https://doi.org/10.26219/heal.aueb.9571

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΚΩΝ ΣΠΟΥΔΩΝ ΣΤΟ ΤΜΗΜΑ ΑΝΑΠΤΥΞΗΣ

ΚΑΙ ΑΣΦΑΛΕΙΑΣ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

TRANSPARENCY SERVICES FOR SECURING KUBERNETES IMAGE REGISTRIES

ΝΤΕΝΕΖΟΣ ΤΑΞΙΑΡΧΗΣ - ΜΙΧΑΗΛ (p3312313)

Αθήνα, Οκτώβριος 2025

https://doi.org/10.26219/heal.aueb.9571

https://doi.org/10.26219/heal.aueb.9571

MASTER IN DEVELOPMENT AND SECURITY OF INFORMATION SYSTEMS

MASTER THESIS IN TRANSPARENCY SERVICES FOR SECURING KUBERNETES IMAGE REGISTRIES

NTENEZOS TAXIARCHIS - MICHAIL (p3312313)

Athens, October 2025

https://doi.org/10.26219/heal.aueb.9571

Ευχαριστίες

Φτάνοντας στο τέλος των μεταπτυχιακών μου σπουδών, θα ήθελα να ευχαριστήσω τον καθηγητή μου Γεώργιο Ξυλωμένο και τον επιβλέποντα Δρ. Νίκο Φωτίου για την πολύτιμη καθοδήγηση και τις ουσιαστικές συμβουλές που μου παρείχαν καθ' όλη τη διάρκεια της εκπόνησης της διπλωματικής μου εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την υποστήριξη καθ' όλη την διάρκεια των σπουδών μου.

https://doi.org/10.26219/heal.aueb.9571

https://doi.org/10.26219/heal.aueb.9571

TRANSPARENCY SERVICES FOR SECURING KUBERNETES IMAGE REGISTRIES

Περίληψη

Η συνεχώς αυξανόμενη ανάπτυξη των τεχνολογιών λογισμικού και η εξάρτηση τους από βιβλιοθήκες, πακέτα και υποδομές τρίτων έχει οδηγήσει σε σημαντικούς κινδύνους στην αλυσίδα εφοδιασμού λογισμικού. Η ασφάλεια δεν εξαρτάται πλέον μόνο από την σωστή ανάπτυξη κώδικα αλλά καλύπτει ολόκληρη τη διαδικασία, από τη δημιουργία και την διάθεση μιας εφαρμογής έως και τον τρόπο εκτέλεσης της. Η εμφάνιση επιθέσεων που εκμεταλλεύονται την εμπιστοσύνη που έχουμε σε τρίτα μέρη, όπως είναι οι πρόσφατες επιθέσεις στο Npm καθώς επίσης και το περιστατικό του SolarWinds, είναι η απόδειξη ότι η παραδοσιακή λογική της εμπιστοσύνης χωρίς επαλήθευση δεν επαρκεί.

Η παρούσα διπλωματική εργασία εστιάζει στην ενσωμάτωση του οικοσυστήματος Sigstore, που είναι ένα έργο ανοιχτού κώδικα, σε περιβάλλον Kubernetes, με σκοπό να ενισχυθεί η ασφάλεια και η διαφάνεια στην διανομή των container images. Το Sigstore, αξιοποιεί τις υπηρεσίες Fulcio, Rekor και cosign ώστε να γίνεται δυνατή η δημιουργία και επαλήθευση ψηφιακών υπογραφών χωρίς να χρειάζεται η χρήση μόνιμων κλειδιών. Μέσω των παραπάνω υπηρεσιών, οι υπογραφές των artifacts συνδέονται με επαληθεύσιμες ταυτότητες μέσω OIDC και καταγράφονται σε δημόσια, αμετάβλητα transparency log.

Στο πρακτικό μέρος γίνεται διερεύνηση της πρακτικής ενσωμάτωσης του sigstore σε ένα ασφαλές περιβάλλον Kubernetes, με χρήση private registry για την αποθήκευση των container images και αξιοποίηση policy controllers για την επιβολή κανόνων επαλήθευσης υπογραφών. Επιπλέον, αναπτύσσεται ένα εργαλείο watcher, το οποίο εντοπίζει αυτόματα μη επαληθευμένα images και τα αφαιρεί, συμβάλλοντας στη διατήρηση ενός καθαρού και ασφαλούς registry.

https://doi.org/10.26219/heal.aueb.9571

https://doi.org/10.26219/heal.aueb.9571

TRANSPARENCY SERVICES FOR SECURING KUBERNETES IMAGE REGISTRIES

Abstract

The continuously growing development of software technologies and their dependency on third-party libraries, packages, and infrastructures have introduced significant risks within the software supply chain. Security no longer depends solely on proper code development but now encompasses the entire process from the creation and distribution of an application to the way it is executed. The emergence of attacks that exploit the inherent trust in third parties, such as the recent npm incidents and the SolarWinds compromise, demonstrates that the traditional model of trust without verification is no longer sufficient.

This thesis focuses on the integration of the Sigstore ecosystem, an open-source project into a Kubernetes environment, aiming to enhance security and transparency in the distribution of container images. Sigstore leverages the Fulcio, Rekor and Cosign services to enable the creation and verification of digital signatures without the need for permanent key management. Through these services, artifacts signatures are associated with verifiable identities via OpenID Connect (OIDC) and are recorded in public, immutable transparency logs.

