (VM) είναι στη ουσία ένα αρχείο με δεδομένα, έτσι είναι εύκολη η μεταφορά του οπουδήποτε αλλού με ευκολία. Όταν εκτελείται ένα εικονικό περιβάλλον και ένας χρήστης ή διεργασία απαιτεί πρόσθετους πόρους από το φυσικό περιβάλλον τότε ο hypervisor δημιουργεί ένα αίτημα στο φυσικό σύστημα και αποθηκεύει τις αλλαγές στην κρυφή μνήμη (cache) [3].




*Εικόνα 1 Διαφορές μεταξύ αρχιτεκτονικών virtualization και μη*

### 2.2.2 Τύποι Virtualization

Η τεχνολογία Virtualization χρησιμοποιείται κυρίως στους παρακάτω έξι τύπους:

- **Network Virtualization** είναι μια μέθοδος συνδυασμού των διαθέσιμων πόρων σε ένα δίκτυο με διαχωρισμό του διαθέσιμου εύρους ζώνης σε κανάλια, καθένα από τα οποία είναι ανεξάρτητο από τα άλλα και μπορεί να εκχωρηθεί. Ο στόχος είναι η απόκρυψη της πολυπλοκότητας του δικτύου διαχωρίζοντας το σε διαχειρίσιμα κομμάτια.

- **Storage Virtualization** είναι μια μέθοδος η οποία συγκεντρώνει τον αποθηκευτικό χώρο από πολλές συσκευές αποθήκευσης δικτύου σε κάτι που φαίνεται να είναι μια ενιαία συσκευή αποθήκευσης που διαχειρίζεται από μια κεντρική κονσόλα.
- **Server Virtualization** είναι μια μέθοδος η οποία κρύβει τους πόρους του server από τους χρήστες του. Η πρόθεση της μεθόδου αυτής είναι να απαλλαγεί ο χρήστης από το να χρειάζεται να κατανοήσει και να διαχειριστεί περίπλοκες λεπτομέρειες των πόρων του server.
- **Data Virtualization** είναι μια μέθοδος η οποία αφαιρεί τις παραδοσιακές τεχνικές λεπτομέρειες των δεδομένων και της διαχείρισης τους, όπως η τοποθεσία, η απόδοση ή η μορφή, προς όφελος της ευρύτερης πρόσβασης και μεγαλύτερης ανθεκτικότητας που συνδέεται με τις επιχειρηματικές ανάγκες.
- **Desktop Virtualization** είναι μια μέθοδος η οποία επιτρέπει την δημιουργία εικονικών μηχανημάτων με γραφικό περιβάλλον. Η μέθοδος αυτή χρησιμοποιείται κυρίως για την φιλοξενία workstations σε server στα οποία οι χρήστες αποκτούν πρόσβαση απομακρυσμένα.
- **Application Virtualization** είναι μια μέθοδος η οποία αποκρύπτει το επίπεδο εφαρμογής από το λειτουργικό σύστημα. Έτσι η εφαρμογή μπορεί να τρέξει ανεξάρτητα από το λειτουργικό σύστημα.

### 2.2.3 Πλεονεκτήματα του Virtualization

Όπως είδαμε η τεχνολογία virtualization έχει πολλές εφαρμογές στα σύγχρονα υπολογιστικά συστήματα. Παρακάτω θα συγκεντρώσουμε τα πιο σημαντικά πλεονεκτήματα που προσφέρει η τεχνολογία αυτή:

- **Μειωμένο κόστος**: Με το virtualization μειώνεται το κόστος καθώς μειώνεται το φυσικό hardware που χρειάζεται ένα οργανισμός ή μια εταιρία. Επιπλέον, αφού μειώνεται η ανάγκη για περισσότερο hardware μειώνεται και το κόστος συντήρησης αυτού.
- **Ευκολία ανάκαμψης από καταστροφή**: Το virtualization βοηθάει στην εύκολη αποκατάσταση των υπηρεσιών καθώς μπορούν να χρησιμοποιηθούν πρόσφατα snapshots (backup των virtual machines).

- **Ευκολία στη απόκτηση backup**: Όπως είπαμε μπορούν να οριστούν διαδικασίες οι οποίες μπορούν να παίρνουν snapshots των virtual machines σε καθημερινή βάση.

## 2.3 Containers

### 2.3.1 Εισαγωγή

Τα containers είναι μια μορφή virtualization λειτουργικού συστήματος. Ένα container μπορεί να χρησιμοποιηθεί για την εκτέλεση οτιδήποτε, από μια μικρή διεργασία μικροϋπηρεσιών ή λογισμικού έως μια μεγαλύτερη εφαρμογή [4]. Πολλαπλά containers μπορούν να εκτελούνται στο ίδιο μηχάνημα και να μοιράζονται τον πυρήνα του λειτουργικού συστήματος με άλλα container, καθένα από τα οποία εκτελείται ως μεμονωμένες διεργασίες στο user space.

Η τεχνολογία των container δίνει λύση στο πως μπορεί το λογισμικό να λειτουργεί αξιόπιστα όταν μετακινείται από το ένα υπολογιστικό περιβάλλον στο άλλο. Αυτό είναι δυνατό καθώς, μέσα σε ένα container υπάρχουν όλα τα απαραίτητα εκτελέσιμα, κώδικας, βιβλιοθήκες και αρχεία διαμόρφωσης που χρειάζεται το λογισμικό για να τρέξει.

### 2.3.2 Πλεονεκτήματα των containers

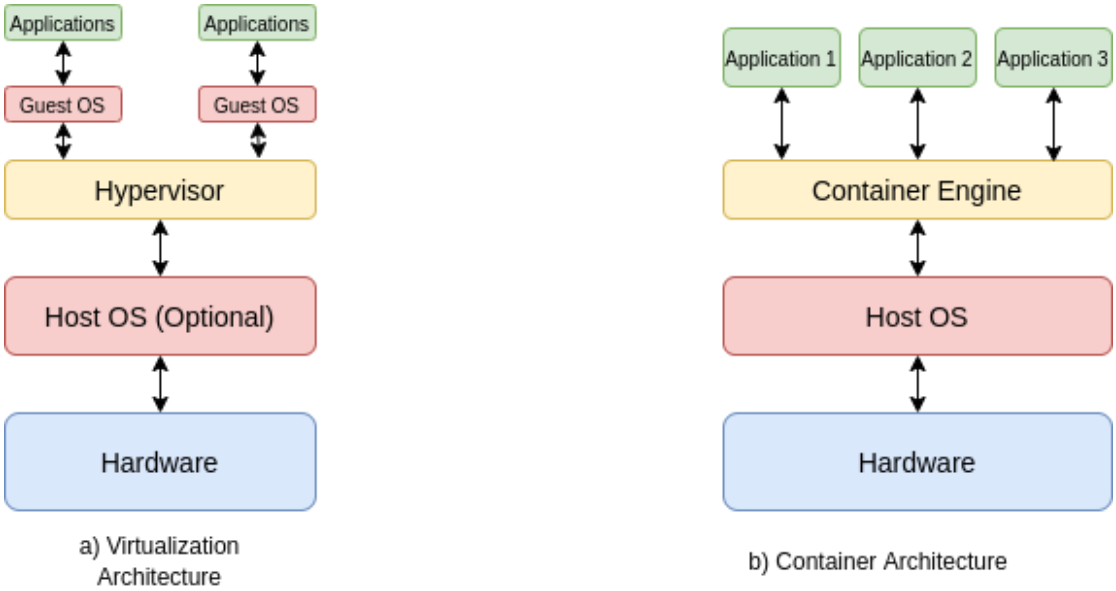
Παρακάτω θα δούμε τα πλεονεκτήματα που έχουν τα containers:

- Τα containers απαιτούν λιγότερους πόρους σε σχέση με τα virtual machines καθώς δεν εμπεριέχουν το image του λειτουργικού συστήματος.
- Μπορούμε εύκολα να αναπτύξουμε εφαρμογές σε μηχανήματα με διαφορετικά λειτουργικά συστήματα και hardware.
- Οι εφαρμογές μέσα σε ένα container τρέχουν πανομοιότητα οπουδήποτε και αν τρέχει το container με αποτέλεσμα την ομαλή λειτουργία των εφαρμογών.

- Με την χρήση containers επιτυγχάνουμε μεγαλύτερη απόδοση καθώς οι εφαρμογές που τρέχουν σε container μπορούν να αναπτυχθούν και να μεγαλώσουν γρηγορότερα.

### 2.3.3 Διαφορές μεταξύ Virtual Machine και Container

Ενώ τόσο τα container όσο και τα virtual machines επιτρέπουν την εκτέλεση εφαρμογών σε απομονωμένο περιβάλλον, τα container δεν είναι πλήρως ανεξάρτητα μηχανήματα. Ένα container είναι στην πραγματικότητα μια απομονωμένη διεργασία που μοιράζεται τον ίδιο πυρήνα Linux με το λειτουργικό σύστημα του host, καθώς και τις βιβλιοθήκες και άλλα αρχεία που απαιτούνται για την εκτέλεση του προγράμματος που εκτελείται μέσα στο container. Συνήθως, τα containers είναι σχεδιασμένα να εκτελούν ένα μόνο μια υπηρεσία, σε αντίθεση με την εξομοίωση ενός διακομιστή πολλαπλών χρήσεων. Συνοπτικά, τα containers είναι πολύ πιο “ελαφριά” από τα VMs, τα containers χρησιμοποιούν το virtualization σε επίπεδο λειτουργικού συστήματος ενώ τα virtual machines σε επίπεδο hardware, τα containers μοιράζονται τον πυρήνα του λειτουργικού συστήματος ενώ τα VMs έχουν το καθένα τον δικό του πυρήνα και τέλος τα containers χρησιμοποιούν πολύ λιγότερη μνήμη από τα VMs.




*Εικόνα 2 Διαφορές μεταξύ VM και Container αρχιτεκτονικών*

## 2.4 Δοκιμές Παρείσδυσης (Penetration Testing)

### 2.4.1 Εισαγωγή

Μια δοκιμή παρείσδυσης ,ή αλλιώς penetration test, είναι μια προσπάθεια αξιολόγησης της ασφάλειας μιας υποδομής πληροφορικής, προσπαθώντας με ασφάλεια να εκμεταλλευτούμε πιθανές ευπάθειας. Ευπάθειες μπορεί να υπάρχουν σε λειτουργικά συστήματα, σε υπηρεσίες και εφαρμογές, ακατάλληλες διαμορφώσεις ή επικίνδυνη συμπεριφορά τελικού χρήστη. Οι δοκιμές παρείσδυσης είναι επίσης χρήσιμες για την επικύρωση της αποτελεσματικότητας των αμυντικών μηχανισμών, καθώς και για τη συμμόρφωση του τελικού χρήστη στις πολιτικές ασφαλείας. Οι δοκιμές παρείσδυσης συνήθως εκτελούνται με τη χρήση μη αυτόματων ή αυτοματοποιημένων τεχνολογιών για συστηματική παραβίαση διακομιστών, τερματικών σημείων, εφαρμογών Ιστού, ασύρματων δικτύων, συσκευών δικτύου, φορητών συσκευών και άλλων πιθανών σημείων έκθεσης. Εάν εκμεταλλευτούμε με επιτυχία ευπάθειες σε ένα συγκεκριμένο σύστημα, τότε μπορούμε να επιχειρήσουμε να χρησιμοποιήσουμε το παραβιασμένο σύστημα για να ξεκινήσουμε επακόλουθες εκμεταλλεύσεις σε άλλους εσωτερικούς πόρους, ειδικά προσπαθώντας να επιτύχουμε σταδιακά υψηλότερα επίπεδα ασφαλείας και βαθύτερη πρόσβαση σε ηλεκτρονικά στοιχεία και πληροφορίες μέσω privilege escalation [5].

Οι πληροφορίες σχετικά με τυχόν ευπάθειες ασφαλείας που αξιοποιούνται επιτυχώς μέσω δοκιμών παρείσδυσης συνήθως συγκεντρώνονται και παρουσιάζονται στους διαχειριστές συστημάτων πληροφορικής και δικτύου για να βοηθήσουν αυτούς τους επαγγελματίες να βγάλουν στρατηγικά συμπεράσματα και να δώσουν προτεραιότητα στα σημεία που χρειάζονται βελτίωση. Ο θεμελιώδης σκοπός της δοκιμής παρείσδυσης είναι η μέτρηση του πόσο εύκολα μπορεί κάποιος να αποκτήσει πρόσβαση στο σύστημα ή πόσο εύκολα μπορεί να αποκτήσει κάνεις πρόσβαση μέσω ενός χρήστη του συστήματος και η αξιολόγηση τυχόν σχετικών συνεπειών που μπορεί να έχουν τέτοια περιστατικά στους εμπλεκόμενους πόρους ή λειτουργίες.

### 2.4.2 Στάδια της δοκιμής παρείσδυσης

Η διαδικασία εκτέλεσης μιας δοκιμής παρείσδυσης αποτελείται από έξι φάσεις:

1. Συλλογή πληροφοριών - Information Gathering
2. Αναγνώριση - Reconnaissance
3. Ανακάλυψη και σάρωση - Discovery and Scanning
4. Εκμετάλλευση ευπαθειών - Exploitation
5. Ανάλυση και Αναφορά - Analysis and Reporting
6. Καθαρισμός και αποκατάσταση - Clean Up and Remediation

**Συλλογή πληροφοριών - Information Gathering**

Το πρώτο στάδιο της δοκιμής παρείσδυσης είναι η συλλογή πληροφοριών. Ο οργανισμός που δοκιμάζεται θα παρέχει στον ελεγκτή παρείσδυσης γενικές πληροφορίες σχετικά με στόχους εντός πεδίου.

**Αναγνώριση - Reconnaissance**

Το στάδιο αναγνώρισης είναι κρίσιμο για ενδελεχείς δοκιμές παρείσδυσης, επειδή οι ελεγκτές παρείσδυσης μπορούν να εντοπίσουν πρόσθετες πληροφορίες που μπορεί να είναι σημαντικές για τη δοκιμή παρείσδυσης. Αυτή η φάση είναι ιδιαίτερα χρήσιμη σε δοκιμές παρείσδυσης εσωτερικού και/ή εξωτερικού δικτύου, ωστόσο, συνήθως δεν εκτελούμε αυτήν την αναγνώριση σε δοκιμές παρείσδυσης εφαρμογών ιστού, εφαρμογών για κινητά ή API.

**Ανακάλυψη και σάρωση - Discovery and Scanning**

Μόλις καθοριστεί το πεδίο εφαρμογής, οι ομάδες δοκιμών παρείσδυσης μπορούν να αρχίσουν να δουλεύουν. Στη φάση της ανακάλυψης, οι ομάδες εκτελούν διαφορετικούς τύπους αναγνώρισης στον στόχο τους. Από τεχνικής πλευράς, πληροφορίες όπως οι διευθύνσεις IP μπορούν να βοηθήσουν στον προσδιορισμό πληροφοριών σχετικά με τα τείχη προστασίας και άλλες συνδέσεις. Από την προσωπική πλευρά, δεδομένα τόσο απλά όπως ονόματα, τίτλοι εργασίας και

διευθύνσεις ηλεκτρονικού ταχυδρομείου μπορούν να έχουν μεγάλη αξία. Οι testers μπορούν να χρησιμοποιήσουν αυτά τα δεδομένα για να στείλουν μηνύματα ηλεκτρονικού ψαρέματος (phishing) ή να καταλάβουν ποιος μπορεί να έχει προνομιακά διαπιστευτήρια, με τα οποία μπορούν να έχουν πλήρη πρόσβαση στο περιβάλλον. Επιπλέον, πριν από την εκμετάλλευση ενός συστήματος, οι ομάδες δοκιμών παρείσδυσης πρέπει να αναζητήσουν αδυναμίες στο περιβάλλον. Συχνά αναφέρεται ως footprinting, αυτή η φάση της ανακάλυψης περιλαμβάνει τη συλλογή όσο το δυνατόν περισσότερων πληροφοριών σχετικά με τα συστήματα-στόχους, τα δίκτυα και τους ιδιοκτήτες τους χωρίς να επιχειρείται να διεισδύσει σε αυτά. Η αυτοματοποιημένη σάρωση είναι μια τεχνική που μπορεί να χρησιμοποιηθεί για την αναζήτηση ευπαθειών που μπορούν να χρησιμοποιηθούν ως πόρτα.

**Εκμετάλλευση ευπαθειών - Exploitation**

Σε αυτή τη φάση αυτοί που υλοποιούν τη δοκιμή μπορούν να αρχίσουν να χρησιμοποιούν αυτά τα πρόσφατα ανακαλυφθέντα σημεία εισόδου από την προηγούμενη φάση, δοκιμάζοντας να εκμεταλλευτούν όλες τις ευπάθειες που ανακάλυψαν προκειμένου να αποκτήσουν πρόσβαση στο σύστημα. Μόλις εισέλθουν σε ένα σύστημα, θα προσπαθήσουν να αυξήσουν τα προνόμια πρόσβασής τους (privilege escalation) στο περιβάλλον, επιτρέποντάς τους να προβούν σε οποιονδήποτε αριθμό πρόσθετων ενεργειών. Η απόκτηση προνομίων διαχείρισης (root privileges) επιτρέπει στους penetration testers να εντοπίζουν αδυναμίες ασφάλειας σε άλλους τομείς και πόρους, όπως κακή διαμόρφωση, αφύλακτη πρόσβαση σε ευαίσθητα δεδομένα ή αναποτελεσματική διαχείριση λογαριασμών και κωδικών πρόσβασης. Επιπλέον, μπορούν να δοκιμαστούν πολλαπλοί τύποι πόρων, εκτός από την εσωτερική υποδομή δικτύου και τους σταθμούς εργασίας που θα περιμέναμε να είναι ευάλωτοι σε επιθέσεις, μπορούν να δοκιμαστούν κινητές συσκευές, εφαρμογές ιστού, ακόμη και συσκευές IoT, όπως κάμερες ασφαλείας.

**Ανάλυση και Αναφορά - Analysis and Reporting**

Οι penetration testers θα πρέπει να παρακολουθούν προσεκτικά ό,τι κάνουν κατά τη διαδικασία εκμετάλλευσης ευπαθειών. Από εκεί, μπορούν να δημιουργήσουν μια αναφορά που περιλαμβάνει όλες αυτές τις λεπτομέρειες, επισημαίνοντας τι

χρησιμοποιήθηκε για την επιτυχή παρείσδυση στο σύστημα, ποιες αδυναμίες ασφαλείας εντοπίστηκαν και οποιαδήποτε άλλη σχετική πληροφορία ανακαλύφθηκε. Αυτές οι αναφορές θα πρέπει επίσης να περιλαμβάνουν ανάλυση για να βοηθήσουν στη χαρτογράφηση των επόμενων βημάτων μόλις ολοκληρωθεί η δοκιμή. Οι ομάδες δοκιμών παρείσδυσης μπορούν να βοηθήσουν στον προσδιορισμό των στοιχείων υψηλότερης προτεραιότητας που πρέπει να φροντίσει ένας οργανισμός το συντομότερο δυνατό, καθώς και προτάσεις για μεθόδους αποκατάστασης.

**Καθαρισμός και αποκατάσταση - Clean Up and Remediation**

Ακριβώς όπως με μια πραγματική επίθεση, οι penetration testers μπορούν να αφήσουν «ίχνη». Είναι σημαντικό οι ομάδες δοκιμών παρείσδυσης να επιστρέψουν στα συστήματα και να αφαιρέσουν τυχόν τεχνουργήματα που χρησιμοποιήθηκαν κατά τη διάρκεια της δοκιμής, καθώς θα μπορούσαν να αξιοποιηθούν στο μέλλον από κάποιον με κακόβουλες προθέσεις. Μόλις ολοκληρωθεί αυτό, ένας οργανισμός μπορεί να ασχοληθεί με τη διόρθωση των αδυναμιών ασφαλείας που ανακαλύφθηκαν και τέθηκαν σε προτεραιότητα κατά τη φάση της δοκιμής. Αυτό μπορεί να περιλαμβάνει τη θέσπιση αντισταθμιστικών ελέγχων για την προστασία αδυναμιών που δεν μπορούν εύκολα να αποκατασταθούν ή ακόμη και την επένδυση σε νέες λύσεις που μπορούν να βελτιώσουν την ασφάλεια και την αποτελεσματικότητα.

### 2.4.3 Τύποι δοκιμών παρείσδυσης

Υπάρχουν πολλά είδη δοκιμών παρείσδυσης. Οι περισσότερες δοκιμές διαφέρουν ως προς το ποιες πληροφορίες σχετικά με το σύστημα λαμβάνει ο αξιολογητής από τον διαχειριστή ή τον ιδιοκτήτη του συστήματος πριν από την έναρξη της δοκιμής ή τι είδους συστήματα ή εφαρμογές δοκιμάζονται. Παρακάτω θα δούμε μερικούς από τους γνωστότερους τύπους:

- **Black Box Penetration Test**: Η ομάδα που υλοποιεί τη δοκιμή παρείσδυσης δεν λαμβάνει πληροφορίες σχετικά με το σύστημα που βρίσκεται υπό αξιολόγησης.

- **Grey Box Penetration Test**: Η ομάδα που υλοποιεί τη δοκιμή παρείσδυσης λαμβάνει κάποιες συγκεκριμένες πληροφορίες σχετικά με το σύστημα όπως για παράδειγμα διαπιστευτήρια ή χαρτογράφηση δικτύου.
- **White Box Penetration Test**: Η ομάδα που υλοποιεί τη δοκιμή παρείσδυσης λαμβάνει όλες τις πληροφορίες για το σύστημα υπό αξιολόγηση. Αυτές οι πληροφορίες έχουν να κάνουν με το ίδιο το σύστημα αλλά και με πληροφορίες για κάθε κομμάτι αυτού.
- **Red Teaming**: Σε αυτόν τον τύπο η Η ομάδα που υλοποιεί τη δοκιμή παρείσδυσης προσπαθεί να προσομοιώσει μια πραγματική επίθεση. Άρα, οι υπάλληλοι δεν είναι ενήμερη και η ομάδα από πλευράς της προσπαθεί να είναι όσο πιο διακριτική γίνεται με τις επιθέσεις. Η ομάδα που εκτελεί το red teaming μπορεί να χρησιμοποιήσει οποιοδήποτε μέσω προκειμένου να εισβάλει στο σύστημα.
- **Κοινωνική Μηχανική**: Σε αυτόν τον τύπο η ομάδα που πραγματοποιεί την δοκιμή παρείσδυσης ασχολείται κυρίως με τους ανθρώπους που αλληλεπιδρούν με το σύστημα. Συνήθως, χρησιμοποιούνται τεχνικές ψαρέματος (phishing) καθώς και εισβολές σε φυσικούς χώρους.
- **Αξιολόγηση κώδικα**: Σε αυτόν τον τύπο η ομάδα δοκιμών παρείσδυσης αναλύει πηγαίο κώδικα εφαρμογών προκειμένου να ανακαλύψει ευπάθειες.

# 3. Docker

## 3.1 Εισαγωγή

Το Docker αποτελεί ένα εργαλείο που χρησιμοποιείται για τη δημιουργία και την ανάπτυξη εφαρμογών σε container. Η Docker Inc. ιδρύθηκε από τους Kamel Founadi, Solomon Hykes και Sebastien Pahl κατά τη διάρκεια της ομάδας εκκολαπτηρίων startup Y Combinator Summer 2010 και ξεκίνησε το 2011. Το Docker κυκλοφόρησε στο κοινό επίσημα το 2013 στο συνέδριο PyCon. Επιπλέον, διατέθηκε σαν open-source λογισμικό το Μάρτιο του 2013. Σήμερα, το Docker χρησιμοποιείται ευρέως για την ανάπτυξη εφαρμογών σε containers από μεγάλες εταιρείες ανάπτυξης καθώς και από παρόχους cloud [6].

## 3.2 Βασικές έννοιες Docker

### 3.2.1 Docker Image

Τα Docker Images είναι αρχεία τα οποία χρησιμοποιούνται για την εκτέλεση κώδικα μέσα σε container. Αποτελούνται από διάφορα επίπεδα και οδηγίες για τη δημιουργία των Docker containers. Πιο συγκεκριμένα, ένα Docker Image περιέχει τον κώδικα, τις βιβλιοθήκες, τα εργαλεία και άλλα αρχεία που χρειάζεται μια εφαρμογή για να τρέξει. Ένα Docker Image μοιάζει με την έννοια του snapshot στα VMs. Μπορούμε να δούμε μια λίστα με τα images που βρίσκονται στο σύστημα μας με την εντολή:

```
$ docker images
```

Επιπλέον μπορούμε να τρέξουμε ένα image με την εντολή:

```
$ docker run <image_name>
```

Επίσης μπορούμε να διαγράψουμε ένα docker image από το σύστημα μας με την εντολή:

```
$ docker rmi <image_id>
```

Τέλος, με την εντολή:

```
$ docker inspect <image_name>
```

μπορούμε να δούμε πληροφορίες για κάποιο docker image.

### 3.2.2 Docker Hub

Το Docker Hub είναι μια υπηρεσία registry στο cloud που επιτρέπει στους χρήστες Docker να κάνουν λήψη Docker Images που έχουν δημιουργηθεί από άλλους. Μπορεί επίσης ο καθένας να ανεβάσεις τα δικά του Docker Images στο Docker Hub. Μπορούμε να κατεβάσουμε ένα έτοιμο Docker Image από το Docker Hub με την εντολή:

```
$ docker pull <image_name>
```

### 3.2.3 Docker Containers

Ένα Docker Container αποτελεί ένα instance ενός Docker Image που εκτελείται. Μπορούμε να τρέξουμε και να αποκτήσουμε πρόσβαση μέσω τερματικού σε ένα Docker Container με την εντολή:

```
$ docker run -it <container_name> /bin/bash
```

Επιπλέον, μπορούμε να τρέξουμε οποιαδήποτε εντολή σε ένα container που ήδη εκτελείται με την εντολή:

```
$ docker exec -it <container_name> <command>
```

Μπορούμε να δούμε όλα τα Docker Container στο σύστημα μας χρησιμοποιώντας την εντολή:

```
$ docker ps -a
```

Για να δούμε μόνο τα Docker Container που εκτελούνται στο σύστημα μπορούμε να χρησιμοποιήσουμε την εντολή:

```
$ docker ps
```

### 3.2.4 Docker Daemon

Docker Daemon είναι η διεργασία που εκτελείται στον host και είναι υπεύθυνη για τη διαχείριση του Docker. Οποιαδήποτε ενέργεια που θέλει να εκτελέσει ο χρήστης στο Docker (π.χ τερματισμός ενός container) το Docker Daemon είναι η διεργασία που θα το κάνει αυτό. Είναι σημαντικό να σημειωθεί ότι η διεργασία αυτή εκτελείται με δικαιώματα root.

### 3.2.5 Dockerfile

Τα Dockerfiles αποτελούν οδηγίες τις οποίες χρησιμοποιεί το Docker για να κατασκευάσει (build) Docker Images. Πιο συγκεκριμένα, ένα Dockerfile είναι ένα αρχείο το οποίο περιέχει όλες τις εντολές που θα χρησιμοποιούσε ένας χρήστης στο τερματικό για να δημιουργήσει ένα Docker Image [7]. Μπορούμε να δημιουργήσουμε ένα Docker Image από το αντίστοιχο Dockerfile χρησιμοποιώντας την εντολή:

```
$ docker build -t  <Image_Name:Tag> <path_to_Dockerfile>
```

Η δομή ενός Dockerfile ακολουθεί τη μορφή:

**INSTRUCTION** arguments

δηλαδή με κεφαλαία γράμματα ορίζουμε την οδηγία Docker που θέλουμε να εκτελεστεί και δίπλα της ορίζουμε τα απαραίτητα arguments.

Παρακάτω ορίζουμε ένα πίνακα που περιέχει οδηγίες που μπορούμε να χρησιμοποιήσουμε σε ένα Dockerfile.

| Όνομα | Περιγραφή | Σύνταξη |
|---|---|---|
| **FROM** | Η οδηγία FROM χρησιμοποιείται στην αρχή κάθε Dockerfile και όριζει ένα BaseImage για της επερχόμενες εντολές Docker. | FROM <image_name:tag> |
| **RUN** | Η οδηγία RUN χρησιμοποιείται για την εκτέλεση εντολών σε ένα καινούριο επίπεδο πάνω από το BaseImage που έχει οριστεί. | RUN <command> ή<br>RUN ["executable", "param1", "param2"] |
| **CMD** | Η οδηγία CMD χρησιμοποιείται για να ξεκινήσει το λογισμικό που απαιτεί το container (π.χ εκτέλεση ενός executable). Μπορεί να υπάρχει μόνο μια οδηγία σε ένα Dockerfile και αν υπάρχουν περισσότερες λαμβάνετε υπόψη μόνο η τελευταία. | CMD ["executable","param1","param2"] |
| **LABEL** | Η οδηγία LABEL χρησιμοποιείται για να προσθέσουμε μεταδεδομένα (metadata) στο Docker Image που δημιουργούμε μέσω του Dockerfile. | LABEL <key>=<value> |
| **ENTRYPOINT** | Η οδηγία ENTRYPOINT | ENTRYPOINT ["executable", |

| | | |
|---|---|---|
| | χρησιμοποιείται για να ορίσουμε εκτελέσιμα αρχεία τα οποία θα τρέχουν όποτε ξεκινάει το container. Οι εντολές που ορίζουμε με την οδηγία ENTRYPOINT δεν μπορούν να αγνοηθούν ακόμα και αν τρέξουμε το container με arguments από το τερματικό. | "param1", "param2"] |
| **ENV** | Η οδηγία ENV χρησιμοποιείται για να ορίσουμε μια μεταβλητή περιβάλλοντος σε μια τιμή. Έτσι, μπορούμε να χρησιμοποιήσουμε τη μεταβλητή περιβάλλοντος στις επερχόμενες οδηγίες. | ENV <key>=<value> |
| **EXPOSE** | Η οδηγία EXPOSE χρησιμοποιείται για να ορίσουμε το Docker Container να ακούει σε συγκεκριμένες πόρτες δικτύου κατά την εκκίνηση του. | EXPOSE [<port>/<protocol>...] |
| **ADD** | Η οδηγία ADD χρησιμοποιείται για να προσθέσουμε αρχεία, φακέλους ή απομακρυσμένα αρχεία από την πηγή που ορίζουμε στο προορισμό. | ADD <src>... <dest> |
| **COPY** | Η οδηγία COPY χρησιμοποιείται για να αντιγράψουμε αρχεία ή φακέλους από την πηγή στον προορισμό. | COPY <src>... <dest> |
| **VOLUME** | Η οδηγία VOLUME χρησιμοποιείται για να ορίσουμε ένα σημείο αποθήκευσης για την διατήρηση των δεδομένων ακόμα και μετά των τερματισμό του container. | VOLUME ["/data"] |
| **USER** | Η οδηγία USER χρησιμοποιείται για να ορίσουμε το όνομα χρήστη (ή το | USER <user>[:<group>] |

| | | |
|---|---|---|
| | UID) και προαιρετικά την ομάδα χρηστών (ή το GID) που θα χρησιμοποιηθεί κατά την εκτέλεση του Image και για τυχόν οδηγίες RUN, CMD και ENTRYPOINT μέσα στο Dockerfile. | |
| **WORKDIR** | Η οδηγία WORKDIR χρησιμοποιείται για να ορίσουμε τον κατάλογο εργασίας για οποιεσδήποτε οδηγίες RUN, CMD, ENTRYPOINT, COPY και ADD μέσα στο Dockerfile | WORKDIR /path/to/workdir |
| **ARG** | Η οδηγία ARG χρησιμοποιείται για να ορίσουμε μια μεταβλητή κατά το χρόνο δημιουργίας του Image. | ARG <name>[=<default value>] |

*Πίνακας 1 Οδηγίες που μπορούμε να χρησιμοποιήσουμε σε Dockerfile*

### 3.2.5 Docker Networking

Το Docker διαχειρίζεται το κομμάτι της δικτύωσης για το κάθε container ώστε αυτό να μπορεί να επικοινωνήσει με άλλα container ή με τον host του Docker. Πιο συγκεκριμένα, η διεργασία Docker (Docker Daemon) δίνει τους απαραίτητους δικτυακούς πόρους στο Docker Container (όπως IP διεύθυνση, καταχωρήσεις DNS) οι οποίοι είναι διαχωρισμένοι από αυτούς του host. Το Docker χρησιμοποιεί κάποιους συγκεκριμένους drivers δικτύου (network drivers) τους οποίους θα δούμε στον παρακάτω πίνακα [8].

| **Όνομα** | **Περιγραφή** |
|---|---|
| **bridge** | Αποτελεί τον προκαθορισμένο driver δικτύου και χρησιμοποιείται συνήθως όταν οι εφαρμογές τρέχουν σε container που χρειάζεται να επικοινωνούν |
| **host** | Αφαιρεί την δικτυακή απομόνωση μεταξύ του host και του container και χρησιμοποιεί απευθείας το δίκτυο του host. |

| | |
|---|---|
| **overlay** | Συνδέει πολλούς Docker Daemons μεταξύ τους και επιτρέπει σε υπηρεσίες swarm να επικοινωνούν μεταξύ τους. |
| **ipvlan** | Δίνει τη δυνατότητα στους χρήστες να έχουν πλήρη έλεγχο τον διευθύνσεων IPv4 και IPv6. |
| **macvlan** | Επιτρέπει στους χρήστες να δώσουν MAC διεύθυνση στο container ώστε να φαίνεται σαν μια φυσική συσκευή στο δίκτυο |
| **none** | Αφαιρεί τελείως την δικτύωση από το Docker Container. |

*Πίνακας 2 Docker Network Drivers*

Μπορούμε να δούμε μια λίστα με τα δίκτυα Docker χρησιμοποιώντας την εντολή:

```
$ docker network ls
```

Επιπλέον μπορούμε να δούμε πιο αναλυτικές πληροφορίες για ένα συγκεκριμένο δίκτυο χρησιμοποιώντας την εντολή:

```
$ docker network inspect <network_name>
```

Ένα πολύ σημαντικό στοιχεία για την δικτύωση των Docker Containers που πρέπει πάντοτε να λαμβάνουμε υπόψη, είναι ότι από προεπιλογή όλα τα Docker Containers βρίσκονται στο ίδιο εσωτερικό δίκτυο με αποτέλεσμα να μπορούν να επικοινωνούν μεταξύ τους. Αυτό συμβαίνει διότι βρίσκονται στο ίδιο namespace. Σε διαφορετικά namespaces το Docker απομονώνει το container από τον host αλλά και από τα άλλα containers. Αυτό πρέπει να λαμβάνεται υπόψη κατά το σχεδιασμό του συστήματος με τα containers καθώς μπορεί να αποτελέσει απειλή για το σύστημα, νομίζοντας πως τα Docker Containers είναι πλήρως απομονωμένα ενώ στην πραγματικότητα δεν είναι.

### 3.2.6 Docker Socket

Για να αλληλοεπιδράσουν οι χρήστες του Docker με αυτό χρησιμοποιούν τις εντολές Docker. Οι εντολές αυτές ουσιαστικά λένε στη διεργασία Docker (Docker Daemon) τι εντολή να πραγματοποιήσει στο αντίστοιχο container. Η επικοινωνία αυτή, Docker Daemon και χρήστη, πραγματοποιείται κάνοντας requests στο Docker Engine API, δηλαδή μια εντολή αποτελεί και ένα HTTP request. Από προεπιλογή, για το API δημιουργείται ένα unix socket στη διαδρομή /var/run/docker.sock στο οποίο και "ακούει" για requests. Υπάρχει η δυνατότητα το Docker Engine API να λάβει requests και σε άλλου είδους sockets, tcp και fd. Τα δικαιώματα που έχει το Docker Socket (docker.sock) ορίζουν και τα δικαιώματα που χρειάζεται ο χρήστης για να αλληλεπιδράσει με το Docker Daemon. Ένας χρήστης για να μπορεί να πραγματοποιήσει requests στη διεργασία docker θα πρέπει να έχει δικαιώματα προσπέλασης και εγγραφής στο socket. Από προεπιλογή, το docker.sock χρειάζεται δικαιώματα root ή ο χρήστης να ανήκει στο docker group προκειμένου να μπορεί να το χρησιμοποιήσει.

### 3.2.7 Docker Registries

Τα Docker Registries αποτελούν μέσω αποθήκευσης και διάδοσης των Docker Images. Ένα Docker Registry αποτελείται από αποθετήρια Docker όπου το κάθε ένα από αυτά κρατάει όλες τι εκδόσεις ενός Docker Image. Όπως είδα και σε προηγούμενο κεφάλαιο το Docker Hub αποτελεί το μεγαλύτερο Docker Registry και είναι αυτό με το οποίο επικοινωνεί το Docker από προεπιλογή [9]. Για να κατεβάσουμε ένα Docker Image από ένα δικό μας registry ή κάποιο άλλο πέρα από το Docker Hub μπορούμε να χρησιμοποιήσουμε την εντολή:

```
$ docker pull registry_name:port image_name:tag
```

Πέρα από το Docker Hub υπάρχουν και άλλο γνωστά public registries όπως τα παρακάτω:

- Amazon Elastic Container Registry (ECR)

- Google Container Registry (GCR)
- Azure Container Registry (ACR)

Ένα registry μπορεί επίσης να είναι private και να χρησιμοποιείται εσωτερικά από τον οργανισμό μας για:

- Το διαμοιρασμός Docker Images μέσω του εσωτερικού δικτύου.
- Τη δημιουργία γρήγορων CI/CD pipelines.
- Την ανάπτυξη ενός Docker Image σε πολλά μηχανήματα μέσα στον οργανισμό.
- Τον καλύτερο έλεγχο του χώρου αποθήκευσης των Docker Images (βρίσκονται μέσα στον οργανισμό και όχι στο Internet).

### 3.2.8 Docker Compose

Το Docker Compose αποτελεί ένα εργαλείο το οποίο επιτρέπει τη χρήση αρχείων YAML[1] για την δημιουργία Docker Container. Χρησιμοποιώντας το Docker Compose μπορούμε να ορίσουμε τις απαραίτητες ρυθμίσεις για τα Docker Container μας χωρίς την χρήση εντολών Docker. Στο YAML αρχείο ορίζουμε τα services που θέλουμε να τρέξουμε καθώς και όλες τις απαραίτητες πληροφορίες για τα services αυτά. Στη συνέχεια θα δούμε ένα παράδειγμα υλοποίησης μια υπηρεσίας που χρησιμοποιεί Python Flask σε docker.