In the practical part, the thesis explores the integration of Sigstore within a secure Kubernetes environment using a private registry for storing container images and employing policy controllers to enforce signature verification rules. Additionally, a watcher tool is developed, which automatically detects unverified images and removes them, contributing to the maintenance of a clean and secure registry.

x

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Ευχαριστίες v

Περίληψη vii

Abstract ix

1. Εισαγωγικές Έννοιες 1

1.1 Ταυτοποίηση (Authentication) και Εξουσιοδότηση (Authorization) 2

1.2 OpenID Connect (OIDC) και Ταυτοποίηση στο Sigstore 2

1.3 Πιστοποιητικά και Αρχές Πιστοποίησης 2

1.4 Transparency Logs και Merkle Trees 3

1.5 Containers και Private registries 3

1.6 Policy controller και Επαλήθευση υπογραφών στο Kubernetes 3

2. Sigstore 4

2.1 Εισαγωγή 4

2.2 Περιγραφή και Βασικές Αρχές του Sigstore 4

2.3 Προκλήσεις και Προβλήματα που Επιλύει 4

2.4 Αρχιτεκτονική του Sigstore 5

3. Supply Chain Attacks 8

4. Πρακτικό Μέρος 12

4.1 Δημιουργία και ρύθμιση ενός private registry 12

4.2 Ενσωμάτωση private registry στο Kubernetes και εφαρμογή πολιτικών ασφαλείας 15

4.3 Υπογραφή και επαλήθευση container images μέσω sigstore 19

4.4 Υλοποίηση watcher για αυτόματο έλεγχο / διαγραφή Images 23

Επίλογος 31

Βιβλιογραφία 33

https://doi.org/10.26219/heal.aueb.9571

 https://doi.org/10.26219/heal.aueb.9571

1. Εισαγωγικές Έννοιες

Ο σκοπός της παρούσας διπλωματικής αποτελείται από δύο σκέλη. Στο θεωρητικό κομμάτι επιδιώκεται η αναλυτική παρουσίαση του οικοσυστήματος Sigstore, των υπηρεσιών που το αποτελούν και της λειτουργίας της υπογραφής λογισμικών artifacts χωρίς μόνιμη διαχείριση κρυπτογραφικών κλειδιών. Στην συνέχεια, αναδεικνύεται ο ρόλος των Transparency logs ως μηχανισμού που ενισχύει την εμπιστοσύνη και την ακεραιότητα στην εφοδιαστική αλυσίδα λογισμικού, καθώς παρέχει επαληθεύσιμη και αμετάβλητη καταγραφή των συμβάντων υπογραφής.

Στο πρακτικό κομμάτι, η εργασία επικεντρώνεται στην υλοποίηση ενός ολοκληρωμένου συστήματος ασφαλείας σε εργαστηριακό περιβάλλον. Η υλοποίηση περιλαμβάνει τη δημιουργία ενός ασφαλούς private registry με υποστήριξη TLS, την ενσωμάτωση του σε περιβάλλον Kubernetes και την εφαρμογή πολιτικών που επιβάλλουν την επαλήθευση ψηφιακών υπογραφών πριν την εκτέλεση ενός container image. Επιπλέον, παρουσιάζεται ο τρόπος με τον οποίο το cosign χρησιμοποιείται για την υπογραφή και επαλήθευση των images, καθώς και η ανάπτυξη ενός watcher script που ανιχνεύει και αφαιρεί αυτόματα μη υπογεγραμμένες εικόνες, διατηρώντας το registry καθαρό.

Για να γίνει κατανοητό το αντικείμενο της εργασίας, η εργασία έχει οργανωθεί έτσι ώστε να ξεκινά από τις βασικές έννοιες και να καταλήγει στην πρακτική υλοποίηση. Στο πρώτο κεφάλαιο παρουσιάζονται οι εισαγωγικές έννοιες που αποτελούν το υπόβαθρο για την καλύτερη κατανόηση της εργασίας, όπως το OIDC, τα ψηφιακά πιστοποιητικά, τα transparency logs, η λειτουργία των containers και των private registries, καθώς και οι μηχανισμοί πολιτικών στο Kubernetes. Στη συνέχεια, το δεύτερο κεφάλαιο εστιάζει στο Sigstore, εξηγώντας αναλυτικά πως λειτουργούν τα Fulcio, Rekor και Cosign και πως συνεργάζονται για να προσφέρουν ένα ασφαλές περιβάλλον υπογραφής και επαλήθευσης.

Το τρίτο κεφάλαιο εξετάζει τις επιθέσεις στην εφοδιαστική αλυσίδα λογισμικού, παρουσιάζοντας τις τεχνικές που χρησιμοποιούν οι επιτιθέμενοι. Επιπλέον γίνεται παρουσίαση μιας πραγματικής επίθεσης στην εφοδιαστική αλυσίδα, όπου περιγράφονται τα στάδια κλιμάκωσης, καθώς και οι επιπτώσεις που είχε το συμβάν. Τέλος, το τέταρτο κεφάλαιο είναι το πρακτικό σκέλος της εργασίας και περιγράφει βήμα προς βήμα την υλοποίηση από τη δημιουργία του private registry, μέχρι την ενσωμάτωση των πολιτικών και την λειτουργία του watcher.

1.1 Ταυτοποίηση (Authentication) και Εξουσιοδότηση (Authorization)

Η ταυτοποίηση (authentication) είναι η διαδικασία με την οποία ένας χρήστης ή μια υπηρεσία αποδεικνύει την ταυτότητα του σε ένα σύστημα. Σκοπός της είναι να διασφαλίσει ότι αυτός που επιχειρεί να έχει πρόσβαση είναι αυτός που ισχυρίζεται πως είναι. Η εξουσιοδότηση (authorization), αφορά το τι επιτρέπεται να κάνει ένας χρήστης αφού ταυτοποιηθεί, δηλαδή ποια δεδομένα ή λειτουργίες έχει δικαίωμα να χρησιμοποιήσει.

Για παράδειγμα σε ένα Kubernetes cluster, η ταυτοποίηση καθορίζει ποιος χρήστης ή ποια υπηρεσία μπορεί να εκτελέσει μια εντολή, ενώ η εξουσιοδότηση καθορίζει σε ποια namespaces ή pods επιτρέπεται να έχουν πρόσβαση ή να εκτελούν ενέργειες. Χωρίς σωστή ταυτοποίηση δεν υπάρχει εμπιστοσύνη και χωρίς σωστή εξουσιοδότηση δεν υπάρχει έλεγχος.

1.2 OpenID Connect (OIDC) και Ταυτοποίηση στο Sigstore

Το OpenID Connect (OIDC) είναι ένα πρωτόκολλο ταυτοποίησης το οποίο επιτρέπει στις εφαρμογές να επιβεβαιώνουν την ταυτότητα ενός χρήστη χωρίς να χρειάζεται να διαχειρίζονται οι ίδιες κωδικούς ή διαπιστευτήρια. Αντί να δημιουργεί κάθε υπηρεσία δικό της σύστημα log in, το OIDC επιτρέπει στον χρήστη να συνδεθεί μέσω ενός identity provider (IdP), όπως για παράδειγμα το google ή το github. Εφόσον ένας χρήστης μπορεί να γίνει authenticated από ένα IdP, τότε και η υπηρεσία τον θεωρεί authenticated.

Η λειτουργία του OIDC βασίζεται στην έκδοση ενός ID Token, το οποίο είναι ένα κρυπτογραφημένο JSON Web Token (JWT). Το token αυτό περιέχει πληροφορίες σχετικά με την ταυτότητα του χρήστη , τον πάροχο που το εξέδωσε και το χρονικό διάστημα που ισχύει. Όταν μια εφαρμογή λάβει το token, μπορεί να το επαληθεύσει με το δημόσιο κλειδί του IdP ώστε να βεβαιωθεί ότι ο χρήστης είναι πράγματι αυτός που ισχυρίζεται.

1.3 Πιστοποιητικά και Αρχές Πιστοποίησης

Τα ψηφιακά πιστοποιητικά αποτελούν την βάση της εμπιστοσύνης, καθώς συνδέουν ένα δημόσιο κλειδί με μια ταυτότητα, διασφαλίζοντας ότι το κλειδί ανήκει στον δηλωμένο κάτοχο. Η εγκυρότητα των πιστοποιητικών επιβεβαιώνεται από μια αρχή πιστοποίησης (CA), η οποία τα εκδίδει και τα υπογράφει, λειτουργώντας ως έμπιστος ενδιάμεσος.

Στο Sigstore τον ρόλο αυτό τον αναλαμβάνει το Fulcio, που λειτουργεί ως προσωρινό CA. Η λειτουργία του είναι να εκδίδει σύντομης διάρκειας πιστοποιητικά, τα οποία συσχετίζουν το κλειδί με την ταυτότητα του χρήστη μόνο για λίγα λεπτά.

1.4 Transparency Logs και Merkle Trees

Τα Transparency log είναι αρχεία καταγραφής όπου αποθηκεύονται όλες οι ενέργειες (έκδοση πιστοποιητικών, υπογραφές artifacts) με τρόπο αμετάβλητο και επαληθεύσιμο.

Η δομή τους είναι σε μορφή Merkle Tree, που είναι ένα δέντρο όπου κάθε κόμβος περιέχει ένα hash των παιδιών του. Έτσι και η παραμικρή αλλαγή να γίνει σε ένα φύλλο προκαλεί αλλαγές σε ολόκληρο το δέντρο, κάτι το οποίο κάνει την αλλοίωση άμεσα ανιχνεύσιμη.

1.5 Containers και Private registries

Τα containers αποτελούν απομονωμένα και ελαφριά περιβάλλοντα εκτέλεσης, τα οποία περιλαμβάνουν όλα τα απαραίτητα στοιχεία μιας εφαρμογής. Η αποθήκευση και η διανομή τους πραγματοποιείται μέσω container registries, τα οποία μπορεί να είναι είτε δημόσια, είτε ιδιωτικά.

Αν και τα δημόσια registries παρέχουν βασικούς μηχανισμούς ασφαλείας, όπως TLS και έλεγχο πρόσβασης, δεν προσφέρουν τον ίδιο βαθμό ελέγχου στις πολιτικές ασφαλείας, ούτε επιτρέπουν την διασφάλιση ότι όλα τα αποθηκευμένα images συμμορφώνονται με συγκεκριμένα πρότυπα, όπως συμβαίνει σε ένα private registry, όπου μπορούν να επιβληθούν πολύ αυστηρότερες πολιτικές.

1.6 Policy controller και Επαλήθευση υπογραφών στο Kubernetes

Ο policy controller είναι ένα εργαλείο που ενσωματώνεται στο Kubernetes και ελέγχει εάν τα images που επιχειρούν να εκτελεστούν είναι υπογεγραμμένα και έγκυρα σύμφωνα με συγκεκριμένες πολιτικές. Αν ένα image δεν διαθέτει έγκυρη υπογραφή που να συνδέεται με το επιτρεπτό OIDC, το deployment απορρίπτεται. Με αυτό τον τρόπο διασφαλίζεται η αυτόματη εφαρμογή των πολιτικών ασφαλείας.

https://doi.org/10.26219/heal.aueb.9571

2. Sigstore

2.1 Εισαγωγή

Η αυξανόμενη πολυπλοκότητα των σύγχρονων συστημάτων λογισμικού, σε συνδυασμό με την εκτεταμένη χρήση βιβλιοθηκών και πακέτων ανοικτού κώδικα, έχει κάνει την ασφάλεια της εφοδιαστικής αλυσίδας λογισμικού (software supply chain security) από τα σημαντικότερα ζητήματα στο DevSecOps περιβάλλον. Στόχος πλέον δεν είναι μόνο η ασφαλής ανάπτυξη, αλλά και η επαλήθευση της προέλευσης και η διασφάλιση της ακεραιότητας κάθε κομματιού λογισμικού που φτάνει στον τελικό χρήστη.