## 3.3 Παράδειγμα υλοποίησης εφαρμογής σε Docker

### 3.3.1 Εισαγωγή

Σε αυτή την ενότητα θα υλοποιήσουμε μια web εφαρμογή σε περιβάλλον Docker Container. Η εφαρμογή χρησιμοποιεί Python Flask. Το παράδειγμα χωρίζεται σε δύο μέρη. Στο πρώτο μέρος θα δούμε πως θα κάνουμε Dockerize την εφαρμογή μας και πως θα την τρέξουμε και στο δεύτερο μέρος θα δούμε πως μας βοηθάει το docker-compose στη διαδικασία αυτή.

### 3.3.2 Βήμα 1: Ανάπτυξη εφαρμογής

Για το παράδειγμα θα χρησιμοποιήσουμε μια πολύ απλή εφαρμογή σε Python Flask που απλά δημιουργεί ένα service που γυρνάει ένα μήνυμα. Ο κώδικας της εφαρμογής φαίνεται παρακάτω.

```
from flask import Flask
app = Flask(__name__)


@app.route('/')
def hello():
   return "This is a demo flask app for docker-lab"




if __name__ == "__main__":
   app.run(host ='0.0.0.0', port = 9001, debug = False)
```

Επιπλέον χρειαζόμαστε και το αρχείο requirements.txt το οποία περιέχει τις απαιτήσεις της εφαρμογής. Στη συγκεκριμένη περίπτωση, χρησιμοποιούμε μόνο το Flask modules.

```
Flask==2.0.2
```

### 3.3.3 Βήμα 2: Ανάπτυξη Dockerfile

Τώρα, αφού έχουμε όλα τα αρχεία της υπηρεσίας πρέπει να φτιάξουμε τα απαραίτητα αρχεία για την να μπορεί η εφαρμογή να τρέξει μέσα σε περιβάλλον Docker Container. Το αρχεία που πρέπει να φτιάξουμε είναι το Dockerfile. Παρακάτω θα δούμε βήμα προς βήμα την ανάπτυξη του Dockerfile.

Αρχικά θέλουμε τα ορίσουμε το BaseImage στο οποίο θα βασιστούμε. Στη συγκεκριμένη περίπτωση μπορούμε να χρησιμοποιήσουμε ένα Image για Python3.

```
FROM python:alpine3.9
```

Στη συνέχεια, πρέπει να ορίσουμε στο container ένα φάκελο στον οποίο θα έχουμε τα αρχεία της εφαρμογής μας.

```
RUN mkdir /python_web_app
```

Τώρα μπορούμε να αντιγράψουμε τα αρχεία της εφαρμογής που βρίσκονται στον host μέσα στο Container:

```
COPY . /python_web_app
```

Ορίζουμε τον κατάλογο εργασίας:

```
WORKDIR /python_web_app
```

Τώρα, αφού έχουμε τα αρχεία μέσα στο Container θέλουμε να εγκαταστήσουμε όλες τις απαιτήσεις που χρειάζεται η εφαρμογή μας. Για αυτό και δημιουργήσαμε το αρχείο requirements.txt:

```
RUN pip3 install -r requirements.txt
```

Στη συνέχεια, θα πρέπει να ορίσουμε το port στο οποίο θα ακούει το Docker Container που όπως ορίσαμε και στην εφαρμογή είναι το 9001.

```
EXPOSE 9001
```

Τέλος, θα πρέπει να πούμε στο Docker Container να εκκινεί την εφαρμογή μας μόλις τρέξει. Αυτό το κάνουμε με τις τελευταίες δύο οδηγίας:

```
ENTRYPOINT [ "python3" ]
CMD [ "app.py" ]
```

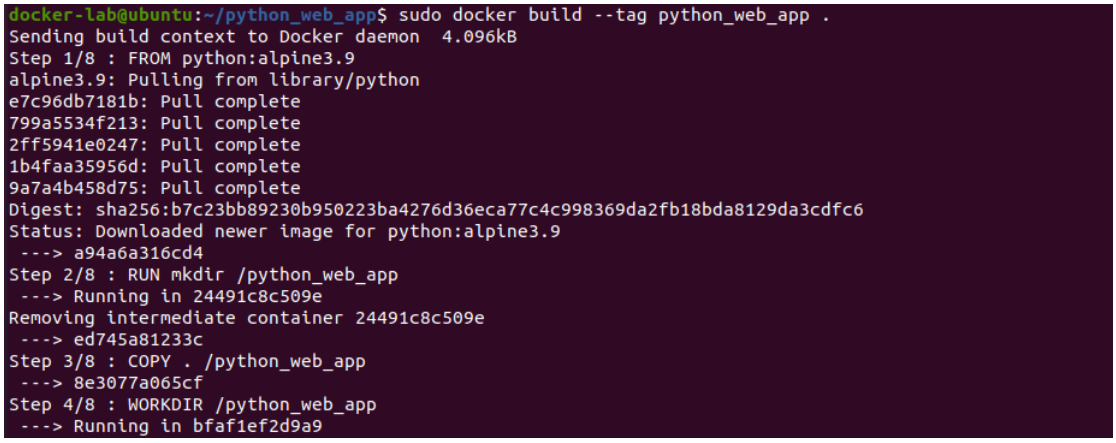
Συνολικά το Dockerfile φαίνεται παρακάτω:

```
FROM python:alpine3.9

RUN mkdir /python_web_app

COPY . /python_web_app

WORKDIR /python_web_app

RUN pip3 install -r requirements.txt

EXPOSE 9001

ENTRYPOINT [ "python3" ]

CMD [ "app.py" ]
```

### 3.3.4 Βήμα 3: Δημιουργία Docker Image από Dockerfile

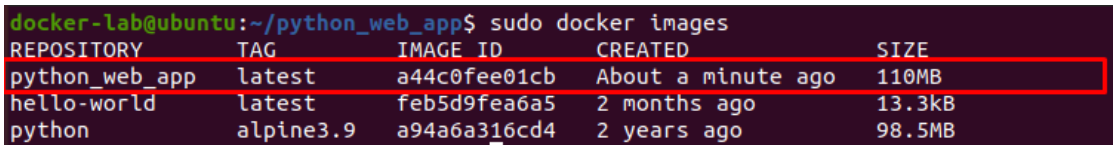
Τώρα αφού έχουμε όλα τα αρχεία που χρειαζόμαστε, πρέπει να δημιουργήσουμε το Docker Image για την εφαρμογή μας. Όπως έχουμε δει αυτό θα το κάνουμε χρησιμοποιώντας την εντολή “docker build”.

```
$ sudo docker build --tag python_web_app .
```




*Εικόνα 3 Δημιουργία Docker Image από Dockerfile*

Μπορούμε να δούμε παρακάτω ότι το Image δημιουργήθηκε με επιτυχία.

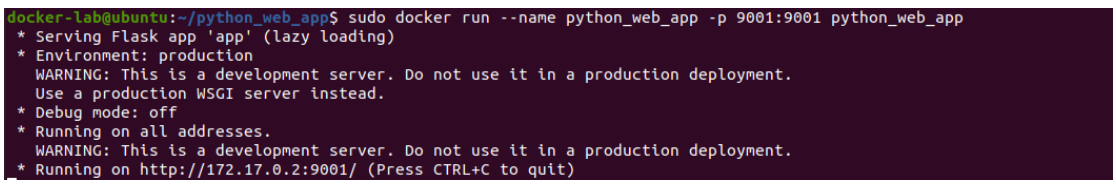



*Εικόνα 4 Λίστα με τα Docker Images στο σύστημα*

### 3.3.5 Βήμα 4: Εκτέλεση του Docker Container

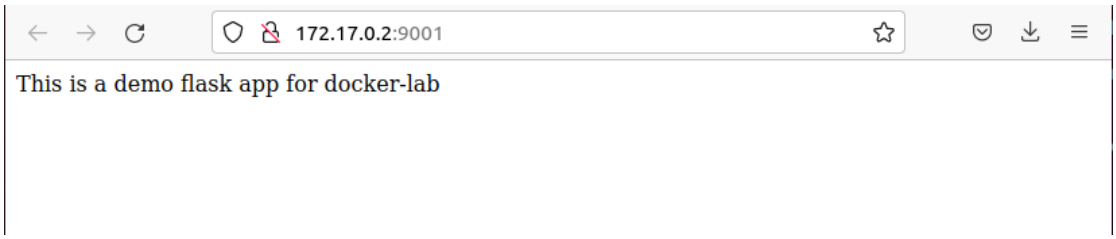
Τέλος, αφού έχουμε δημιουργήσει το Docker Image μπορούμε να εκτελέσουμε το Docker Container με την εντολή “docker run” που είδα και σε προηγούμενη ενότητα. Για να τρέξουμε το Image θα πρέπει να ορίσουμε και το port στο οποίο θα ακούει το Docker Container.

```
$ sudo docker run --name python_web_app -p 9001:9001 python_web_app
```




*Εικόνα 5 Εκκίνηση Docker Container*

Όπως βλέπουμε η εφαρμογή μας τρέχει και μπορούμε να αποκτήσουμε πρόσβαση στη διεύθυνση που αναγράφεται.




*Εικόνα 6 Πρόσβαση στην εφαρμογή*

### 3.3.6 Χρήση του Docker Compose

Σε αυτή την υποενότητα, θα δούμε πως μπορούμε να χρησιμοποιήσουμε το docker-compose στο παράδειγμα που δημιουργήσαμε. Όπως είδαμε και στην προηγούμενη υποενότητα όταν θέλουμε να εκκινήσουμε το container θα πρέπει να ορίζουμε τα ports και ίσως και άλλες παραμέτρους αν μιλάμε για πιο περίπλοκα συστήματα. Για να μπορέσουμε να διαχειριστούμε όλες αυτές τι παραμετροποιήσεις θα χρησιμοποιήσουμε το docker-compose. Με βάση το παράδειγμα που έχουμε αναπτύξει θα δημιουργήσουμε απλά ένα καινούριο αρχείο με όνομα “docker-compose.yml". Σε αυτό το αρχεία μέσα θα ορίσουμε όλα όσα χρειάζονται για την εφαρμογή μας. Στη συγκεκριμένη περίπτωση, θα ορίσουμε volumes δηλαδή ο φάκελος που έχουμε μέσα όλα τα αρχεία της εφαρμογής θα αντιστοιχεί σε ένα φάκελο που θα ορίσουμε μέσα στο container και το port στο οποίο θα ακούει το container. Το docker-compose.yml αρχείο φαίνεται παρακάτω:

```
version: '3'
services:
 app:
  build: .
  volumes:
   - .:/app
  ports:
   - "9001:9001"
```

Τώρα, μπορούμε με την παρακάτω εντολή να εκκινήσουμε το container:

```
$ sudo docker-compose up
```


*Εικόνα 7 Χρήση docker-compose για εκκίνηση container*

# 4. Ασφάλεια στο Docker

## 4.1 Εισαγωγή

Όπως είδαμε στο προηγούμενο κεφάλαιο το Docker είναι μια τεχνολογία που βοηθάει σε μεγάλο βαθμό τη διαδικασία ανάπτυξης και υποστήριξης διάφορων υπηρεσιών. Όμως παραμένει να είναι λογισμικό και έτσι μπορεί να αποτελέσει απειλή για το σύστημα σε περίπτωση ευπάθειας ή εσφαλμένης διαμόρφωσης (misconfiguration). Σε αυτό το κεφάλαιο θα δούμε του τύπους επιθέσεων σε περιβάλλοντα Docker, και τι βασικότερες επιθέσεις, καθώς και τεχνικές βελτίωσης της ασφάλειας στα περιβάλλοντα αυτά.

## 4.2 Τύποι επιθέσεων σε περιβάλλοντα Docker

### 4.2.1 Εισαγωγή

Οι τύποι των επιθέσεων σε περιβάλλοντα Docker χωρίζονται σε δύο μεγάλες κατηγορίες ανάλογα με το που “βρίσκεται ο επιτιθέμενος”, μέσα σε Docker container ή έξω από αυτό. Όπως είδαμε, όταν βρισκόμαστε μέσα σε Docker Container έχουμε πρόσβαση μόνο στους πόρους που επιτρέπεται να έχει πρόσβαση το container και πουθενά αλλού στο σύστημα. Από την άλλη όταν βρισκόμαστε έξω από το container στο host δηλαδή τότε μπορούμε να δούμε τις διεργασίες του συστήματος καθώς και άλλα containers που μπορεί να τρέχουν. Έτσι, έχουμε αυτές τις δύο μεγάλες κατηγορίες επιθέσεων οι οποίες έχουν να κάνουν με την εκάστοτε οπτική του επιτιθέμενου.

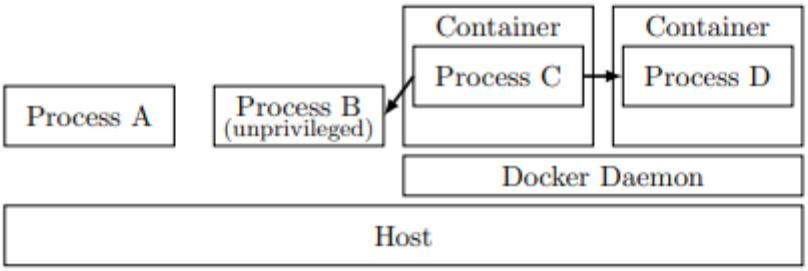
Με βάση αυτή την κατηγοριοποίηση έχουμε τις δύο σημαντικότερες κατηγορίες επιθέσεων σε περιβάλλοντα Docker container, επιθέσεις στο Docker Daemon και Container Escapes.

Ενώ με η παραπάνω κατηγοριοποίηση καλύπτει ένα ευρύ φάσμα επιθέσεων θα κάνουμε μια περαιτέρω ανάλυση το σεναρίων που μπορούν να προκύψουν. Καθώς

πρέπει να λάβουμε υπόψη τόσο τις ευπάθειες του ίδιου του docker αλλά και τις ευπάθειες που μπορούν να προκύψουν από τις εφαρμογές που τρέχουν μέσα στο container και το πως τα container συνδέονται μεταξύ τους.

### 4.2.1 Container Escapes

Οι επιθέσεις που σχετίζονται με το όρο Container Escapes έχουν ένα στόχο, να παρακάμψουν την απομόνωση των πόρων που προσφέρει το container. Αυτές οι επιθέσεις προϋποθέτουν ότι ο επιτιθέμενος έχει αποκτήσει με κάποιο τρόπο (συνήθως από κάποια ευπάθεια της εφαρμογής που τρέχει μέσα στο container) πρόσβαση στο container. Ουσιαστικά ο επιτιθέμενος που έχει πρόσβαση στο container προσπαθεί να ελέγξει διεργασίες τις οποίες δεν θα έπρεπε. Αυτές οι διεργασίες μπορεί να είναι είτε διεργασίες του host είτε διεργασίες κάποιου άλλου container. Με αυτή την παραδοχή μπορούμε να χωρίσουμε τα Docker Container Escapes σε δύο είδη ανάλογα με την πρόσβαση που αποκτά ο επιτιθέμενος στο τέλος [10].

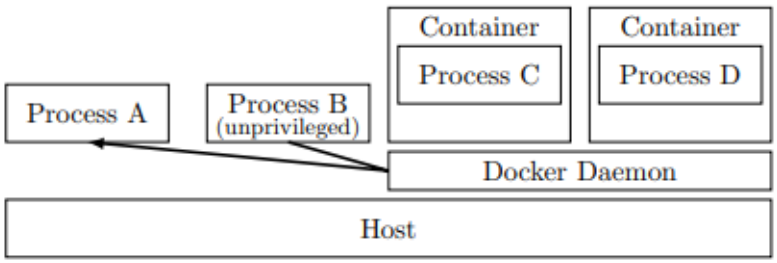



*Εικόνα 8 Επίθεση Container Escape*

### 4.2.2 Επιθέσεις Docker Daemon

Οι επιθέσεις που επικεντρώνονται στο Docker Daemon έχουν ως στόχος τον έλεγχο υψηλής αξίας πόρων μέσο του Docker Daemon. Πιο συγκεκριμένα, για αυτού του τύπου τις επιθέσεις θεωρούμε ότι ο επιτιθέμενος έχει πρόσβαση στον host που είναι εγκατεστημένο το Docker. Συνήθως, μιλάμε για πρόσβαση χαμηλών δικαιωμάτων. Ο

επιτιθέμενος αφού εντοπίσει την ύπαρξη του Docker τότε προσπαθεί να χρησιμοποιήσει το Docker Daemon για να προβεί σε κακόβουλες ενέργειες που θα απαιτούσαν δικαιώματα root. Αυτό είναι εφικτό καθώς το Docker για τις λειτουργίες του χρειάζεται πρόσβαση σε λειτουργίες του kernel, έτσι το Docker Daemon τρέχει με root δικαιώματα [10].




*Εικόνα 9 Επίθεση στο Docker Daemon*

## 4.3 Σενάρια επιθέσεων σε περιβάλλοντα Docker

Σε αυτή την ενότητα θα ορίσουμε κάποια σενάρια επιθέσεων με τη λογική πηγής και προορισμού. Πιο συγκεκριμένα, θα δούμε τα πιθανά attack surfaces ενός συστήματα που χρησιμοποιεί Docker Container [11].

### 4.3.1 Σενάριο 1: Επίθεση από την εφαρμογή προς το Container

Σε αυτό το σενάριο η επίθεση πραγματοποιείται από την εφαρμογή που τρέχει μέσα στο container προς το ίδιο το container. Μια ευπάθεια του ίδιου του λογισμικού που τρέχει μέσα στο container μπορεί να έχει τεράστιες επιπτώσεις στο σύστημα καθώς μπορεί επιτευχθεί ακόμα και host takeover [11]. Ένα παράδειγμα είναι η ευπάθεια **Log4Shell** η οποία επέτρεψε σε επιτιθέμενους να πραγματοποιήσουν container escapes και έτσι να αποκτήσουν πρόσβαση σε σημαντικούς πόρους του συστήματος [12].

### 4.3.2 Σενάριο 2: Επίθεση από ένα Container προς ένα άλλο Container

Μια τέτοια επίθεση είναι πιθανό να συμβεί όταν έχουν πολλαπλά containers που τρέχουν στον ίδιο host και επικοινωνούν μεταξύ τους [11]. Με άλλα λόγια τα container βρίσκονται στο ίδιο σύστημα και στο ίδιο δίκτυο. Αυτό το σενάριο επίθεσης είναι πολύ συχνό καθώς συνήθως χρησιμοποιούμε πολλαπλά containers για την κάθε υπηρεσία που θέλουμε να τρέξουμε. Ένα απλό παράδειγμα είναι όταν έχουμε μια εφαρμογή διαδικτύου η οποία χρησιμοποιεί μια βάση δεδομένων για αποθήκευση, τότε συνηθίζεται να χρησιμοποιούμε ένα container το οποίο τρέχει ο κώδικα της ίδιας της εφαρμογής και κάνει expose την πόρτα στην οποία τρέχει και ένα άλλο container για τη βάση δεδομένων το οποίο είναι προσβάσιμο μόνο από το container της εφαρμογής. Σε μια τέτοια περίπτωση λοιπόν αν ένας κακόβουλος αποκτήσει πρόσβαση στο container της εφαρμογής μέσω κάποιας ευπάθειας τότε θα μπορεί να "δει" και το container της βάσης αν αποκτήσει πρόσβαση και σε αυτό το container μέσω κάποιας ευπάθειας πάλι τότε ίσως μπορεί να δει και άλλα containers. Αυτός ο τύπος επιθέσεων ονομάζεται pivoting και μπορεί να αποφέρει μεγάλη ζημιά στο σύστημα.