2.2 Περιγραφή και Βασικές Αρχές του Sigstore

Το sigstore είναι ένα project ανοικτού κώδικα, το οποίο ουσιαστικά είναι μια συλλογή εργαλείων και υπηρεσιών υποδομής που έχει ως στόχο την διευκόλυνση των προγραμματιστών στο να μπορούν να υπογράφουν και να επαληθεύουν με ασφάλεια αντικείμενα λογισμικού όπως release files, container images. Οι υπογραφές δημιουργούνται με εφήμερα (ephemeral) κλειδιά κάτι το οποίο έχει σαν αποτέλεσμα να μην χρειάζεται η διαχείριση κλειδιών. Επιπροσθέτως, τα συμβάντα υπογραφής καταγράφονται σε ένα δημόσιο αρχείο καταγραφής το οποίο είναι ανθεκτικό σε παραβιάσεις και παρέχει τη δυνατότητα ελέγχου στα συμβάντα υπογραφής από τους προγραμματιστές.

2.3 Προκλήσεις και Προβλήματα που Επιλύει

Στόχος του sigstore είναι να λύσει θέματα που αφορούν την ασφάλεια και την διαφάνεια στη διανομή λογισμικού. Στην πράξη, οι προγραμματιστές μπορούν να γράψουν κώδικα και να τα ανεβάσουν σε ένα αποθετήριο πακέτων. Στην συνέχεια, το πακέτο αυτό μπορεί να χρησιμοποιηθεί ως εξάρτηση σε άλλα έργα. Το πρόβλημα είναι ότι κανένας δεν επαληθεύει τη λειτουργία του κώδικα ή το ποιος είναι ο δημιουργός αυτού του κώδικα. Αυτό έχει ως αποτέλεσμα η αλυσίδα εφοδιασμού λογισμικού να έχει κάποια κενά ασφαλείας και ως απόρροια αυτού να είναι ευάλωτη σε τυχόν επιθέσεις που προέρχονται από κακόβουλα πακέτα. Ένα τέτοιο κακόβουλο πακέτο, ακριβώς επειδή μπορεί να χρησιμοποιηθεί ως εξάρτηση σε άλλα έργα, μπορεί να πλήξει μεγάλο αριθμό συστημάτων ταυτόχρονα. Η λύση σε αυτό το πρόβλημα είναι η χρήση υπογραφής στα πακέτα.

Πολλοί προγραμματιστές δεν χρησιμοποιούν την υπογραφή πακέτων, καθώς πρόκειται για μια διαδικασία πρακτικά δύσκολη και απαιτητική, δεδομένου ότι πρέπει κάποιος να διαχειρίζεται τα κρυπτογραφικά κλειδιά που χρειάζονται για την υπογραφή του πακέτου.

Αυτό σημαίνει ότι θα πρέπει να δημιουργηθούν τα κλειδιά, να γίνει η υπογραφή και στην συνέχεια να αποθηκευτούν με ασφάλεια. Σε περίπτωση που το ιδιωτικό κλειδί διαρρεύσει ή παραβιαστεί, απαιτείται χρόνος και συντονισμός για την αντικατάσταση του και την αποκατάσταση της εμπιστοσύνης στο νέο.

Έπειτα τίθεται το ζήτημα της αξιοπιστίας. Για να εμπιστευθεί κάποιος ένα πακέτο θα πρέπει να θεωρεί αξιόπιστο και τον δημιουργό του. Το παραδοσιακό μοντέλο web of trust δεν προσφέρει την δυνατότητα να επιβεβαιωθεί ότι ένα δημόσιο κλειδί στο διαδίκτυο ανήκει στην πραγματικότητα σε αυτόν που ισχυρίζεται ότι το δημιούργησε. Επομένως η εμπιστοσύνη παραμένει άτυπη και όχι επαληθεύσιμη. Επιπροσθέτως, όλα τα παραπάνω για να πραγματοποιηθούν πρέπει να υπάρχει υποστήριξη από τον διαχειριστή πακέτων ώστε να είναι εύκολη η υπογραφή, καθώς και η επαλήθευση της από τους τελικούς χρήστες. Δεν έχει κανένα νόημα η υπογραφή εάν όταν κατεβάζει κάποιος ένα υπογεγραμμένο πακέτο, δεν πραγματοποιείται η επαλήθευση της υπογραφής.

Την λύση αυτού του προβλήματος ήρθε να δώσει το Sigstore, το οποίο είναι ένα έργο ανοιχτού κώδικα το οποίο χρησιμοποιείται για την υπογραφή artifacts και σκοπός του είναι να κάνει πιο εύκολο στους προγραμματιστές να χρησιμοποιούν υπογραφές, χωρίς να χρειάζεται να διαχειρίζονται μόνιμα κρυπτογραφικά κλειδιά και να ενισχύσει την ασφάλεια της διαδικασίας υπογραφής. Στην ανάπτυξη και συντήρηση του Sigstore συμμετέχουν πολλές μεγάλες τεχνολογικές εταιρείες , όπως η Google, η Red Hat, GitHub, γεγονός που ενισχύει ακόμα περισσότερο την αξιοπιστία και τη διαφάνεια του.

2.4 Αρχιτεκτονική του Sigstore

Το Sigstore επιτρέπει την υπογραφή λογισμικών artifacts χωρίς την χρήση μόνιμων κρυπτογραφικών κλειδιών, καθώς το ιδιωτικό κλειδί που δημιουργείται για κάθε υπογραφή διαγράφεται αμέσως μετά την χρήση του. Με αυτόν τον τρόπο εξαλείφεται ο κίνδυνος παραβίασης ή διαρροής του ιδιωτικού κλειδιού και ταυτόχρονα επιλύεται και το βασικότερο πρόβλημα της διαχείρισης κλειδιών. Η αρχιτεκτονική του Sigstore βασίζεται σε δύο βασικές υπηρεσίες, το Fulcio και το Rekor, οι οποίες εξασφαλίζουν τη διαφάνεια, την ακεραιότητα και την επαληθευσιμότητα των υπογραφών. Επιπλέον, υπάρχει και το εργαλείο Cosign το οποίο λειτουργεί ως μέσο αλληλεπίδρασης με αυτές τις υπηρεσίες, έτσι ώστε οι προγραμματιστές και τα CI/CD pipelines να υπογράφουν, να αποθηκεύουν και να επαληθεύουν artifacts με αυτοματοποιημένο και αξιόπιστο τρόπο.

Στην συνέχεια παρουσιάζονται αναλυτικά οι τρεις βασικές συνιστώσες του Sigstore, οι οποίες είναι το Fulcio, το Rekor και το cosign. Επιπλέον, θα αναλυθεί ο τρόπος με τον οποίο συνεργάζονται για να διασφαλίσουν την εμπιστοσύνη, την διαφάνεια και την ακεραιότητα στην εφοδιαστική αλυσίδα λογισμικού.

Το Fulcio λειτουργεί ως αρχή πιστοποίησης (certificate authority) που εκδίδει βραχυπρόθεσμα πιστοποιητικά. Όταν ένας χρήστης θέλει να υπογράψει ένα artifact χρησιμοποιώντας το Sigstore, πρέπει πρώτα να συνδεθεί μέσω OpenID Connect (OIDC) χρησιμοποιώντας την ταυτότητα ενός αξιόπιστου παρόχου, όπως είναι η Google. Μόλις πραγματοποιηθεί επιτυχώς η ταυτοποίηση, το Fulcio δημιουργεί και εκδίδει ένα προσωρινό πιστοποιητικό που συνδέει την δημόσια ταυτότητα του χρήστη (π.χ. το email) με το δημόσιο κλειδί που δημιουργήθηκε ειδικά για την συγκεκριμένη υπογραφή. Τα πιστοποιητικά αυτά έχουν πολύ μικρή διάρκεια ισχύος (συνήθως 10 λεπτά), ώστε να περιορίζεται δραστικά η πιθανότητα κατάχρησης ή παραβίασης. Κάθε έκδοση πιστοποιητικού καταγράφεται δημόσια στο Rekor, επιτρέποντας να υπάρχει διαφάνεια, καθώς και εύκολη ανίχνευση ύποπτων ενεργειών.

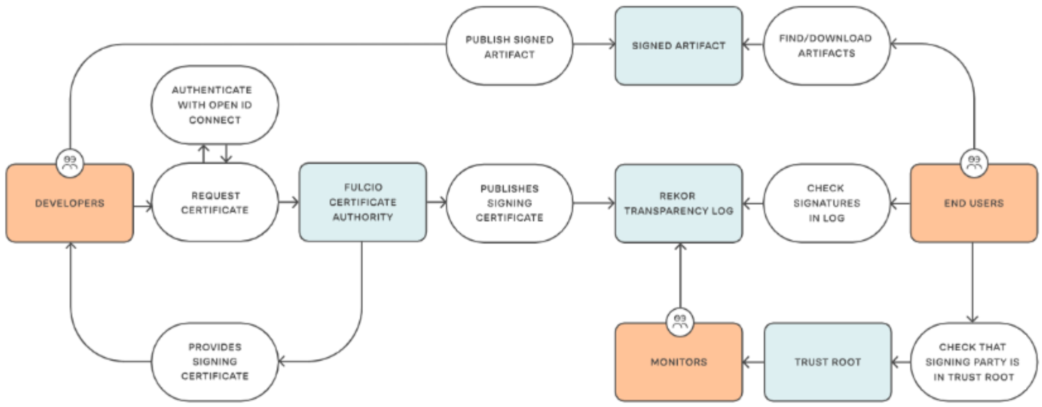
Το Rekor από την άλλη πλευρά, λειτουργεί ως δημόσιο αμετάβλητο μητρώο (transparency log) του οποίου η δομή βασίζεται σε Merkle Tree. Σε αυτό αποθηκεύονται τα μεταδεδομένα που σχετίζονται με τις υπογραφές, τα πιστοποιητικά και τα artifacts, αλλά όχι τα ίδια τα αρχεία, δηλαδή περιλαμβάνεται το hash του artifact, η υπογραφή του, το πιστοποιητικό που χρησιμοποιήθηκε και η ταυτότητα του υπογράφοντα. Χάρη στην δενδρική μορφή, κάθε νέα καταχώρηση συνδέεται κρυπτογραφικά με την προηγούμενη, διασφαλίζοντας ότι οποιαδήποτε αλλαγή είναι άμεσα ανιχνεύσιμη. Το Rekor επιτρέπει σε οποιονδήποτε θέλει να ελέγξει αν ένα artifact παραμένει αμετάβλητο μετά την υπογραφή του, κατεβάζοντας την υπογραφή και το δημόσιο κλειδί από το log.

Η συνεργασία μεταξύ του Fulcio και του Rekor είναι συμπληρωματική. Όταν το Fulcio εκδίδει ένα προσωρινό πιστοποιητικό που επιβεβαιώνει την ταυτότητα του υπογράφοντος, η υπογραφή που παράγεται με αυτό το πιστοποιητικό καταγράφεται στο Rekor, μαζί με τα αντίστοιχα μεταδεδομένα. Αυτό είναι ιδιαίτερα σημαντικό γιατί μπορεί ο οποιοσδήποτε να αποδείξει εύκολα ότι έχει υπογράψει ένα αρχείο, ή αντίστροφα, να επαληθεύσει την αυθεντικότητα ενός artifact αναζητώντας την αντίστοιχη καταχώριση στο Rekor.