### 4.3.3 Σενάριο 3: Επίθεση από ένα Container προς τον Host

Σε ένα τέτοιο σενάριο επίθεσης κάποιος κακόβουλος που έχει αποκτήσει πρόσβαση σε ένα container μπορεί να πραγματοποιήσει επιθέσεις ενάντια του host με σκοπό την "δραπέτευση" από το απομονωμένο περιβάλλον του container. Τέτοιου είδους επιθέσεις αποτελούν επιθέσεις υψηλού ρίσκου καθώς σε περίπτωση επιτυχίας τότε ο επιτιθέμενος αποκτά πρόσβαση σε πολύ σημαντικούς πόρους του συστήματος. Είδαμε και πιο πάνω αυτού του είδους τις επιθέσεις στο υποκεφάλαιο Container Escapes. Μια από τις πιο γνωστές τέτοιες επιθέσεις είναι η επίθεση **RunC container breakout** με CVE 2019-5736 την οποία και θα αναλύσουμε σε επόμενο κεφάλαιο [13].

### 4.3.4 Σενάριο 4: Επίθεση από τον Host προς ένα Container

Σε αυτό το σενάριο επίθεσης θεωρούμε ότι ο επιτιθέμενος έχει αποκτήσει πρόσβαση στον host που τρέχει το container. Ο επιτιθέμενος μπορείς να πραγματοποιήσει διαφόρων ειδών επιθέσεις προς το container με σκοπό την απόκτηση πληροφοριών, την αλλαγή της συμπεριφοράς της εφαρμογής ή ακόμα και την "καταστροφή" του container δημιουργώντας άρνηση υπηρεσίας.

### 4.3.5 Σενάριο 5: Επίθεση από ένα Container προς το Docker Engine

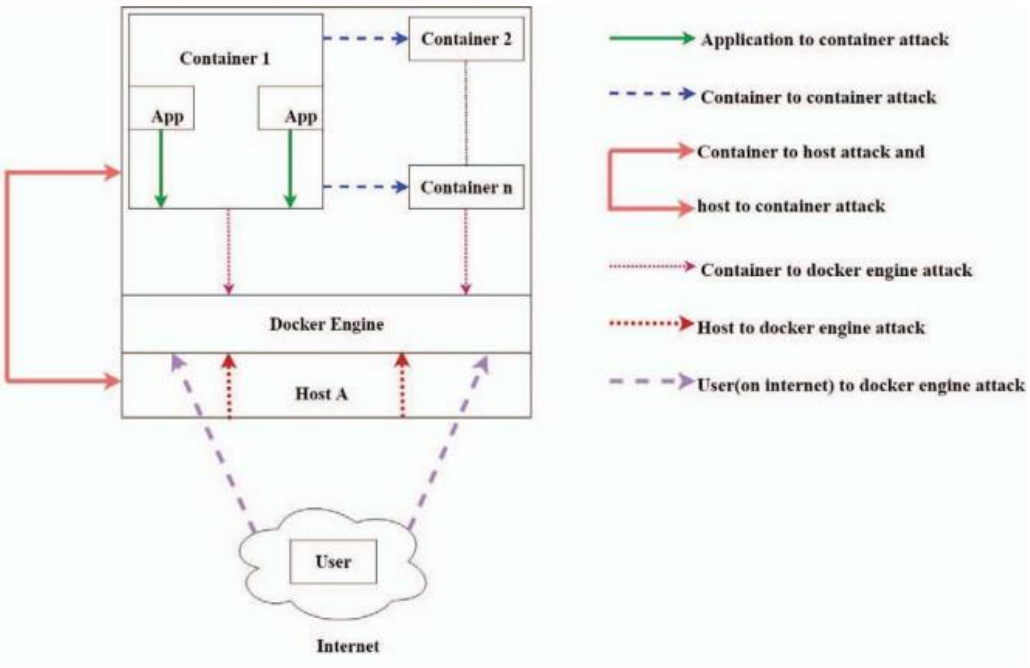
Σε αυτό το σενάριο επίθεσης ο επιτιθέμενος στοχεύει στο Docker Engine μέσω ενός Docker Container. Αυτό μπορεί να γίνει χρησιμοποιώντας ένα κακόβουλο Docker Image και κάνοντας κατάχρηση κάποιον misconfigurations. Ένα παράδειγμα είναι μια επίθεση DDoS που πραγματοποιήθηκε χρησιμοποιώντας κακόβουλα Docker Images από το Docker Hub και ένα εκτεθειμένο Docker Engine API [14].

### 4.3.6 Σενάριο 6: Επίθεση από τον Host προς το Docker Engine

Σε αυτό το σενάριο επίθεσης θεωρούμε ότι ο επιτιθέμενος έχει όλα τα δικαιώματα που έχει ο Host έτσι μπορεί εύκολα να επιτεθεί στο Docker Engine. Έτσι είναι πολύ σημαντική η μέριμνα για ασφάλεια του Host καθώς σε πιθανή κατάκτηση του από κάποιον επιτιθέμενο μπορεί να προκαλέσει σοβαρά ζητήματα ασφαλείας για το Docker.

### 4.3.7 Σενάριο 7: Επίθεση από κακόβουλο χρήστη προς το Docker Engine

Σε αυτό το σενάριο επίθεσης θεωρούμε ότι το Docker Engine μπορεί να είναι σε περιβάλλον cloud ή να έχει ενεργοποιηθεί πρόσβαση στο Docker Daemon μέσω HTTP/HTTPS. Σε μια τέτοια περίπτωση μπορεί να πραγματοποιηθεί επίθεση από κάποιον απομακρυσμένο κακόβουλο χρήστη χωρίς αυτός να έχει πρόσβαση σε κάποιο άλλο πόρο της υποδομής.




*Εικόνα 10 Docker attack layers*

Όπως είδαμε στα παραπάνω σενάριο υπάρχουν τέσσερα βασικά attack vectors **Container, Application, Docker Engine, Host** στα οποία μπορούν να πραγματοποιηθούν επιθέσεις και να θέσουν το περιβάλλον Docker σε κίνδυνο. Στη συνέχεια θα δούμε και θα αναλύσουμε βασικές επιθέσεις και σε ποιο από τα τέσσερα attack vectors μπορούν να χρησιμοποιηθούν, προκειμένου να αποκτήσουμε μια ολοκληρωμένη εικόνα για τις επιθέσεις που μπορούν να πραγματοποιηθούν σε ένα περιβάλλον Docker.

## 4.4 Γνωστοί τύποι επιθέσεων

### 4.4.1 Επίθεση κακόβολου λογισμικού (Malware attack)

Μια επίθεση κακόβουλου λογισμικού είναι μια κοινή κυβερνοεπίθεση όπου κακόβουλο λογισμικό εκτελεί μη εξουσιοδοτημένες ενέργειες στο σύστημα του θύματος. Το κακόβουλο λογισμικό περιλαμβάνει πολλούς συγκεκριμένους τύπους

επιθέσεων όπως ransomware, spyware, command and control και άλλα [15].Ένα παράδειγμα για την περίπτωση του Docker μπορεί να αποτελεί η δημιουργία ενός κακόβουλου container το οποίο να μπορεί να εκτελεί κακόβουλες ενέργειες στον Host και στη συνέχεια να μπορεί να μεταδίδεται κάνοντας inject τον εαυτό του σε Docker Images.

### 4.4.2 Επιθέσεις άρνησης υπηρεσίας ( Denial of Service Attacks)

Μια επίθεση Denial-of-Service (DoS) είναι μια επίθεση που έχει σκοπό να τερματίσει τη λειτουργία ενός μηχανήματος ή ενός δικτύου, καθιστώντας το απρόσιτο στους χρήστες για τους οποίους προορίζεται. Οι επιθέσεις DoS το επιτυγχάνουν αυτό πλημμυρίζοντας τον στόχο με κίνηση ή στέλνοντάς του πληροφορίες που προκαλούν συντριβή. Και στις δύο περιπτώσεις, η επίθεση DoS στερεί από τους νόμιμους χρήστες (δηλαδή υπαλλήλους, μέλη ή κατόχους λογαριασμού) την υπηρεσία ή τον πόρο που περίμεναν [16]. Τα Docker containers μπορούν να επικοινωνούν απευθείας με τον kernel του host, επιτρέποντας έτσι σε έναν επιτιθέμενο την ευκαιρία να αποκτήσει πρόσβαση στον host μέσω κάποιας επίθεσης docker escape. Αυτό εγείρει ουσιαστικά μια ανησυχία για την ασφάλεια των container. Όταν ένα ή περισσότερα container παραβιάζονται, τότε υπάρχουν αρκετοί πόροι για επακόλουθες επιθέσεις και κινδύνους. Ο εισβολέας μπορεί να εκτελέσει διάφορες επιθέσεις, όπως άρνηση υπηρεσίας (DoS) και κλιμάκωση προνομίων [17].

### 4.4.3 Κλιμάκωση Προνομίων (Privilege Escalation)

Η κλιμάκωση προνομίων είναι η πράξη εκμετάλλευσης ενός σφάλματος, ενός ελαττώματος σχεδιασμού ή μιας παράβλεψης ρύθμισης παραμέτρων σε ένα λειτουργικό σύστημα ή μια εφαρμογή λογισμικού για την απόκτηση αυξημένης πρόσβασης σε πόρους που κανονικά προστατεύονται από μια εφαρμογή ή χρήστη. Το αποτέλεσμα είναι ότι μια εφαρμογή με περισσότερα προνόμια από αυτά που είχε ο προγραμματιστής της εφαρμογής ή ο διαχειριστής του συστήματος μπορεί να εκτελέσει μη εξουσιοδοτημένες ενέργειες. Στην περίπτωση του Docker, ο επιτιθέμενος σε ύστερα από επιτυχή επίθεση αποκτά δικαιώματα διαχειριστή. Για την επίτευξη αυτής της επίθεσης χρησιμοποιούνται δύο μέθοδοι, τροποποίηση μνήμης και

τροποποίηση αρχείου. Στην τροποποίηση της μνήμης, ο εισβολέας αντικαθιστά ορισμένες δομές δεδομένων στη μνήμη, με αποτέλεσμα την αλλαγή της ροής ελέγχου. Κατά την τροποποίηση του αρχείου, τα προνομιακά αρχεία θα τροποποιηθούν από τον εισβολέα προκειμένου να αλλάξει τον κωδικό πρόσβασης του root ή να τροποποιήσει τα χαρακτηριστικά του αρχείου για να εκτελέσει κακόβουλα προγράμματα με δικαιώματα root.

### 4.4.4 Επιθέσεις Container Escape

Όπως έχουμε δει και παραπάνω στις επιθέσεις container escapes ο επιτιθέμενος καταφέρνει να "αποδράσει" από την απομόνωση που προσφέρεται μεταξύ container και host, και να αποκτήσει πρόσβαση διαχειριστή στον host.

### 4.4.5 Επιθέσεις ARP spoofing and MAC Flooding

Σε περιβάλλοντα Docker χρησιμοποιείται μια Virtual Ethernet bridge για την παροχή συνδεσιμότητας μεταξύ του container και του host Χρησιμοποιώντας αυτή την προσέγγιση, το Docker δημιουργεί μια Virtual Ethernet Bridge η οποία προωθεί αυτόματα τα πακέτα μεταξύ όλων των διεπαφών δικτύου του. Κάθε φορά που δημιουργείται ένα νέο container από το Docker, δημιουργείται ένα νέο Virtual Ethernet Interface (με νέο όνομα) και συνδέεται με τη γέφυρα. Το προεπιλεγμένο μοντέλο συνδεσιμότητας του Docker είναι ευάλωτο σε ARP spoofing και MAC flooding επειδή όλα τα εισερχόμενα πακέτα προωθούνται από τη γέφυρα στα επιθυμητά interfaces χωρίς να εκτελείται κανένα φιλτράρισμα. Όταν ένας επιτιθέμενος στέλνει πλαστά μηνύματα ARP μέσω ενός τοπικού δικτύου, αυτό ονομάζεται ARP spoofing. Σε αυτή την επίθεση, ο επιτιθέμενος θα συνδέσει τη διεύθυνση MAC του/της με τη διεύθυνση IP ενός εξουσιοδοτημένου χρήστη και θα αρχίσει να λαμβάνει κάθε πληροφορία από αυτή τη διεύθυνση IP. Σε μια τέτοια κατάσταση, ο επιτιθέμενος θα είναι σε θέση να υποκλέψει μυστικές πληροφορίες που μοιράζονται μεταξύ της εφαρμογής ιστού και των container και θα είναι επίσης σε θέση να εισάγει ένα κακόβουλο ωφέλιμο φορτίο στο δίκτυο.

### 4.4.6 Επιθέσεις Man-in-the-Middle

Στις επιθέσεις Man-in-the-Middle, ένας κακόβουλος χρήστης εισέρχεται στην επικοινωνία μεταξύ δύο μη κακόβουλων μερών και προσπαθεί να παρακολουθήσει, να τροποποιήσει και να κλέψει πολύτιμες πληροφορίες. που μοιράζονται μεταξύ τους τα δύο ή περισσότερα μη κακόβουλα μέρη.

### 4.4.7 Επιθέσεις μη εξουσιοδοτημένης πρόσβασης

Οι επιθέσεις μη εξουσιοδοτημένης πρόσβασης αναφέρονται σε άτομα που έχουν πρόσβαση στα δίκτυα, τα δεδομένα, τα τελικά σημεία, τις εφαρμογές ή τις συσκευές ενός οργανισμού, χωρίς να έχουν λάβει άδεια. Ένας κακόβουλος χρήστης που αποκτά πρόσβαση σε ένα νόμιμο σύστημα Docker μπορεί να προκαλέσει πολύ μεγαλύτερη ζημιά από ό,τι μπορούμε να φανταστούμε. Από την επίθεση σε όλα τα container του Docker μέχρι την επίθεση στον ίδιο το host, η μη εξουσιοδοτημένη πρόσβαση μπορεί να προκαλέσει ζημιά με πολλούς τρόπους. Ως εκ τούτου, κάθε είδους μη εξουσιοδοτημένη πρόσβαση θα πρέπει να αποτρέπεται.

### 4.4.8 Επιθέσεις Poisoned Images

Η εισαγωγή μολυσμένου λογισμικού ή η εκτέλεση ξεπερασμένων και ευάλωτων εκδόσεων λογισμικού μπορεί να προκαλέσει το πρόβλημα των Poisoned Images. Στο Docker, όταν κατεβάζουμε ένα Image τότε αυτό επαληθεύεται με βάση την παρουσία ενός υπογεγραμμένου μανιφέστου, αλλά το checksum της κατεβασμένης εικόνας από το μανιφέστο δεν πιστοποιείται ποτέ από το Docker. Εξαιτίας αυτού, ένας επιτιθέμενος με ένα υπογεγραμμένο μανιφέστο μπορεί να μεταδώσει οποιαδήποτε εικόνα, γεγονός που μπορεί να οδηγήσει σε σοβαρές ευπάθειες.

### 4.4.9 Επιθέσεις που προκύπτουν από ξεπερασμένο λογισμικό

Το ξεπερασμένο λογισμικό έχει πολλά προβλήματα που διορθώνονται στις ενημερωμένες εκδόσεις. Έτσι, η εκτέλεση ξεπερασμένου λογισμικού μπορεί να προκαλέσει σοβαρά προβλήματα ασφάλειας. Συνεπώς, το ξεπερασμένο λογισμικό θα πρέπει να αποφεύγεται.

### 4.4.10 Επιθέσεις σε περιβάλλον cloud

Οποιαδήποτε κυβερνοεπίθεση που στοχεύει σε off-site πλατφόρμες υπηρεσιών , οι οποίες προσφέρουν υπηρεσίες αποθήκευσης, υπολογιστικής ισχύς ή φιλοξενίας μέσω cloud, μπορεί να χαρακτηριστεί ως κυβερνοεπίθεση σε περιβάλλον cloud. Αυτό μπορεί να περιλαμβάνει επιθέσεις σε πλατφόρμες υπηρεσιών που χρησιμοποιούν μοντέλα παροχής υπηρεσιών όπως το SaaS, το IaaS και το PaaS. Τέτοιες επιθέσεις μπορούν να θέσουν σε κίνδυνο περιβάλλοντα Docker τα οποία είναι υλοποιημένα σε περιβάλλον cloud.

### 4.4.11 Επιθέσεις Code Injection

Σε αυτού του είδους τι επιθέσεις ο επιτιθέμενος έχει ως σκοπό την εκτέλεση κακόβουλου κώδικα προκειμένου να εκμεταλλευτεί ευπάθειες του συστήματος. Αυτές οι επιθέσεις μπορούν να θέσουν σε κίνδυνο και περιβάλλοντα Docker container αφού κάποιος κακόβουλος μπορεί να μολύνει ένα container εισάγοντας κακόβουλο λογισμικό σε αυτό, το οποίο μπορεί να διαφύγει και να επιτεθεί στον host. Ένα παράδειγμα αποτελεί η ευπάθεια, η οποία επιτρέπει σε ένα κακόβουλο container να αντικαταστήσει τη βιβλιοθήκη run του host και έτσι να αποκτήσει προνόμια επιπέδου root στον host. Για να γίνει αυτό, ο επιτιθέμενος πρέπει να τοποθετήσει ένα κακόβουλο container μέσα στο σύστημά μας, πράγμα που δεν είναι πολύ δύσκολο να γίνει.

### 4.4.12 Επιθέσεις στον kernel του συστήματος

Υπάρχει διάφορες ευπάθειες που έχουν βρεθεί ανά τα χρόνια τα οποία στοχεύουν και εκμεταλλεύονται τον kernel. Σε περιβάλλοντα Docker Container οι εφαρμογές που εκτελούνται μπορούν να εκτεθούν και να γίνουν αντικείμενο εκμετάλλευσης. Καθώς όλα τα container μοιράζονται τον ίδιο kernel, οπότε αν κάποια εφαρμογή υποκλαπεί και αποκτήσει κάποια προνομιακά δικαιώματα του kernel, τότε όλα τα τρέχοντα container καθώς και η πλατφόρμα του host μπορούν να εκτεθούν στον κακόβουλο.

### 4.4.13 Επιθέσεις Cryptojacking

Αυτού του είδους οι επιθέσεις είναι σχετικά νέες αλλά πολύ σημαντικές. Πρόκειται για επιθέσεις που έχουν ως σκοπό την εκμετάλλευση των υπολογιστικών πόρων (CPU) του θύματος για την εξόρυξη κρυπτονοσμισμάτων.