Η τρίτη συνιστώσα του Sigstore, το cosign αποτελεί το εργαλείο μέσω του οποίου οι προγραμματιστές και τα CI/CD pipelines αλληλοεπιδρούν με το Sigstore. Ουσιαστικά το cosign λειτουργεί σαν γέφυρα ανάμεσα σε προγραμματιστές και τις υπηρεσίες Fulcio και Rekor, προσφέροντας έναν απλό τρόπο υπογραφής, αποθήκευσης και επαλήθευσης λογισμικών artifacts, όπως για παράδειγμα container images. Ένα από τα σημαντικότερα χαρακτηριστικά που έχει το cosign, είναι ότι υποστηρίζει keyless signing, δηλαδή έχει την δυνατότητα υπογραφής χωρίς τη χρήση μόνιμων ιδιωτικών κλειδιών. Για κάθε υπογραφή δημιουργείται ένα προσωρινό ζεύγος κλειδιών (private/public), το οποίο ισχύει μόνο για την συγκεκριμένη διαδικασία. Στην συνέχεια πραγματοποιείται ταυτοποίηση μέσω OIDC, όπως για παράδειγμα μέσω GitHub με σκοπό το σύστημα να γνωρίζει ποιος είναι ο υπογράφων. Έπειτα το cosign επικοινωνεί με το Fulcio, το οποίο όπως ειπώθηκε και προηγουμένως εκδίδει ένα βραχυπρόθεσμο πιστοποιητικό που συνδέει την ταυτότητα του χρήστη με το δημόσιο κλειδί που δημιουργήθηκε. Μετά το πέρας της υπογραφής το cosign χρησιμοποιεί το ιδιωτικό κλειδί για να δημιουργήσει την ψηφιακή υπογραφή του artifact και στην συνέχεια καταχωρεί την υπογραφή και το πιστοποιητικό στο Rekor. Για την επαλήθευση, το cosign αναζητά την υπογραφή και το πιστοποιητικό του artifact στο registry του Rekor, ώστε να ελέγξει ότι όλα είναι έγκυρα. Επιβεβαιώνει ότι η υπογραφή αντιστοιχεί στο περιεχόμενο του artifact, ότι το πιστοποιητικό έχει εκδοθεί από το Fulcio και ότι η σχετική καταχώρηση υπάρχει στο Rekor. Με αυτό τον τρόπο εξασφαλίζεται ότι το artifact είναι αυθεντικό και δεν έχει τροποποιηθεί μετά την υπογραφή του. Σημαντικό να αναφερθεί είναι ότι το cosign εκτός από την απλή υπογραφή, υποστηρίζει και attestations, δηλαδή πληροφορίες που συνοδεύουν το artifact και περιγράφουν το πώς δημιουργήθηκε. Μερικές από τις πληροφορίες είναι το ποιος δημιούργησε το αρχείο και ποια πακέτα ή βιβλιοθήκες χρησιμοποιήθηκαν. Αντίστοιχα και οι attestations υπογράφονται και καταγράφονται στο Rekor, προσφέροντας ένα επιπλέον επίπεδο διαφάνειας.

Πέρα από τη διαδικασία υπογραφής και επαλήθευσης, το cosign μπορεί να συνδυαστεί με πολιτικές ασφαλείας μέσω του Kubernetes policy controller, ώστε ένα σύστημα να επιτρέπει την εκτέλεση μόνο των εικόνων που είναι σωστά υπογεγραμμένες και επαληθευμένες μέσω Sigstore. Με αυτόν τον τρόπο, η ασφάλεια της εφοδιαστικής αλυσίδας λογισμικού ενισχύεται από την δημιουργία έως την εκτέλεση των εφαρμογών.

Διάγραμμα 1 Το οικοσύστημα του Sigstore



Πηγή: Sigstore

3. Supply Chain Attacks

Επίθεση στην εφοδιαστική αλυσίδα έχουμε όταν ο επιτιθέμενος στοχεύει σε κάποιο ενδιάμεσο στοιχείο της αλυσίδας που καταλήγει στο τελικό προϊόν ή υπηρεσία και όχι απευθείας στο τελικό θύμα. Αυτό το ενδιάμεσο στοιχείο μπορεί να είναι βιβλιοθήκη από κάποιον τρίτο, κάποιος προμηθευτής λογισμικού, ένας πάροχος υπηρεσιών, ακόμα και ένα εργαλείο build (CI/CD). Σαν βασική ιδέα έχει ότι αν καταφέρει κάποιος να μολύνει ένα αξιόπιστο κομμάτι της αλυσίδας, τότε η μόλυνση θα διανεμηθεί μέσω των κανονικών, έμπιστων διαύλων.