### 4.4.14 Επιθέσεις Παραποίησης

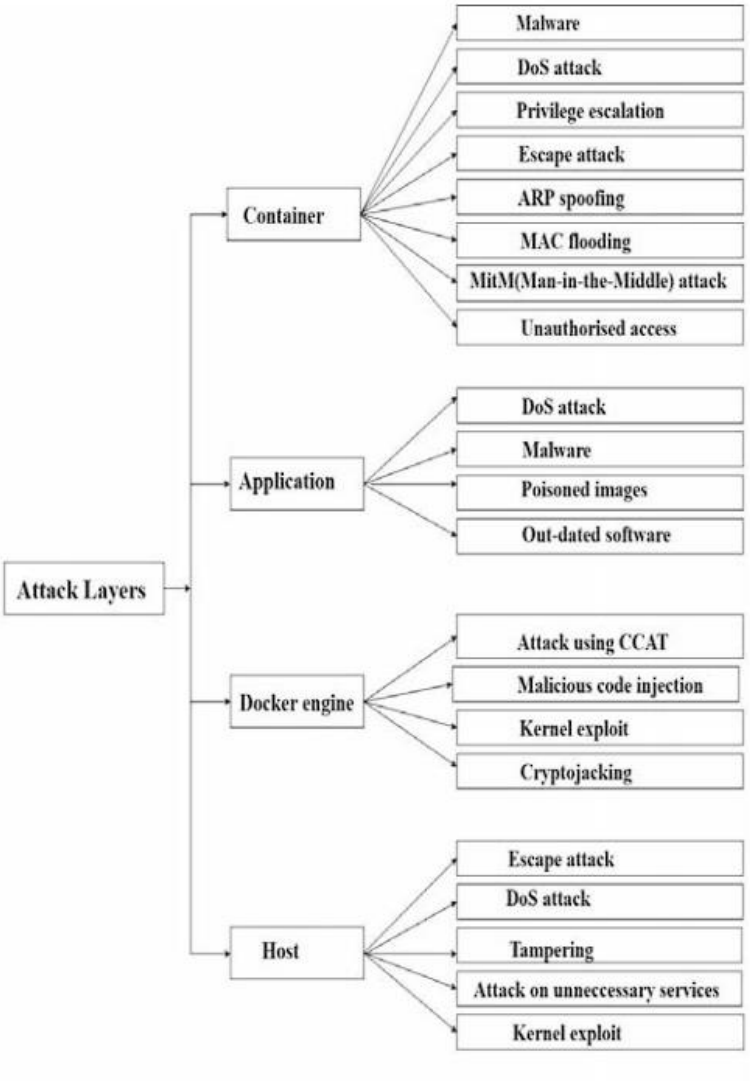
Οι μη ασφαλείς διαμορφώσεις των container μπορεί να οδηγήσουν σε κίνδυνο παραποίησης του host. Εάν επιτραπεί σε ένα container να προσαρτήσει ευαίσθητους καταλόγους στον host, το container μπορεί αργότερα να αλλάξει τα αρχεία σε αυτούς τους καταλόγους. Αυτές οι αλλαγές μπορεί να προκαλέσουν επιπτώσεις στην ασφάλεια και τη σταθερότητα του host και όλων των άλλα container που εκτελούνται σε αυτόν.

### 4.4.15 Επιθέσεις σε λοιπές υπηρεσίες

Οι υπηρεσίες είναι προγράμματα που ακούν και απαντούν στο δίκτυο. Ορισμένες υπηρεσίες επιτρέπουν επίσης την άμεση πρόσβαση στον υπολογιστή μας, όπως διακομιστές FTP, διακομιστές ιστού, διακομιστές αρχείων, διακομιστές ηλεκτρονικού ταχυδρομείου και διακομιστές μεσολάβησης, ανοικτές πόρτες κ.λπ. Οι μη απαραίτητες και μη ασφαλείς υπηρεσίες μπορούν να οδηγήσουν σε μια ανοιχτή πόρτα για τους επιτιθέμενους. Από την άλλη πλευρά, όσο περισσότερες υπηρεσίες

τρέχουν στον host, τόσο περισσότερες ευκαιρίες θα υπάρχουν για άλλους να τις χρησιμοποιήσουν, να εισβάλουν σε αυτές και να πάρουν τον έλεγχο του συστήματος μέσω αυτών.

Στη παρακάτω εικόνα βλέπουμε τα διάφορα attack layers ή attack vectors που αναλύσαμε πιο πάνω καθώς σε ποιο από αυτά αντιστοιχούν ποιες από τις επιθέσεις που αναλύσαμε.




*Εικόνα 11 Αντιστοίχιση επιθέσεων σε layer*

## 4.5 Capabilities και ασφάλεια

### 4.5.1 Εισαγωγή

Τα capabilities παρέχουν ένα υποσύνολο των διαθέσιμων προνομίων root σε μια διεργασία. Αυτό ουσιαστικά διασπά τα προνόμια root σε μικρότερες και διακριτές μονάδες. Καθεμία από αυτές τις μονάδες μπορεί στη συνέχεια να παραχωρηθεί ανεξάρτητα σε διεργασίες. Με αυτόν τον τρόπο μειώνεται το πλήρες σύνολο των προνομίων και μειώνονται οι κίνδυνοι εκμετάλλευσης. Για παράδειγμα, ας υποθέσουμε ότι εκτελούμε μια διεργασία ως κανονικός χρήστης. Αυτό σημαίνει ότι είμαστε μη προνομιούχοι χρήστες. Μπορούμε να έχουμε πρόσβαση μόνο σε δεδομένα που ανήκουν σε εμάς, στην ομάδα (group) μας ή τα οποία είναι σημειωμένα για πρόσβαση από όλους τους χρήστες. Κάποια στιγμή, η διεργασία μας χρειάζεται λίγο περισσότερα δικαιώματα για να εκπληρώσει τα καθήκοντά της, όπως το άνοιγμα ενός network socket. Το πρόβλημα είναι ότι οι κανονικοί χρήστες δεν μπορούν να ανοίξουν μια υποδοχή, καθώς αυτό απαιτεί δικαιώματα root. Έτσι τα capabilities έρχονται να δώσουν λύσει σε τέτοιου είδους προβλήματα. Στην παραπάνω περίπτωση μπορεί να δοθεί το capability στο χρήστη να ανοίξει το network socket όμως να μην του δοθεί κανένα άλλο προνόμιο root.

### 4.5.2 Docker Capabilities

Το Docker χρειάζεται κάποια capabilities για τις βασικές του λειτουργίες. Έτσι από προεπιλογή, κάθε Docker container εκκινείτε μόνο με τα απαραίτητα ελάχιστα capabilities. Τα capabilities αυτά είναι τα παρακάτω:

- CAP_CHOWN
- CAP_DAC_OVERRIDE
- CAP_FOWNER
- CAP_FSETID
- CAP_KILL
- CAP_SETGID
- CAP_SETUID
- CAP_SETPCAP

- CAP_NET_BIND_RAW
- CAP_NET_RAW
- CAP_SYS_CHROOT
- CAP_MKNOD
- CAP_AUDIT_WRITE
- CAP_SETFCAP

Θα δούμε αναλυτικότερα σε τι αναφέρεται κάθε ένα από αυτά τα capabilities καθώς και επιπλέον άλλα που υπάρχουν και πως αυτά μπορούν να θέσουν σε κίνδυνο το Docker. Για να προσθέσουμε και να αφαιρέσουμε capabilities όταν τρέχουμε κάποιο container μπορούμε να χρησιμοποιήσουμε τα flags:

```
Add Capability:--cap-add=<CAP_TO_ADD>
Remove Capability: --cap-drop=<CAP_TO_DROP>
```

Μπορούμε να ελέγξουμε τα capabilities ενός Docker container χρησιμοποιώντας το εκτελέσιμο capsh με την παρακάτω εντολή:

```
$ capsh --print
Current: =
cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_s
etgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadc
ast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_ra
wio,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+ep
Bounding set
=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_broad
cast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_r
awio,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
Securebits: 00/0x0/1'b0
secure-noroot: no (unlocked)
secure-no-suid-fixup: no (unlocked)
secure-keep-caps: no (unlocked)
uid=0(root)
gid=0(root)
groups=0(root)
```

Χρησιμοποιώντας το capsh μπορούμε να δούμε τα capabilities που έχει ένα container ώστε να μπορέσουμε να αξιολογήσουμε στη συνέχεια την ασφάλεια του. Πιο κάτω θα δούμε διάφορα capabilities και πως αυτά μπορούν να επηρεάσουν την ασφάλεια σε ένα σύστημα.

### 4.5.3 CAP_SYS_ADMIN

Το CAP_SYS_ADMIN είναι σε μεγάλο βαθμό ένα capability που μπορεί εύκολα να οδηγήσει σε πρόσθετα capabilities ή σε πλήρη root capabilities. Το CAP_SYS_ADMIN απαιτείται για την εκτέλεση μιας σειράς λειτουργιών root, η οποία είναι δύσκολο να εγκαταλειφθεί από τα container, εάν οι προνομιακές λειτουργίες εκτελούνται εντός του container. Η διατήρηση αυτού του capability είναι συχνά απαραίτητη για container που μιμούνται ολόκληρα συστήματα έναντι μεμονωμένων container εφαρμογών που μπορεί να είναι πιο περιορισμένα. Μεταξύ άλλων, αυτό επιτρέπει την προσάρτηση συσκευών ή την κατάχρηση του release_agent για την διαφυγή από το container.

Στην περίπτωση του Docker όταν ένα container έχει το capability CAP_SYS_ADMIN μπορεί κάποιος μέσα από το container να κάνει mount το δίσκο του host και έτσι να έχει πρόσβαση σε αυτόν. Το γεγονός απόκτησης πρόσβαση στο δίσκο του host μέσα από το container αποτελεί μεγάλο κενό ασφαλείας καθώς πρόκειται για docker escape και ο κακόβουλος όχι μόνο μπορεί να έχει πρόσβαση στα αρχεία του host αλλά μπορεί και να τα παραποιήσει ή και να δημιουργήσει νέα.

### 4.5.4 CAP_SYS_PTRACE

Το CAP_SYS_PTRACE επιτρέπει τη χρήση του ptrace και των προσφάτως εισαχθέντων sys calls cross memory attach, όπως οι process_vm_readv και process_vm_writev. Εάν παραχωρηθεί αυτή το capability και η ίδια η κλήση συστήματος ptrace δεν έχει μπλοκαριστεί από ένα φίλτρο seccomp, αυτό θα επιτρέψει σε έναν εισβολέα να παρακάμψει άλλους περιορισμούς του seccomp.

Αν παρατηρήσουμε ότι σε ένα container έχει παραχωρηθεί το capability CAP_SYS_PTRACE τότε μπορούμε να πραγματοποιήσουμε επίθεση docker escape κάνοντας inject κακόβουλο shellcode σε μια διεργασία του host. Επιπλέον, εάν μέσα

στο container υπάρχει το εκτελέσιμο gdb τότε μπορούμε να το χρησιμοποιήσουμε για να κάνουμε debug μια διεργασία του host και να την κάνουμε να καλέσει τη συνάρτηση system προκειμένου να εκτελέσουμε ότι εντολή θέλουμε. Αυτό μπορούμε να το κάνουμε τις παρακάτω εντολές:

```
# Το output της εντολής δεν φαίνεται στο τερματικό οπότε μπορούμε να χρησιμοποιήσουμε κάποια εντολή reverse shell.
$ gdb -p <pid>
(gdb) call (void)system("<command_to_execute>")
```

### 4.5.5 CAP_SYS_MODULE

Το CAP_SYS_MODULE επιτρέπει στη διεργασία να φορτώνει και να αφαιρεί αυθαίρετα kernel module (κλήσεις συστήματος init_module, finit_module και delete_module). Αυτό θα μπορούσε να οδηγήσει σε κλιμάκωση προνομίων και διακινδύνευση του ring-0. Ο kernel μπορεί να τροποποιηθεί κατά βούληση, υπονομεύοντας όλη την ασφάλεια του συστήματος. Αυτό σημαίνει ότι μπορούμε να εισάγουμε/αφαιρούμε kernel modules στον/από τον kernel του host.

Ένας κακόβουλος μπορεί να εκμεταλλευτεί ένα container που του έχει παραχωρηθεί το capability CAP_SYS_MODULE για να πραγματοποιήσει container escape. Για να το κάνει αυτό θα πρέπει να φτιάξει ένα kernel module το οποίο εκτελεί ένα reverse shell όπως επίσης και το αντίστοιχο Makefile το οποίο κάνει compile αυτό το reverse shell module. Όταν ο κακόβουλος φορτώσει το kernel module τότε θα λάβει πίσω σύνδεση από το reverse shell. Και θα έχει πλήρες shell στον host [18].

### 4.5.6 CAP_DAC_READ_SEARCH

Το CAP_DAC_READ_SEARCH επιτρέπει σε μια διεργασία να παρακάμπτει τα δικαιώματα ανάγνωσης αρχείων, ανάγνωσης καταλόγων και εκτέλεσης. Αν και σχεδιάστηκε για να χρησιμοποιείται για αναζήτηση ή ανάγνωση αρχείων, παρέχει

επίσης στη διεργασία την άδεια να καλεί την open_by_handle_a. Οποιαδήποτε διεργασία με το capability CAP_DAC_READ_SEARCH μπορεί να χρησιμοποιήσει την open_by_handle_at για να αποκτήσει πρόσβαση σε οποιοδήποτε αρχείο, ακόμη και σε αρχεία εκτός του χώρου ονομάτων της προσάρτησης. Το handle που περνάει στην open_by_handle_at προορίζεται να είναι ένα αδιαφανές αναγνωριστικό που ανακτάται με τη χρήση της name_to_handle_at. Ωστόσο, αυτό το handle περιέχει ευαίσθητες και παραποιήσιμες πληροφορίες, όπως αριθμούς inode. Αυτό αποδείχτηκε για πρώτη φορά ότι αποτελεί πρόβλημα στα Docker container από τον Sebastian Krahmer με το exploit shocker. Αυτό σημαίνει ότι ένας κακόβουλος μπορεί όχι απλά να παρακάμψει τους ελέγχους δικαιωμάτων ανάγνωσης αρχείων και τους ελέγχους δικαιωμάτων ανάγνωσης/εκτέλεσης καταλόγων αλλά και να έχει πρόσβαση σε αρχεία άλλων διεργασιών/container. Όπως καταλαβαίνουμε και αυτό το capability μπορεί να οδηγήσει σε Docker escape.

### 4.5.7 CAP_DAC_OVERRIDE

Αυτό το capability επιτρέπει σε μια διεργασία να παρακάμπτει τους ελέγχους δικαιωμάτων και να μπορεί να γράφει σε οποιοδήποτε αρχείο. Σε ένα περιβάλλον Docker container αυτό μπορεί να αποτελέσει αίτιο container escape καθώς ένας επιτιθέμενος μπορεί να γράψει νέα ή υπάρχοντα αρχεία του host. Ένα παράδειγμα επίθεση μπορεί να αποτελεί ότι ο επιτιθέμενος μέσα από το container τροποποιεί τα αρχεία `/etc/shadow` και `/etc/passwd` προκειμένου να προσθέσει ένα καινούριο χρήστη στο σύστημα με τον οποίο για παράδειγμα να μπορεί να συνδεθεί μέσω ssh.

### 4.5.8 CAP_CHOWN

Αυτό το capability επιτρέπει σε μια διεργασία να μπορεί να αλλάζει τον ιδιοκτήτη οποιουδήποτε αρχείου του συστήματος. Αυτό το capability εκχωρείται από προεπιλογή στο root user του container.

### 4.5.9 CAP_FOWNER

Με αυτό το capability μια διεργασία μπορεί να αλλάζει τα δικαιώματα οποιουδήποτε αρχείου του συστήματος. Είναι και αυτό ένα από τα capabilities που εκχωρούνται από προεπιλογή όταν εκκινείτε ένα container.

### 4.5.10 CAP_SETUID

Με αυτό το capability μια διεργασία μπορεί να ορίσει το UID της διεργασίας που δημιουργείται. Είναι και αυτό ένα από τα capabilities που εκχωρούνται από προεπιλογή όταν εκκινείτε ένα container.

### 4.5.11 CAP_SETGID

Με αυτό το capability μια διεργασία μπορεί να ορίσει το GID της διεργασίας που δημιουργείται. Είναι και αυτό ένα από τα capabilities που εκχωρούνται από προεπιλογή όταν εκκινείτε ένα container.

### 4.5.12 CAP_SETFCAP

Το CAP_SETFCAP είναι ένα capability το οποίο επιτρέπει σε μια διεργασία να εκχωρεί capabilities σε αρχεία και διεργασίες. Πρόκειται για ένα capability το οποίο εκχωρείται από προεπιλογή σε ένα container. Ένας επιτιθέμενος μπορεί να σκεφτεί ότι αφού έχει τη δυνατότητα να εκχωρεί οποιοδήποτε capability μέσα στο container τότε θα μπορεί να εκχωρείς το CAP_SYS_ADMIN στο gdb και να πραγματοποιήσει έτσι container escape όπως είδαμε και πιο πάνω. Όμως αυτό δεν γίνεται καθώς επιτρέπεται να προσθέτουμε capabilities στο inheritable set μόνο από το bounding set και έτσι δεν μπορούμε να προσθέσουμε όποια θέλουμε όπως το CAP_SYS_ADMIN.

### 4.5.13 CAP_SYS_RAWIO

Το CAP_SYS_RAWIO παρέχει μια σειρά ευαίσθητων λειτουργιών, συμπεριλαμβανομένης της πρόσβασης στα /dev/mem, /dev/kmem ή /proc/kcore, της τροποποίησης της mmap_min_addr, της πρόσβασης στις κλήσεις συστήματος ioperm και iop και διάφορες εντολές δίσκου. Το FIBMA ενεργοποιείται επίσης μέσω αυτού του capability, γεγονός που έχει προκαλέσει προβλήματα στο παρελθόν. Με αυτό το capability ένα container είναι ευάλωτο καθώς μπορεί να προκληθεί container escape.

### 4.5.14 CAP_KILL

Με αυτό το capability μια διεργασία μπορεί να “σκοτώσει” οποιαδήποτε άλλη διεργασία. Είναι και αυτό ένα από τα capabilities που εκχωρούνται από προεπιλογή όταν εκκινείτε ένα container.

### 4.5.15 CAP_NET_BIND_SERVICE

Με αυτό το capability μια διεργασία μπορεί να ακούει σε οποιοδήποτε port ακόμα και σε αυτά που χρειάζονται δικαιώματα root. Είναι και αυτό ένα από τα capabilities που εκχωρούνται από προεπιλογή όταν εκκινείτε ένα container.

### 4.5.16 CAP_NET_RAW

Το CAP_NET_RAW επιτρέπει σε μια διεργασία να μπορεί να δημιουργεί τύπους RAW και PACKET socket για τα διαθέσιμα namespaces. Αυτό επιτρέπει την αυθαίρετη δημιουργία και μετάδοση πακέτων μέσω των εκτεθειμένων διεπαφών δικτύου. Σε πολλές περιπτώσεις αυτή η διεπαφή θα είναι μια εικονική συσκευή Ethernet, η οποία μπορεί να επιτρέψει σε ένα κακόβουλο ή παραβιασμένο container να παραποιήσει πακέτα σε διάφορα επίπεδα δικτύου. Μια κακόβουλη διεργασία ή ένα παραβιασμένο container με αυτή τη δυνατότητα μπορεί να εισχωρήσει σε upstream γέφυρα δικτύου, να εκμεταλλευτεί τη δρομολόγηση μεταξύ containers, να παρακάμψει τους ελέγχους πρόσβασης στο δίκτυο και να αλλοιώσει με άλλο τρόπο

τη δικτύωση του host, εάν δεν υπάρχει τείχος προστασίας για τον περιορισμό των τύπων και του περιεχομένου των πακέτων. Τέλος, αυτό το capability επιτρέπει στη διεργασία να δεσμεύεται σε οποιαδήποτε διεύθυνση εντός των διαθέσιμων namespaces. Αυτό το capability συχνά διατηρείται από προνομιούχα container για να επιτρέπει τη λειτουργία του ping με τη χρήση υποδοχών RAW για τη δημιουργία αιτημάτων ICMP από ένα container και εκχωρείται από προεπιλογή κατά την εκκίνηση του container.

### 4.5.17 CAP_NET_ADMIN

Το CAP_NET_ADMIN επιτρέπει στον κάτοχο του capability να τροποποιεί το τείχος προστασίας, τους πίνακες δρομολόγησης, τα δικαιώματα υποδοχής, τη διαμόρφωση της διασύνδεσης δικτύου και άλλες σχετικές ρυθμίσεις στις εκτεθειμένες διασυνδέσεις δικτύου. Αυτό παρέχει επίσης τη δυνατότητα να ενεργοποιεί τη λειτουργία promiscuous mode για τις συνδεδεμένες διασυνδέσεις δικτύου και ενδεχομένως μπορεί παρακολουθεί την κίνηση σε όλα τα namespaces.

### 4.5.18 CAP_SYS_CHROOT

Αυτό το capability επιτρέπει τη χρήση της κλήσης συστήματος chroot. Είναι και αυτό ένα από τα capabilities που εκχωρούνται από προεπιλογή όταν εκκινείτε ένα container.

## 4.6 Γνωστές ευπάθειες Docker

Παρακάτω συγκεντρώνονται σε ένα πίνακα οι κυριότερες ευπάθειες στο Docker συνοπτικά.

| Όνομα | Περιγραφή |
|---|---|
| **Άδειες Docker** | Μια συνηθισμένη εσφαλμένη διαμόρφωση (misconfiguration) είναι η παροχή πρόσβασης σε μη προνομιούχους χρήστες Docker, το οποίο τους επιτρέπει να δημιουργούν, να ξεκινούν και να αλληλεπιδρούν με το container. |
| **Αναγνώσιμα αρχεία διαμόρφωσης Docker** | Τα αρχεία διαμόρφωσεις Docker (όπως τα my αρχεία του docker compose) μπορούν να περιέχουν ευαίσθητες πληροφορίες. Αν μπορεί να διαβάσει ο κάθε χρήστης αυτά τα αρχεία τότε αυτό είναι επικίνδυνο για το σύστημα. |
| **Προνομιακή λειτουργία (privileged mode) Docker** | Η λειτουργία αυτή επιτρέπει την πρόσβαση σε όλες τις συσκευές του host και τις δυνατότητες του πυρήνα. Το μειονέκτημα της προνομιακής λειτουργίας είναι ότι όλες οι λειτουργίες του πυρήνα επιτρέπουν σε έναν επιτιθέμενο μέσα στο container να διαφύγει και να αποκτήσει πρόσβαση στο host. |
| **Δυνατότητες (Capabilities) Docker** | Τα Docker container ξεκινούν με ελάχιστες δυνατότητες, αλλά είναι δυνατή η προσθήκη επιπλέον δυνατοτήτων κατά το χρόνο εκτέλεσης. Η παροχή επιπλέον δυνατοτήτων στο container του δίνει την άδεια να εκτελεί ορισμένες ενέργειες. Μερικές από αυτές τις ενέργειες επιτρέπουν τις αποδράσεις από το container. |
| **Docker Socket** | Το Docker Socket είναι ο τρόπος που χρησιμοποιούν οι clients για να επικοινωνήσουν με το Docker daemon. Ένας επιτιθέμενος μπορεί να στείλει κατευθείαν HTTP Requests στο Docker Socket |

*Πίνακας 3 Γνωστές ευπάθειες Docker*

## 4.7 Γνωστά Docker CVEs

### 4.7.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα δούμε κάποια από τα σημαντικότερα CVEs που υπάρχουν στο Docker και θέτουν σε κίνδυνο την ασφάλεια του συστήματος.

### 4.7.2 CVE-2018-8115

Η ευπάθεια αυτή επηρεάζει το Docker for Windows και όχι γενικά το Docker. Η ουσία της ευπάθειας είναι ότι κατά τη διαδικασία `image pull`, τα αρχεία από ένα κακόβουλο Docker Image μπορούν να εξαχθούν σε οποιονδήποτε κατάλογο στο σύστημα αρχείων του host. Αυτό συμβαίνει ως μέρος της διαδικασίας `αποσυμπίεσης` του Docker Image, όπου ο κώδικας που επεξεργάζεται το αρχείο tar, ενώνει τον κατάλογο προορισμού με τη διαδρομή αρχείου που καθορίζεται σε ένα αρχείο [19].

Η ευπάθεια έγκειται στο ότι αυτή η διαδρομή του αρχείου δεν ελέγχεται για παράδειγμα ένα αρχείο του Docker Image μπορεί να περιλαμβάνει διάσχιση καταλόγου (directory traversal π.χ ../../../../). Το αρχείο προορισμού μπορεί να εγγραφεί σε αυθαίρετη τοποθεσία στο host.

Όσον αφορά την εκμετάλλευση, ένα απλό `docker pull` του ειδικά διαμορφωμένου κακόβουλου Docker Image αρκεί.

### 4.7.3 CVE-2018-15664

Η ευπάθεια αυτή επιτρέπει σε ένα container να έχει πρόσβαση εγγραφής και ανάγνωσης διαχειριστή στο σύστημα αρχείων του host [20]. Πρόκειται για μια ευπάθεια symlink race condition. Ουσιαστικά η ευπάθεια αυτή εκμεταλεύεται το γεγονός ότι χρειάζεται ένα χρονικό διάστημα προκειμένου να γίνει έλεγχος για το αν ένα symlink με σε ένα container είναι σε ασφαλές path [21]. Έτσι, ένας κακόβουλος μπορεί να στην αρχή να έχει το symlink σε μια ασφαλή τοποθεσία και στο χρονικό

διάστημα που γίνεται ο έλεγχος να προλάβει να αλλάξει το symlink και έτσι να έχει πρόσβαση σε αρχεία που δεν θα έπρεπε.

### 4.7.4 CVE-2019-5736

Το CVE-2019-5736 πρόκειται για μια σοβαρή ευπάθεια στο Docker και συγκεκριμένα στο runC. Ουσιαστικά, επιτρέπει στους επιτιθέμενους να αντικαταστήσουν το binary αρχείο runc του host (και κατά συνέπεια να αποκτήσουν πρόσβαση root στον host) αξιοποιώντας την ικανότητα εκτέλεσης μιας εντολής ως root μέσα σε ένα νέο container με ένα Docker Image που ελέγχεται από τον επιτιθέμενο ή σε ένα υπάρχον container στο οποίο ο επιτιθέμενος έχει πρόσβαση εγγραφής σε αυτό και μπορεί να χρησιμοποιήσει την εντολή `docker exec` σε αυτό [22]. Το runC μπορεί να εκτελέσει τον εαυτό του μέσα στο container, λέγοντας στο container να ξεκινήσει το /proc/self/exe το οποίο κατά τη διάρκεια της εκκίνησης συνδέεται με το binary runC [23]. Το /proc/self/exe στο container είναι symlink στο binary runC του host. Ο χρήστης root στο container είναι στη συνέχεια σε θέση να αντικαταστήσει το binary runC του host χρησιμοποιώντας αυτή το symlink. Μόλις εκτελεστεί ξανά το runC θα εκτελεστεί το binary που ο κακόβουλος έχει βάλλει στη θέση του. Έτσι, ένας επιτιθέμενος έχει τη δυνατότητα να εκτελέσει κώδικα στον host.

### 4.7.5 CVE-2019-5021

Η ευπάθεια αυτή πρόκειται για μια ευπάθεια που βρίσκεται στο Alpine Linux Docker Image. Η ευπάθεια οφείλεται στον κωδικό πρόσβασης του διαχειριστή (root) ο οποίος έχει οριστεί, από προεπιλογή, σε NULL [24]. Για να δούμε αν ένα Docker Image είναι ευάλωτο μπορούμε να διαβάσουμε το αρχείο `/etc/shadow`. Το αρχείο /etc/shadow αποθηκεύει τον κωδικό πρόσβασης του χρήστη σε κατακερματισμένη μορφή. Έχει μια ειδική μορφή στην οποία κάθε γραμμή έχει το όνομα του χρήστη και πεδία (διαχωρισμένα με άνω και κάτω τελεία) που καθορίζουν πληροφορίες σχετικά με τον κωδικό πρόσβασης, όπως ο κατακερματισμένος κωδικός πρόσβασης, ο χρόνος αλλαγής και ο χρόνος λήξης. Αν διαβάσουμε το αρχείο και δούμε ότι δεν υπάρχει

κωδικός για το χρήστη root τότε πρόκειται για ένα ευάλωτο Docker Image. Μπορούμε να χρησιμοποιήσουμε την παρακάτω εντολή:

```
$ cat /etc/shadow
root:::0:::::
```

### 4.7.6 CVE-2020-13401

Ένας επιτιθέμενος μέσα σε ένα container, με εκχωρημένο το capability CAP_NET_RAW, μπορεί να λαμβάνει και να εφαρμόζει ψεύτικα μηνύματα RA (Router Advertisement) από άλλα container στο δίκτυο [25]. Η λήψη RA είναι μια φυσιολογική συμπεριφορά του λειτουργικού συστήματος, αλλά αν ο αποστολέας RA δεν είναι αξιόπιστος στο δίκτυο, το container θύμα μπορεί να λάβει το μήνυμα και να ενταχθεί στο δίκτυο και στη συνέχεια να στείλει όλα τα πακέτα δικτύου στο νέο ψεύτικο δρομολογητή (επίθεση man-in-the-middle).

### 4.7.7 Συνοπτικά Docker CVEs

Παρακάτω υπάρχει πίνακας με τα Docker CVEs είδαμε συνοπτικά σε ένα πίνακα

| CVE | Όνομα | Περιγραφή |
|---|---|---|
| **CVE-2018-8115** | Jack-in-the-box | Επιτρέπει στους επιτιθέμενους να εισάγουν κακόβουλο κώδικα σε Docker Images. |
| **CVE-2018-15664** | Docker cp | Απομακρυσμένοι επιτιθέμενοι μπορούν να εκμεταλλευτούν αυτήν την ευπάθεια του Docker για να αποκτήσουν πρόσβαση root στο σύστημα αρχείων του κεντρικού υπολογιστή, παραβιάζοντας μια εκκινημένη λειτουργία Docker cp. |
| **CVE-2019-5736** | runC container escape | Αυτή είναι μια ευπάθεια Docker που επιτρέπει σε επιτιθέμενους να αποκτήσουν πρόσβαση σε επίπεδο root στο σύστημα ενός κεντρικού υπολογιστή αντικαθιστώντας το δυαδικό αρχείο runC. Εάν η επίθεση είναι επιτυχής, ο επιτιθέμενος αποκτά τον έλεγχο του συστήματος του κεντρικού υπολογιστή, όπου έχει απεριόριστη πρόσβαση στον διακομιστή και πρόσβαση σε τυχόν container που έχουν αναπτυχθεί σε αυτόν τον διακομιστή. |
| **CVE-2019-5021** | Alpine Linux: NULL root password | Πρόκειται για μια ευπάθεια η οποία βρισκόταν στο Alpine Linux Docker Image το οποίο περιείχε root user με null κωδικό. |
| **CVE-2020-13401** | IPv6 input validation | Επιτρέπει στους επιτιθέμενους να εκτελέσουν μια επίθεση man-in-the-middle (MitM) εναντίον άλλου container ή του δικτύου του κεντρικού υπολογιστή. |

*Πίνακας 4 Γνωστά Docker CVEs*

# 5. Δοκιμές Παρείσδυσης στο Docker

## 5.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο είδαμε τις διάφορες πτυχές ασφαλείας σε περιβάλλοντα Docker και πως μπορεί ένα τέτοιο σύστημα να τεθεί σε κίνδυνο. Σε αυτό το κεφάλαιο το πως μπορούμε να πραγματοποιήσουμε δοκιμές παρείσδυσης σε τέτοια περιβάλλοντα και θα ορίσουμε μια λίστα ελέγχου σε μορφή ερωτήσεων για να διευκολύνουμε τη διαδικασία.

## 5.2 Ορισμός του τύπου δοκιμής παρείσδυσης

Όπως είδαμε στο Κεφάλαιο 2 υπάρχουν διάφοροι τύποι δοκιμών παρείσδυσης με τους πιο γνωστούς να είναι οι δοκιμές Black Box και White Box. Εν συντομία, όταν αναφερόμαστε σε μια Black Box δοκιμή εννοούμε ότι αυτός που πραγματοποιεί τη δοκιμή δεν γνωρίζει τίποτα για το σύστημα το οποίο ελέγχει, κάποιος μπορεί να πει ότι προσομοιάζει ακριβώς κάποιον εξωτερικό κακόβουλο που μπορεί να επιτεθεί στο σύστημα. Από την άλλη όταν αναφερόμαστε σε White Box δοκιμές τότε εννοούμε ότι αυτός που πραγματοποιεί τη δοκιμή γνωρίζει ακριβώς για τι σύστημα πρόκειται όλες τις λειτουργίες και τις παραμέτρους του συστήματος ακόμα και το πηγαίο κώδικα πιθανών εφαρμογών. Στην περίπτωση των συστημάτων Docker container ο τύπος της δοκιμής παρείσδυσης παίζει ρόλο καθώς στην Black Box περίπτωση αύτος που πραγματοποιεί τη δοκιμή δεν ξέρει στην πραγματικότητα καν αν το σύστημα έχει να κάνει με Docker και πρέπει να το ανακαλύψει από μόνος του. Στη White Box περίπτωση αυτός που πραγματοποιεί τη δοκιμή γνωρίζει όλη την αρχιτεκτονική του συστήματος, άρα και ότι πρόκειται για ένα σύστημα Docker καθώς και όλα τα container που σχετίζονται με το εύρος της δοκιμής. Στη συνέχεια, θα δούμε πως πραγματοποιούμε ελέγχους σε περιβάλλον Docker container χωρίζοντας τη διαδικασία σε δύο περιπτώσεις:

1. Βρισκόμαστε σε host που τρέχει Docker containers
2. Βρισκόμαστε μέσα σε Docker container

## 5.3 Δοκιμές παρείσδυσης σε host που τρέχει Docker containers

### 5.3.1 Εισαγωγή

Πραγματοποιώντας δοκιμές παρείσδυσης σε host που υλοποιεί ένα περιβάλλον Docker container πέρα από τις ευπάθειες που μπορεί να έχει το ίδιο το σύστημα (π.χ ευπάθειες πυρήνα, παλιά έκδοση συστήματος κ.α) τα όποια δεν θα καλύψουμε σε αυτή την εργασία μας ενδιαφέρει να δούμε πως είναι διαμορφωμένο το περιβάλλον Docker προκειμένου να δούμε μήπως εντοπίσουμε κάποιο κενό ασφαλείας.

### 5.3.2 Αναγνώριση της έκδοσης Docker

Το πρώτο πράγμα που πρέπει να κάνουμε κατά την πραγματοποίηση του ελέγχου παρείσδυσης είναι να αναγνωρίσουμε την έκδοση Docker που είναι εγκατεστημένη στο σύστημα. Αυτό μπορούμε να το κάνουμε εκτελώντας την παρακάτω εντολή:

```
$ docker -v
```

Η εντολή αυτή θα μας δώσει την έκδοσης του Docker την οποία στη συνέχεια θα την χρησιμοποιήσουμε για να αναζητήσουμε σε κάποια βάση δεδομένων με CVE για τυχόν γνωστές ευπάθειες που σχετίζονται με την έκδοση που αντιμετωπίζουμε.

### 5.3.3 Αναγνώριση χρηστών που μπορούν να τρέξουν το Docker

Είναι πολύ σημαντικό σε ένα σύστημα κάθε χρήστης να μπορεί να κάνει μόνο τις απολύτως απαραίτητες ενέργειες που χρειάζεται. Έτσι σαν δεύτερο έλεγχο που θα πρέπει να πραγματοποιήσουμε είναι να δούμε ποιοι χρήστες του συστήματος μπορούν να τρέξουν εντολές Docker καθώς όπως είδαμε στο προηγούμενο κεφάλαιο όποιος μπορεί να τρέξει εντολές Docker είναι σαν να έχει δικαιώματα root στον host. Το ποιος έχει δικαίωμα να τρέξει εντολές Docker μπορούμε να το ελέγξουμε με δύο τρόπους. Πρώτα, μπορούμε να ελέγξουμε ποιοι χρήστες ανήκουν στο Docker group. Αυτό μπορούμε να το κάνουμε διαβάζοντας το αρχείο /etc/group με την παρακάτω εντολή:

```
$ cat /etc/group | grep docker
```

Επιπλέον, ένας δεύτερος τρόπος είναι να ελέγξουμε ποιοι έχουν δικαιώματα εγγραφής και ανάγνωσης του docker socket εφόσον το docker socket είναι mounted. Για να βρούμε το docker socket μπορούμε να τρέξουμε την εντολή:

```
$  find / -name docker.sock 2>/dev/null
```

Τέλος, μπορούμε να δούμε αν το docker έχει ενεργοποιημένο το setuid bit.

### 5.3.4 Αναγνώριση των Docker Images που υπάρχουν στο host

Προκειμένου να έχουμε μια ολοκληρωμένη εικόνα του συστήματος οφείλουμε να ελέγξουμε πόσα και ποια Docker Images υπάρχουν στο host πότε δημιουργήθηκαν και αν τρέχουν ή όχι. Για να το κάνουμε αυτό μπορούμε να τρέξουμε την εντολή:

```
$   docker images -a
```

η οποία θα μας εμφανίσει όλες αυτές τις πληροφορίες που προαναφέραμε. Επιπλέον, για κάθε container που μας ενδιαφέρει μπορούμε να λάβουμε περισσότερες πληροφορίες με την εντολή:

```
$ docker inspect <id>
```

Με το docker inspect μας ενδιαφέρει να δούμε τυχόν environment variables που μπορεί να περνάνε στο container με ευαίσθητους κωδικούς π.χ κωδικούς της βάσης δεδομένων. Τέλος, με το inspect μπορούμε να δούμε τυχόν volumes των container και έτσι να βρούμε τυχόν ευαίσθητες πληροφορίες.

Αφού ελέγξουμε ποια images υπάρχουν στο host μπορούμε να αποθηκεύσουμε το docker image που μας ενδιαφέρει προκειμένου να το μεταφέρουμε σε κάποιο άλλο μηχάνημα για ανάλυση με την εντολή:

```
$ docker save -o <image_name>.tar <image_name>
```

Μπορούμε να φορτώσουμε το docker image σε άλλο μηχάνημα με την εντολή:

```
$ docker load < <image_name>
```

Για την ανάλυση μπορούμε να χρησιμοποιήσουμε ένα εργαλεία που ονομάζεται grype [26] το οποίο θα δούμε αναλυτικότερα σε επόμενο κεφάλαιο. Μπορούμε να ξεκινήσουμε την ανάλυση με την εντολή:

```
$ grype <image_name>
```

Το εργαλείο αυτό θα πραγματοποιήσει ανάλυση στο docker image και θα μας παρουσιάσει πιθανές ευπάθειες που μπορεί να υπάρχουν.

### 5.3.5 Αναγνώριση αρχείων διαμόρφωσης

Εφαρμόζοντας ελέγχους παρείσδυσης στο host ενός περιβάλλοντος Docker Container μας ενδιαφέρει να δούμε διάφορα αρχεία διαμόρφωσης του Docker για τυχόν κινδύνους που μπορεί να προκύψουν από αυτά. Συνήθως, ψάχνουμε για ευαίσθητες πληροφορίες όπως διαπιστευτήρια τα οποία είναι γραμμένα σε αρχεία διαμόρφωσης και σε περίπτωσης εισβολής θα μπορούσαν να θέσουν σε μεγάλο κίνδυνο στο σύστημα. Τέτοια αρχεία μπορεί να είναι το docker-compose.yml αν ο host χρησιμοποιεί docker-compose ή το daemon.json το οποίο αποτελεί αρχείο διαμόρφωσης για το Docker Daemon και μπορεί να μας δείξει σημαντικές πληροφορίες όπως για παράδειγμα ότι το Docker Socket έχει διαμορφωθεί να ακούει σε HTTP πράγμα που θέτει το σύστημα σε σοβαρό κίνδυνο.