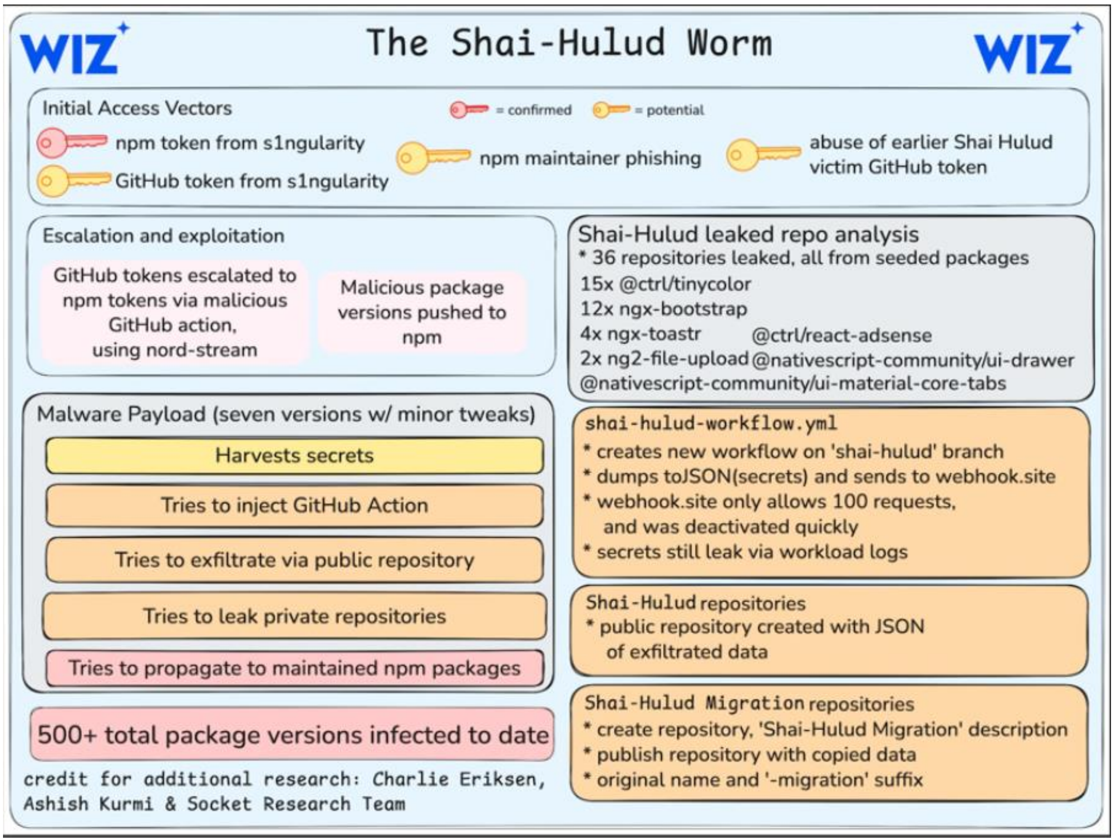
Στόχος αυτών των επιθέσεων είναι να πετύχουν μια επίθεση μεγάλης εμβέλειας, δηλαδή να έχει πολλαπλά θύματα κάτι το οποίο είναι πολύ εύκολο αφού με ένα μολυσμένο πακέτο ή ένα κακόβουλο update μπορεί να επηρεάσει ταυτόχρονα πολλούς οργανισμούς. Επιπλέον, οι επιτιθέμενοι επιδιώκουν να εισάγουν Backdoors που θα τους επιτρέπουν μακροχρόνια πρόσβαση.

Μια γενική εικόνα για το πώς λειτουργεί είναι η ακόλουθη. Στην αρχή ο επιτιθέμενος κάνει επιλογή του στόχου του με βάση τι θα έχει μεγαλύτερη επίδραση ή αδύναμη ασφάλεια ώστε να αποκτήσει πρόσβαση στο επιλεγμένο σημείο. Στην συνέχεια εισάγει κακόβουλο κώδικα ή τροποποιεί ένα artifact ή firmware ή πακέτο. Το στοιχείο το οποίο τροποποίησε διανέμεται στους τελικούς χρήστες μέσω έμπιστων καναλιών (updates, repos). Όταν το στοιχείο φτάσει στον στόχο συνήθως δεν ενεργοποιείται αμέσως για να μην γίνει άμεσα ανιχνεύσιμο. Μπορεί να ενεργοποιηθεί, για παράδειγμα, μόλις το artifact τρέξει, να προσπαθήσει να δημιουργήσει ασφαλή τρόπο επικοινωνίας με τον επιτιθέμενο, και να προσπαθήσει να συλλέξει στοιχεία για τα συστήματα, τους χρήστες, ώστε στην συνέχεια να προχωρήσει σε access escalation.

Ένα πρόσφατο περιστατικό επίθεσης στην εφοδιαστική αλυσίδα του Npm (Node package manager), έγινε τον Σεπτέμβριο του 2025. Ουσιαστικά αυτό που έγινε, ήταν οι επιτιθέμενοι να στείλουν Phishing emails που μιμούνταν ανακοινώσεις υποστήριξης του Npm, ζητώντας από τους συντηρητές πακέτων να κάνουν επαναφορά ή να επιβεβαιώσουν τον κωδικό 2FA. Έτσι κατάφεραν να πάρουν τα διαπιστευτήρια τους και να πάρουν πλήρη πρόσβαση στους λογαριασμούς των συντηρητών. Στην συνέχεια ανέβασαν νέες εκδόσεις, οι οποίες περιλαμβάνανε post-install scripts τα οποία κατά την εγκατάσταση (npm install) εκτελούνταν. Επιπλέον, το malware αυτό είχε την μορφή worm και έτσι κάθε φορά που εκτελούταν έκανε και Install to TruffleHog το οποίο είναι ένα open source εργαλείο που μπορεί να εντοπίσει πάνω από 800 διαφορετικούς τύπους μυστικών, κάτι το οποίο αύξανε πολύ την δυνατότητα να κλαπεί ένα μυστικό. Με αυτό τον τρόπο έκλεβαν tokens, άλλαζαν διευθύνσεις κρυπτονομισμάτων στις εφαρμογές και έβαζαν δικές τους διευθύνσεις, καθώς επίσης έψαχναν και για μυστικά μέσα στους υπολογιστές των προγραμματιστών. Το malware σάρωνε το σύστημα για tokens, SSH keys, GitHub/GitLab API keys κ.α και στην συνέχεια τα έστελνε σε GitHub repository που είχε την ονομασία Shai-Hulud. Όταν το malware εντόπιζε διαπιστευτήρια που είχαν δικαιώματα publish, έκανε χρήση τους ώστε να ανεβάσει νέες μολυσμένες εκδόσεις άλλων πακέτων, κι έτσι εξαπλωνόταν από προγραμματιστή σε προγραμματιστή. Τέλος, οι επιτιθέμενοι φρόντισαν να δημιουργήσουν workflows και τροποποιημένα artifacts ώστε να γίνεται επαναδημοσίευση αυτόματα του malware ακόμα κι αν καθαριζόταν ο αρχικός υπολογιστής, δηλαδή μπορούσαν να διατηρήσουν την πρόσβαση μέσα σε υποδομές αυτοματισμού (CI/CD) ακόμα και αν έχαναν την πρόσβαση από ένα μολυσμένο Laptop ή λογαριασμό.