## 5.4 Δοκιμές παρείσδυσης μέσα σε Docker containers

### 5.4.1 Εισαγωγή

Αρχικά, με κάποιο τρόπο θα πρέπει να αναγνωρίσουμε αν βρισκόμαστε μέσα σε ένα Docker Container καθώς για παράδειγμα σε έναν Black Box έλεγχο μπορεί να έχουμε μια εφαρμογή ιστού μέσω της οποία να μπορούμε να εκμεταλλευτούμε κάποια ευπάθεια και να επιτύχουμε reverse shell. Σε αυτό το reverse shell θα πρέπει να αναγνωρίσουμε αν βρισκόμαστε σε container ή όχι.

### 5.4.2 Αναγνώριση αν βρισκόμαστε μέσα σε container

#### *5.4.2.1 Έλεγχος για το αρχείο .dockerenv*

Το πρώτο πράγμα που μπορούμε να ελέγξουμε το οποίο μας δείχνει ότι βρισκόμαστε μέσα σε ένα Docker Container είναι αν υπάρχει το αρχείο .dockerenv [27]. Μπορούμε να χρησιμοποιήσουμε την παρακάτω εντολή για να δούμε αν υπάρχει το αρχείο αυτό.

```
$ echo `[ ! -f /.dockerenv ]` $?
```

Το .dockerenv αρχείο χρησιμοποιείται από του LXC drivers για να ορίσει με σωστό τρόπο τα environment variables. Στις τελευταίες εκδόσεις του Docker έχει αφαιρεθεί η υποστήριξη του LXC οπότε το αρχείο αυτό μπορεί να μην υπάρχει και να βρισκόμαστε μέσα σε container [27].

#### *5.4.2.2 Έλεγχος μέσω των control groups*

Μπορούμε να αναγνωρίσουμε αν μια διεργασία εκτελείται σε ένα container από το control group της διεργασίας init. Το docker δημιουργεί ένα control group που ονομάζεται docker το οποίο αποτελεί parent control group κάθε container που δημιουργείται. Μπορούμε να ελέξγουμε τα περιεχόμενα του αρχείου control group για τη διεργασία με PID 1 η οποία είναι η init διεργασία [27]. Αυτό μπορούμε να το κάνουμε με την εντολή:

```
$ cat /proc/1/cgroup
14:name=systemd:/docker/184977d6a721a33b0df3855d197b222496d4d049ed118
e3dc5e42ca3d22377b5
13:rdma:/
12:pids:/docker/721a33b0df222496d4d049…
11:hugetlb:/docker/721a33b0df222496d4d049…
10:net_prio:/docker/721a33b0df222496d4d049…
9:perf_event:/docker/721a33b0df222496d4d049…
…
```

Με την παραπάνω εντολή μπορούμε να δούμε ότι οι διεργασίες ανήκουν στο docker control group οπότε μπορούμε να συμπεράνουμε ότι βρισκόμαστε μέσα σε docker container.

#### *5.4.2.3 Έλεγχος μέσω του CPU Scheduling Info*

Ένας ακόμη τρόπος να ελέγξουμε αν βρισκόμαστε μέσα σε container είναι να δούμε το αρχείο /proc/1/sched. Αν βρισκόμαστε μέσα σε container τότε στη πρώτη γραμμή του αρχείου θα φαίνεται η εντολή της κύριας διεργασίες π.χ bash αν πρόκειται για ubuntu. Ενώ αν βρισκόμαστε στον host η πρώτη γραμμή του αρχεία θα λέει init που είναι και η διεργασία αρχικοποίησης [27]. Για να δούμε το αρχείο μπορούμε να τρέξουμε την παρακάτω εντολή:

```
$ cat /proc/1/sched | head -n 1
```

### 5.4.3 Έλεγχος του λειτουργικού συστήματος του host

Όταν βρισκόμαστε μέσα σε ένα Docker container τότε είναι σημαντικό να ξέρουμε το λειτουργικό σύστημα του host προκειμένου να έχουμε μια πιο ολοκληρωμένη εικόνα του συστήματος αλλά και για πιθανές ευπάθειες που μπορεί να μας βοηθήσουν κατά τον έλεγχο μας. Τα container μοιράζονται τον ίδιο πυρήνα με του host οπότε

μπορούμε να χρησιμοποιήσουμε την έκδοση πυρήνα για να καταλάβουμε το λειτουργικό σύστημα. Για να το κάνουμε αυτό μπορούμε να χρησιμοποιήσουμε την εντολή:

```
$ uname -rv
```

### 5.4.4 Έλεγχος του λειτουργικού συστήματος του container

Μαζί με το λειτουργικό σύστημα του host είναι εξίσου σημαντικό να προσδιορίσουμε και το λειτουργικό σύστημα του ίδιου του container. Αυτό μπορούμε να το κάνουμε διαβάζοντας το αρχείο `/etc/os-release` με την εντολή:

```
$ cat /etc/os-release
```

Φυσικά, αφού το container μοιράζεται τον ίδιο πυρήνα με το host όποιες ευπάθειες υπάρχουν στον πυρήνα τότε οι ίδιες ευπάθειες μπορούν να εφαρμοστούν και μέσα στο container.

### 5.4.5 Έλεγχος των χρηστών μέσα στο container

Είναι σημαντικό να ελέγξουμε πιθανούς χρήστες μέσα στο container. Είναι πιο ασφαλές να μην τρέχουν όλες οι διεργασίες σαν root χρήστης όποτε και η χρήση διαφόρων χρηστών για τις λειτουργίες που μπορεί να προσφέρει κάποιο container είναι αναγκαία. Ένα άπλα παράδειγμα είναι ένα container που χρησιμοποιείται ως web server για μία εφαρμογή. Συνήθως για λόγους ασφαλείας, η διεργασία του server (π.χ Apache) δεν τρέχει ως root αλλά ως κάποιος χρήστης με μικρότερα δικαιώματα. Αυτό γίνεται ώστε αν παραβιαστεί η εφαρμογή και ο επιτιθέμενος καταφέρει να λάβει reverse shell να μην έχει δικαιώματα root αλλά να έχει μόνο τα απαραίτητα δικαιώματα που χρειάζεται ο server ώστε να ελαχιστοποιήσουμε τον αντίκτυπο της επίθεσης [10]. Για να δούμε τους χρήστες μέσα σε ένα container μπορούμε να διαβάσουμε το αρχείο `/etc/passwd` το οποίο μας δείχνει όλους του χρήστες του

συστήματος καθώς και τα δικαιώματα τους, το home φάκελο τους και αν έχουν πρόσβαση σε shell. Μπορούμε να διαβάσουμε το αρχείο αυτό με την εντολή:

```
$ cat /etc/passwd
```

### 5.4.6 Έλεγχος των capabilities

Όπως είδαμε και στο προηγούμενο κεφάλαιο αναλυτικότερα το Docker προκείμενου να μπορεί να εκτελεί τις λειτουργίες χρειάζεται κάποια capabilities όμως η εκχώρηση κάποιον capabilities μπορεί να θέσει το σύστημα σε μεγάλο κίνδυνο. Έτσι, όταν πραγματοποιούμε κάποιον έλεγχο παρείσδυσης και βρισκόμαστε μέσα σε docker container είναι σημαντικό να ελέγξουμε ποια capabilities έχουν εκχωρηθεί. Για να ελέγξουμε τα capabilities μέσα από ένα container θα χρειαστεί να κάνουμε δύο ενέργειες [10]. Αρχικά, θα πρέπει να διαβάσουμε το αρχείο `/proc/self/status` και να βρούμε τη γραμμή με που αρχίζει με 'CapEff'. Αυτή η γραμμή περιέχει κωδικοποιημένα σε δεκαεξαδικό σύστημα όλα τα capabilities που έχουν εκχωρηθεί. Ουσιαστικά κάθε capability μπορεί να αναπαρασταθεί από μια δεκαεξαδική τιμή και το CapEff σε μια δεκαεξαδική τιμή τον συνδυασμό όλων των capabilities που έχουν εκχωρηθεί. Μπορούμε να διαβάσουμε αυτή την τιμή με την εντολή:

```
$ cat /proc/self/status | grep CapEff
```

Αφού αποκτήσουμε την δεκαεξαδική τιμή μπορούμε να σε ένα άλλο μηχάνημα να χρησιμοποιήσουμε το capsh προκειμένου να αποκωδικοποιήσουμε την τιμή. Αυτό μπορούμε να το κάνουμε με την εντολή:

```
$ capsh –decode=<CapEff hex_value>
```

Η εντολή αυτή θα μας δώσει σαν έξοδο όλα τα ονόματα όλων των capabilities που συμβάλουν στη δεκαεξαδική τιμή που πήραμε από το container. Με αυτό τον τρόπο μπορούμε να ελέγξουμε γρήγορα και αν ένα container τρέχει σε privileged mode πράγμα που το κάνει αυτομάτως ευάλωτο. Ουσιαστικά το privileged mode εκχωρεί

όλα τα capabilities στο container και η δεκαεξαδική αναπαράσταση στο CapEff είναι 0000003fffffffff. Συνεπώς, αν δούμε αυτή την τιμή τότε μπορούμε εύκολα να αναγνωρίσουμε ότι το container τρέχει σε privileged mode.

### 5.4.7 Έλεγχος των environment variables

Τα environment variables χρησιμοποιούνται για την παροχή πληροφοριών, όπως διαπιστευτήρια, στο container κατά την εκκίνηση του. Έτσι, κατά τη διάρκεια ελέγχου μπορούν να προσφέρουν σημαντικές πληροφορίες. Για να δούμε τα environment variables μέσα σε ένα container μπορούμε να χρησιμοποιήσουμε την εντολή:

```
$ env
```

Η εντολή αυτή θα μας δώσει όλα τα environment variables καθώς και τις τιμές αυτών.

### 5.4.8 Έλεγχος των volumes

Κατά τη διάρκεια του ελέγχου είναι σημαντικό να ελέγξουμε αν υπάρχουν volumes τα οποία έχουν γίνει mount στο container. Τα volumes όπως έχουμε δει χρησιμοποιούνται για τη διατήρηση δεδομένων του container. Ουσιαστικά ένα volume πρόκειται για ένα mounted φάκελο του host στο container. Μέσα από το container μπορούμε να δούμε όλους του mounted φακέλους διαβάζοντας το αρχείο `/proc/mounts`. Στο αρχείο αυτό σε κάθε γραμμή υπάρχει πληροφορία για κάθε mount στο container. Πιο συγκεκριμένα βλέπουμε το path των mounted φακέλων μέσα στο container όμως δεν μπορούμε να δούμε το αντίστοιχο path του host. Η εντολή για να διαβάσουμε το αρχείο `/proc/mounts` είναι η παρακάτω:

```
$ cat /proc/mounts
```

Με αυτό τον τρόπο επιπλέον μπορούμε να ελέγξουμε αν το docker socket είναι mounted στο container πράγμα που μπορεί να θέσει το σύστημα σε κίνδυνο. Για να το

κάνουμε αυτό αρκεί να ψάξουμε στο φάκελο `/proc/mounts` για κάποια εγγραφή που να περιέχει το λεκτικό `docker.sock` ή μπορούμε να ψάξουμε σε όλα τα αρχεία του συστήματος για το αρχείο `docker.sock` με την εντολή:

```
$ find / -name ‘docker.sock’
```

### 5.4.9 Έλεγχος της διάταξης δικτύου

Η δικτύωση του συστήματος που τρέχει Docker containers είναι πολύ σημαντική κατά τη διάρκεια ελέγχων. Ένα σύστημα μπορεί να υλοποιεί πολλαπλά containers τα οποία πρέπει να επικοινωνούν μεταξύ του για τη επίτευξη των λειτουργιών του συστήματος. Έτσι, είναι σημαντικό να δούμε μέσα από ένα container αν μπορούμε να επικοινωνήσουμε με άλλα containers και αν είναι εφικτό να μεταπηδήσουμε μεταξύ αυτών. Είναι σημαντικό να ξέρουμε ότι από προεπιλογή όλα τα containers δέχονται μια IPv4 διεύθυνση που ανήκει στο υποδίκτυο 172.17.0.0/16.

Αρχικά, μπορούμε να διαβάσουμε το αρχείο `/etc/hosts/` για να δούμε αν υπάρχουν εγγραφές για τυχόν host που μπορεί να επικοινωνήσει το container. Αυτό μπορούμε να το κάνουμε με την εντολή:

```
$ cat /etc/hosts
```

Έπειτα, για περαιτέρω έλεγχο θα μπορούσαμε να εγκαταστήσουμε στο container εργαλεία όπως το nmap για να πραγματοποιήσουμε εκτενέστερους δικτυακούς ελέγχους.

# 6. Εργαλεία για δοκιμές παρείσδυσης σε Docker

## 6.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα δούμε και θα αναλύσουμε μερικά εργαλεία τα οποία μπορούν να μας βοηθήσουν στο να πραγματοποιήσουμε ελέγχους που χρειάζονται όπως είδαμε στο προηγούμενο κεφάλαιο.

## 6.2 CDK - Zero Dependency Container Penetration Toolkit

Το Zero Dependency Container Penetration Toolkit ή αλλιώς CDK [28] πρόκειται για ένα εργαλείο το οποίο μας βοηθάει στην πραγματοποίηση ελέγχων μέσα σε ένα Docker container. Ουσιαστικά με το CDK μπορούμε να ανιχνεύσουμε ευπάθειες που μπορεί να υπάρχουν μέσα σε ένα container αλλά μας προσφέρει τη δυνατότητα να εκμεταλλευτούμε αυτόματα κάποια ευπάθεια την οποία έχει αναγνωρίσει.

Για να τρέξουμε το CDK πρέπει να κατεβάσουμε το εκτελέσιμο για την αρχιτεκτονική που αντιστοιχεί στο σύστημα στο οποίο πραγματοποιούμε έλεγχο. Αυτό μπορούμε να το κάνουμε από το Github του εργαλείου όπου έχει έτοιμα τα εκτελέσιμα αρχεία [https://github.com/cdk-team/CDK/releases/tag/v1.5.1].

Το CDK έχει τρείς κύριες λειτουργίες:

- Αξιολόγηση του συστήματος για εύρεση πληροφοριών στο εσωτερικό του container για την ανεύρεση πιθανών ευπαθειών.
- Εκμετάλλευση ευπαθειών όπως container escapes.
- Χρήσιμα δικτυακά εργαλεία.

### 6.2.1 Λειτουργεία Αξιολόγησης (Evaluate Module)

Το CDK παρέχει κάποια scripts για τη συλλογή πληροφοριών και την αξιολόγηση του container όσο αφορά τις ευπάθειες. Ενδεικτικά με τη λειτουργία evaluate μπορούμε να λάβουμε πληροφορίες για το λειτουργικό σύστημα, τα capabilities, τα mounts κ.α. Για την πλήρη λίστα των scripts μπορείτε να ανατρέξετε στο wiki του εργαλείου [https://github.com/cdk-team/CDK/wiki#evaluate-module]. Για να τρέξουμε το CDK σε λειτουργία evaluate μπορούμε να χρησιμοποιήσουμε την παρακάτω εντολή:

```
$ cdk evaluate –full
```

Η εντολή αυτή θα μας δώσει σαν έξοδο όσες πληροφορίες μάζεψε το CDK από το container καθώς και ευπάθειες που μπορεί να υπάρχουν και πως μπορούμε να τις εκμεταλλευτούμε.

### 6.2.2 Λειτουργία Εκμετάλλευσης Ευπαθειών (Exploit Module)

Πέρα από την αναγνώριση ευπαθειών το CDK προσφέρει και μια πληθώρα από scripts για την εκμετάλλευση των ευπαθειών αυτών. Ενδεικτικά προσφέρει δυνατότητα εκμετάλλευσης για τις πιο γνωστές ευπάθειες του Docker όπως την ευπάθεια Docker runC με CVE-2019-5736, cgroup Docker Escape, ευπάθειες για Kubernetes κ.α. Για την πλήρη λίστα των scripts με τις ευπάθειες που μπορεί να εκμεταλλευτεί το εργαλείο μπορείτε να ανατρέξετε στο wiki του εργαλείου [https://github.com/cdk-team/CDK/wiki#exploit-module]. Επίσης μπορούμε να τυπώσουμε τη λίστα με τα διαθέσιμα scripts με την εντολή:

```
$ cdk run –list
```

Για να τρέξουμε ένα συγκεκριμένο script μπορούμε να το κάνουμε με την παρακάτω εντολή:

```
$ cdk run <script-name> [options]
```

### 6.2.3 Λειτουργεία Εργαλείων (Tool Module)

Σε αυτή τη λειτουργία το CDK παρέχει κάποια χρήσιμα εργαλεία, κυρίως δικτυακά, τα οποία μπορούν να βοηθήσουν στο έλεγχο. Ενδεικτικά μερικά από αυτά είναι τα nc, ifconfig, vi , ps κ.α. Για την πλήρη λίστα των εργαλείων που παρέχει το CDK μπορείτε να ανατρέξετε στο wiki του εργαλείου [https://github.com/cdk-team/CDK/wiki#tool-module]. Για να τρέξουμε κάποιο εργαλείο μπορούμε να το κάνουμε με την παρακάτω εντολή:

```
$ cdk <tool_name> [options]
```

## 6.3 .Anchore - Grype

Το Grype [29] πρόκειται για ένα εργαλείο ανίχνευσης ευπαθειών σε Docker Images και συστήματα αρχείων. Μπορούμε να εγκαταστήσουμε το Grype με το script που παρέχεται στο Github [https://github.com/anchore/grype#installation]. Αφού εγκαταστήσουμε το εργαλείο μπορούμε να το τρέξουμε με την εντολή:

```
$ grype <image>
```

Επίσης, μπορούμε να τρέξουμε το Grype σε Docker container προκειμένου να μπορεί να σκανάρει containers που ήδη τρέχουν [30]. Αυτό μπορούμε να το κάνουμε με την παρακάτω εντολή:

```
$ docker run --rm \
--volume /var/run/docker.sock:/var/run/docker.sock \
--name Grype anchore/grype:latest \
$(ImageName):$(ImageTag)
```

Το Grype υποστηρίζει διάφορων ειδών πηγές για σκανάρισμα. Μπορεί να δεχθεί tar Docker Images που προκύπτουν από την εντολή `docker image save …`, Singularity Image Format (SIF) containers ή κάποιο κατάλογο αρχείων.

Το εργαλείο επίσης είναι αρκετά παραμετροποιήσιμο αφού προσφέρει διάφορους τύπου εξόδου όπως πχ json, cyclonedx, ή κάποιο δικό μας template γραμμένο σε Go. Επίσης, μπορούμε να ορίσουμε το εργαλείο ώστε να αγνοεί κάποια CVEs σε περίπτωση που για παράδειγμα υπάρχουν false positives. Αυτό μπορούμε να το κάνουμε ορίζοντας τις παραμέτρους μας στο αρχείο `~/.grype.yml`.

## 6.4 Docker Bench for Security

Το Docker Bench for Security [31] σαρώνει τον host που τρέχει Docker για κοινά προβλήματα διαμόρφωσης, όπως ρυθμίσεις σε αρχεία διαμόρφωσης, δικαιώματα συστήματος και αμφισβητήσιμες προεπιλογές. Το εργαλείο βασίζεται σε μια βάση δεδομένων με κοινές ευπάθειες και εκθέσεις (CVE) για τον έλεγχο των βιβλιοθηκών και των εκτελέσιμων αρχείων στο εν λόγω σύστημα.

Στο τέλος κάθε σάρωσης, παρέχει μια βαθμολογία. Οι διαχειριστές μπορούν να παρακολουθούν τη βαθμολογία Docker Bench for Security στο host για να επισημάνουν τις βελτιώσεις με την πάροδο του χρόνου.

Οι έλεγχοι που πραγματοποιεί το Docker Bench for Security βασίζονται στο CIS Docker Benchmark [32] το οποίο παρέχει κατευθυντήριες γραμμές για την ενίσχυση ασφάλειας του Docker σε ένα σύστημα. Το Docker Bench for Security ουσιαστικά πρόκειται για ένα εργαλείο συμμόρφωσης με τις κατευθυντήριες γραμμές και όχι τόσο ένα εργαλείο που μας βοηθάει στις δοκιμές παρείσδυσης. Πάραυτα, είναι ένα σημαντικό εργαλείο για την ενίσχυση της ασφάλειας ενός συστήματος Docker.

## 6.5 Dockscan

Το Dockscan [33] έχει σχεδιαστεί και αναπτυχθεί ειδικά για τη σάρωση τρωτών σημείων ασφαλείας και τον έλεγχο εγκαταστάσεων Docker. Η χρήση του Dockscan είναι πολύ καθώς επικοινωνεί με το docker daemon είτε μέσω unix socket είτε με TCP πρωτόκολλο. Αυτό μας δίνει τη δυνατότητα να τρέξουμε το εργαλεία τοπικά σε ένα σύστημα ή απομακρυσμένα εφόσον είναι ενεργοποιημένο το Docker Daemon να

επικοινωνεί μέσω TCP [34]. Μπορούμε να τρέξουμε το Dockscan τοπικά με την παρακάτω εντολή:

```
$ dockscan unix:///var/run/docker.sock
```

Επιπλέον, μπορούμε να τρέξουμε το Dockscan για απομακρυσμένο σύστημα με την εντολή:

```
$ dockscan tcp://<domain_to_test>:<port_of_docker_daemon>
```

Το Dockscan αναγνωρίζει αυτόματα τα containers που τρέχουν, πραγματοποιεί τους ελέγχους που χρειάζονται και στο τέλος μας παρουσιάζει τα ευρήματα και τις ευπάθειες.

## 6.6 .Docker Enumeration, Escalation of Privileges and Container Escapes (DEEPCE)

Το DEEPCE [35] πρόκειται για ένα script το οποίο πραγματοποιεί ελέγχους σε host που υλοποιεί σύστημα Docker. Το εργαλείο αυτό έχει δύο λειτουργίες τη λειτουργεία της αναγνώρισης και τη λειτουργεία της εκμετάλλευσης.

Στη λειτουργεία της αναγνώρισης μπορεί να αναγνωρίσει:

- Container IDs και ονόματα μέσω reverse DNS
- Container IP
- Docker Version
- Mounts
- Environment Variables
- Κωδικούς σε αρχεία
- Εκτεθημένο Docker Socket
- Container που επικοινωνούν δικτυακά

Στη λειτουργεία της εκμετάλλευσης το εργαλείο μπορεί να εκμεταλλευτεί:

- Docker Group Privilege Escalation
- Privileged mode host command execution
- Εκτεθημένο Docker Socket

Για να τρέξουμε το εργαλείο αρκεί να το κατεβάσουμε από το Github να κάνουμε εκτελέσιμο το script και να το τρέξουμε όπως παρακάτω:

```
$ chmod +x ./deepce.sh
$ ./deepce.sh
```

Αφού πραγματοποιήσουμε τη λειτουργεία της αναγνώρισης και το εργαλείο βρει κάποια ευπάθεια την οποία μπορεί να εκμεταλλευτεί τότε μπορούμε να χρησιμοποιήσουμε την παρακάτω εντολή για να τρέξουμε το εργαλείο σε λειτουργεία εκμετάλλευσης χωρίς να τρέξει πάλι σε λειτουργεία αναγνώρισης.

```
$ ./deepce.sh --no-enumeration --exploit <exploit_name> [--options]
```

## 6.7 Metasploit

Το Metasploit [36] είναι ένα framework το οποίο παρέχει εργαλεία τα οποία βοηθάνε στην εκμετάλλευση ευπαθειών. Στο Metasploit μπορούμε να βρούμε χρήσιμα εργαλεία που ειδικεύονται στην ανίχνευση και την εκμετάλλευση ευπαθειών σε περιβάλλοντα Docker. Για παράδειγμα το module Linux Gather Container Detection ανιχνεύει αν βρισκόμαστε μέσα σε Docker Container.

## 6.8 Break Out the Box (BOtB)

Το Break Out the Box [37] είναι ένα εργαλείο το οποίο εξειδικεύεται στην ανίχνευση και την εκμετάλλευση ευπαθειών σε Docker containers. Με το BOtB μπορούμε να πραγματοποιήσουμε κυρίως Docker escapes όπως για παράδειγμα το CVE-2019-5736. Το εργαλείο αυτό μπορεί να πραγματοποιήσει και άλλες ενέργειες όπως την ανίχνευση Docker Socket που απαντάνε σε HTTP, την ανίχνευση και εκμετάλλευση Docker container που έχουν εκκινηθεί σε privileged mode κ.α. Για την πλήρη λίστα μπορείτε να ανατρέξετε στο Github του εργαλείου στο παρακάτω σύνδεσμο [https://github.com/brompwnie/botb#current-capabilities].

## 6.9 Dagda

Το Dagda [38] είναι ένα εργαλείο για την εκτέλεση στατικής ανάλυσης γνωστών ευπαθειών, trojans, ιών, κακόβουλου λογισμικού και άλλων κακόβουλων απειλών σε εικόνες/containers docker και για την παρακολούθηση του docker daemon και των εκτελούμενων docker containers για την ανίχνευση ανώμαλων δραστηριοτήτων.

Για να εκπληρώσει την αποστολή του, εισάγονται σε μια MongoDB πρώτα τα γνωστά CVEs (Common Vulnerabilities and Exposures), BIDs (Bugtraq IDs), RHSAs (Red Hat Security Advisories) και RHBAs (Red Hat Bug Advisories), καθώς και τα γνωστά exploits από τη βάση δεδομένων Offensive Security για να διευκολυνθεί η αναζήτηση των τρωτών σημείων και των exploits όταν η ανάλυσή βρίσκεται σε εξέλιξη.

Στη συνέχεια, όταν εκτελείτε μια στατική ανάλυση γνωστών ευπαθειών, το Dagda ανακτά πληροφορίες σχετικά με το λογισμικό που είναι εγκατεστημένο στο Docker Image, όπως τα πακέτα του λειτουργικού συστήματος και τις εξαρτήσεις των γλωσσών προγραμματισμού, και επαληθεύει για κάθε προϊόν και την έκδοσή του αν είναι απαλλαγμένο από ευπάθειες σε σχέση με τις πληροφορίες που έχουν αποθηκευτεί προηγουμένως στη MongoDB. Επίσης, το Dagda χρησιμοποιεί το ClamAV ως μηχανή προστασίας από ιούς για τον εντοπισμό trojans, ιών,

κακόβουλου λογισμικού & άλλων κακόβουλων απειλών που περιλαμβάνονται στα Docker Images ή Docker Containers.

# 7. Λίστα ελέγχου για δοκιμές παρείσδυσης σε περιβάλλοντα docker container

## 7.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα συνοψίσουμε σε μια λίστα ελέγχου σε μορφή ερωτήσεων όλους τους ελέγχους και τα βήματα που πρέπει να κάνουμε σε μια δοκιμή παρείσδυσης σε περιβάλλοντα Docker Container. Οι έλεγχοι αυτοί είναι οι έλεγχοι και τα βήματα που αναλύσαμε στο Κεφάλαιο 5. Όπως και στο Κεφάλαιο 5 έτσι και εδώ θα χωρίσουμε τη λίστα ελέγχου σε δύο μέρη ανάλογα με το αν βρισκόμαστε μέσα σε container ή στο host.

## 7.2 Λίστα ελέγχου για δοκιμές στον host

Παρακάτω θα συγκεντρώσουμε σε ένα πίνακα τα βήματα ελέγχου για τις δοκιμές παρείσδυσης στο host ενός συστήματος που υλοποιεί Docker container. Οι παρακάτω ερωτήσεις προκύπτουν από το Κεφάλαιο 5 στο οποίο και αναλύονται περαιτέρω.

| **Έλεγχος** | **Απάντηση** |
|---|---|
| **Ποια είναι η έκδοση του docker στο host;**<br>($ docker -v) | |
| **Ποιοι χρήστες ανήκουν στο docker group;**<br>($ cat /etc/group \| grep docker) | |
| **Ποιος μπορεί να χρησιμοποιήσει το Docker socket;**<br>($ ls -l $(find / -name docker.sock 2>/dev/null)) | |
| **Ποια Docker Images υπάρχουν στο host;**<br>($ docker images -a) | |

| | |
|---|---|
| **Υπάρχουν αρχεία διαμόρφωσης που μπορούμε να διαβάσουμε όπως docker-compose.yml, .docker/config.json, Dockerfile, daemon.json;** | |
| **Ποια containers υπάρχουν στο host;**<br>($ docker ps -a) | |
| **Ποια containers τρέχουν στο host;**<br>($ docker ps) | |
| **Έχει το docker binary ενεργοποιημένο το setuid bit;**<br>($ ls -l $(which docker)) | |

*Πίνακας 5 Λίστα ελέγχου για δοκιμές στον host*

## 7.3 Λίστα ελέγχου για δοκιμές μέσα σε Docker container

Παρακάτω θα συγκεντρώσουμε σε ένα πίνακα τα βήματα ελέγχου για τις δοκιμές παρείσδυσης μέσα σε ένα Docker container. Οι παρακάτω ερωτήσεις προκύπτουν από το Κεφάλαιο 5 στο οποίο και αναλύονται περαιτέρω.

| **Έλεγχος** | **Απάντηση** |
|---|---|
| **Βρισκόμαστε μέσα σε Docker container;**<br>Υπάρχει το αρχείο .dockerenv; ($ echo `[ ! -f /.dockerenv ]` $?)<br>Υπάρχει εγγραφή /docker/ στο αρχείο /proc/1/cgroup; ($ cat /proc/1/cgroup)<br>Υπάρχουν λιγότερες από 5 διεργασίες; ($ ps -aux)<br>Υπάρχει στο αρχείο /proc/1/sched στην πρώτη γραμμή άλλη εγγραφή εκτός από init ή systemd; ($ cat /proc/1/sched \| head -n 1) | |
| **Ποιο πυρήνα έχει ο host;**<br>($ uname -rv) | |
| **Ποιο είναι το λειτουργικό σύστημα του container;**<br>($ cat /etc/os-release) | |
| **Ποιοι χρήστες υπάρχουν στο container;**<br>($ cat /etc/passwd) | |
| **Ποια capabilities έχουν εκχωρηθεί στο container;**<br>($ cat /proc/self/status \| grep CapEff)<br>($ capsh –decode=<CapEff hex_value>) | |
| **Έχει εκκινηθεί το container σε privileged mode;**<br>($ cat /proc/self/status \| grep CapEff)<br>Αν CapEff ==0000003fffffffff τότε container είναι σε privileged mode. | |
| **Ποια enviroment variables υπάρχουν στο container;**<br>($ env) | |
| **Υπάρχουν mounted volumes στο container;**<br>($ cat /proc/mounts) | |
| **Ποιες διεργασίες τρέχουν στο container;** | |

| | |
|---|---|
| ($ ps aux) | |
| **Είναι το Docker socket mounted στο container;**<br>($ find / -name ‘docker.sock’) | |
| **Μπορεί το container να δει δικτυακά άλλα containers;**<br>($ cat /etc/hosts)<br>Nmap στο τοπικό δίκτυο. | |

*Πίνακας 6 Λίστα ελέγχου για δοκιμές μέσα σε Docker container*

# 8. Συμπεράσματα

Το Docker όπως είδαμε, είναι μια τεχνολογία που διευκολύνει πολύ τη διαδικασία ανάπτυξης λογισμικού και χρησιμοποιείται ευρέως στη αγορά. Όπως είδαμε όμως κρύβει πολλούς κινδύνους και μπορεί να θέσει σε κίνδυνο τα συστήματα που το υλοποιούν. Είναι σημαντικό λοιπόν να ακολουθούμε τις κατευθυντήριες γραμμές ασφαλείας όταν υλοποιούμε ένα σύστημα Docker για να μειώσουμε στο ελάχιστο τους κινδύνους. Όπως είδαμε όμως είναι πολύ σημαντικό να πραγματοποιούμε και δοκιμές παρείσδυσης ανά τακτά χρονικά διαστήματα ή μετά από αλλαγές στο σύστημα για να μπορούμε να διατηρούμε την ασφάλεια και όσο το δυνατόν υψηλότερα επίπεδα.

Οι κίνδυνοι σε ένα σύστημα Docker container μπορούν να προκύψουν από πολλές πλευρές καθώς έχουν τους κινδύνους που μπορούν να προκύψουν από το host, τα Docker Images, το λογισμικό αλλά και το ίδιο το Docker. Έτσι, είναι σημαντικό να χρησιμοποιούμε τις λίστες ελέγχου που είδαμε στο Κεφάλαιο 7 προκειμένου να μην μας διαφύγει κάποιος έλεγχος.

# Αναφορές


[1] «Linux History,» Javapoint, [Ηλεκτρονικό]. Available: https://www.javatpoint.com/linux-history.

[2] «Linux,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Linux.

[3] «What is virtualization,» Redhat, 2 3 2018. [Ηλεκτρονικό]. Available: https://www.redhat.com/en/topics/virtualization/what-is-virtualization.

[4] «What are containers,» NetApp, [Ηλεκτρονικό]. Available: https://www.netapp.com/devops-solutions/what-are-containers/.

[5] «What is penetration Testing,» Coresecurity, [Ηλεκτρονικό]. Available: https://www.coresecurity.com/penetration-testing#what-is-pen-testing.

[6] «Docker,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Docker_(software).

[7] «Docker Builder,» Docker, [Ηλεκτρονικό]. Available: https://docs.docker.com/engine/reference/builder/.

[8] «Networking Overview,» Docker, [Ηλεκτρονικό]. Available: https://docs.docker.com/network/.

[9] «Docker Registry,» aquasec, [Ηλεκτρονικό]. Available: https://www.aquasec.com/cloud-native-academy/docker-container/docker-registry/.

[10] J. Vrancken, «A Methodology for Penetration,» 2020.

[11] D. J. P. M. a. R. B. A. Tomar, «Docker Security: A Threat Model, Attack, Taxonomy and Real-Time Attack Scenario of DoS,» σε *10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2020.

[12] B. Dickson, «Hot patch for Log4Shell vulnerability in AWS allowed full host takeover,» PortSwigger, 25 April 2022. [Ηλεκτρονικό]. Available: https://portswigger.net/daily-swig/vulnerability-aws-log4shell-hot-patch-allowed-full-host-takeover.

[13] «CVE-2019-5736,» Mitre, [Ηλεκτρονικό]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5736.

[14] M. Hill, «Pro-Ukrainian DoS attack compromises Docker Engine honeypots to target Russian, Belarusian websites,» CSO USA, 4 May 2022. [Ηλεκτρονικό]. Available: http://csoonline.com/article/3658917/pro-ukrainian-dos-attack-compromises-docker-engine-honeypots-to-target-russian-belarusian-websites.html.

[15] «Malware Attacks: Definition and Best Practices,» Rapid7, [Ηλεκτρονικό]. Available: https://www.rapid7.com/fundamentals/malware-attacks/.

[16] «What is a denial of service attack (DoS) ?,» Palo Alto Networks, [Ηλεκτρονικό]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos.

[17] J. Chelladhurai, P. R. Chelliah και S. A. Kumar, «Securing docker containers from denial of service (dos) attacks,» *2016 IEEE International Conference on Services Computing (SCC),* 2016.

[18] N. Stoler, «How I Hacked Play-with-Docker and Remotely Ran Code on the Host,» Cyberark, 14 January 2019. [Ηλεκτρονικό]. Available: https://www.cyberark.com/resources/threat-research-blog/how-i-hacked-play-with-docker-and-remotely-ran-code-on-the-host.

[19] «CVE-2018-8115,» Mitre, [Ηλεκτρονικό]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8115.

[20] «Docker CE: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') (CVE-2018-15664),» Rapid7 , [Ηλεκτρονικό]. Available: https://www.rapid7.com/db/vulnerabilities/docker_ce-

cve-2018-15664/.

[21] «CVE-2018-15664,» Openwall, [Ηλεκτρονικό]. Available: https://www.openwall.com/lists/oss-security/2019/05/28/1.

[22] «CVE-2019-5736 Detail,» NIST, [Ηλεκτρονικό]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-5736.

[23] «Breaking out of Docker via runC – Explaining CVE-2019-5736,» Palo alto Networks, [Ηλεκτρονικό]. Available: https://unit42.paloaltonetworks.com/breaking-docker-via-runc-explaining-cve-2019-5736/.

[24] «CVE-2019-5021 Detail,» NIST, [Ηλεκτρονικό]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-5021.

[25] «CVE-2020-13401 Detail,» NIST, [Ηλεκτρονικό]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-13401.

[26] «Grype,» Github, [Ηλεκτρονικό]. Available: https://github.com/anchore/grype.

[27] baeldung, «Determine if a Process Runs Inside a Container,» baeldung, [Ηλεκτρονικό]. Available: https://www.baeldung.com/linux/is-process-running-inside-container.

[28] «CDK,» Github, [Ηλεκτρονικό]. Available: https://github.com/cdk-team/CDK.

[29] «Anchore/Grype,» Github, [Ηλεκτρονικό]. Available: https://github.com/anchore/grype.

[30] bordergate, «Docker Penetration Testing,» Bordergate, [Ηλεκτρονικό]. Available: https://www.bordergate.co.uk/docker-penetration-testing/.

[31] «docker-bench-security,» Github, [Ηλεκτρονικό]. Available: https://github.com/docker/docker-bench-security.

[32] «CIS Docker Benchmarks,» Center for Internet Security, [Ηλεκτρονικό]. Available: https://www.cisecurity.org/benchmark/docker.

[33] «Dockscan,» Github, [Ηλεκτρονικό]. Available: https://github.com/kost/dockscan.

[34] «Identify the issues in Docker setup using Dockscan,» tbhaxor, [Ηλεκτρονικό]. Available: https://tbhaxor.com/identify-issues-in-docker-setup-using-dockscan/.

[35] «deepce,» Github, [Ηλεκτρονικό]. Available: https://github.com/stealthcopter/deepce.

[36] «metasploit,» Rapid7, [Ηλεκτρονικό]. Available: https://www.metasploit.com/.

[37] «botb,» Github, [Ηλεκτρονικό]. Available: https://github.com/brompwnie/botb.

[38] «dagda,» Github, [Ηλεκτρονικό]. Available: https://github.com/eliasgranderubio/dagda.