https://doi.org/10.26219/heal.aueb.9571

Figure 1 Γραφική σύνοψη της επίθεσης Shai-Hulud στην εφοδιαστική αλυσίδα λογισμικού



Πηγή: Wiz.io – Shai-Hulud: npm supply chain attack

Για να είχε περιοριστεί σημαντικά ή αποφευχθεί η επίθεση που έγινε στο Npm θα έπρεπε να είχαν εφαρμοστεί κάποια προστατευτικά μέτρα ασφαλείας. Πρώτα από όλα, που ήταν και το πιο κρίσιμο μέτρο ασφαλείας στην συγκεκριμένη επίθεση καθώς έγινε μέσω phishing email, ήταν ότι οι λογαριασμοί των συντηρητών (maintainers) θα έπρεπε να χρησιμοποιούσαν ισχυρή ταυτοποίηση με φυσικά κλειδιά ασφαλείας (hardware MFA) και όχι απλούς κωδικούς ή εφαρμογές OTP. Έτσι οι επιτιθέμενοι δεν θα μπορούσαν να αποκτήσουν πρόσβαση ακόμα και αν έπαιρναν τα στοιχεία κάποιου συντηρητή, διότι θα έπρεπε να γίνει και επιβεβαίωση με το φυσικό κλειδί το οποίο δεν θα είχαν στην κατοχή τους. Δεύτερον, κάθε νέα έκδοση πακέτου θα έπρεπε να συνοδεύεται από ψηφιακή υπογραφή , απόδειξη προέλευσης (artifact provenance) καθώς και SBOMs (software bill of materials), μέσω εργαλείων όπως το Sigstore ή cosign, έτσι ώστε να μην μπορούν να δημοσιευθούν ψεύτικες εκδόσεις και να γίνουν αποδεκτά από το Npm registry μόνο πακέτα που θα είχαν δημιουργηθεί μέσα από ασφαλή, ελεγχόμενα περιβάλλοντα CI. Επιπλέον, δεν θα έπρεπε να έχουν long-lived static keys καθώς δίνουν στον επιτιθέμενο παρατεταμένη μη εξουσιοδοτημένη πρόσβαση σε συστήματα και

https://doi.org/10.26219/heal.aueb.9571

δεδομένα. Θα έπρεπε να χρησιμοποιούν short-lived tokens, τα οποία λήγουν σύντομα, με αποτέλεσμα αν κλαπούν από τον επιτιθέμενο, να μπορεί να τα χρησιμοποιήσει για πολύ λίγο (π.χ 10 λεπτά). Τέλος, οι maintainers αν είχαν εφαρμόσει σωστό έλεγχο αλλαγών στα CI pipelines (π.χ. ενεργό audit, branch protection, approval steps για wokflows), τότε οι επιτιθέμενοι δεν θα μπορούσαν να προσθέσουν ή να ενεργοποιήσουν κακόβουλα workflows αφού κάθε νέο ή τροποποιημένο workflow θα χρειαζόταν έγκριση πριν εκτελεστεί και ακόμα και αν τα καταφέρναν, από τα audit logs θα κατέγραφαν ποιος το δημιούργησε ή ενέκρινε το workflow.

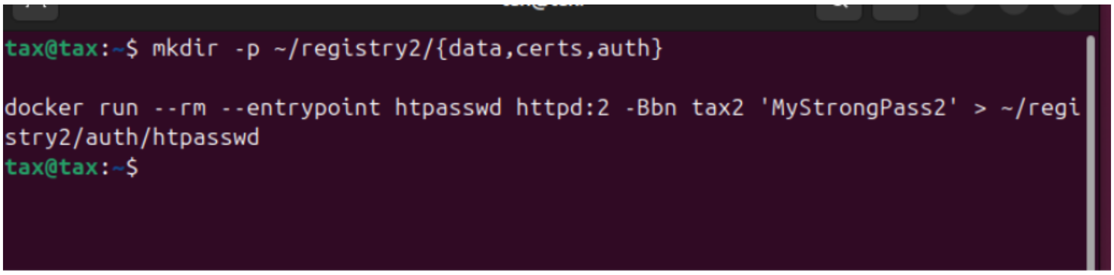
Οι επιθέσεις στην εφοδιαστική αλυσίδα έχουν αυξηθεί δραματικά και η αλήθεια είναι ότι κανείς δεν είναι απρόσβλητος από αυτές. Η κατανεμημένη αρχιτεκτονική που χρησιμοποιείται για την κατασκευή εφαρμογών δίνει στους εισβολείς την τέλεια ευκαιρία να στοχεύσουν εκατοντάδες ακόμη και χιλιάδες οργανισμούς ταυτόχρονα. Ακριβώς λόγω της μεγάλης απόδοσης από μια τέτοια επίθεση, οι επιτιθέμενοι χρησιμοποιούν περισσότερους πόρους για να στοχεύσουν την εφοδιαστική αλυσίδα.

https://doi.org/10.26219/heal.aueb.9571

4. Πρακτικό Μέρος

Το πρακτικό μέρος της παρούσας διπλωματικής εργασίας επικεντρώνεται στην υλοποίηση ενός ασφαλούς συστήματος διαχείρισης container images με την χρήση του Sigstore. Συγκεκριμένα περιγράφεται η διαδικασία δημιουργίας ενός private registry και η ενσωμάτωση του σε περιβάλλον Kubernetes. Στην συνέχεια γίνεται εφαρμογή των πολιτικών επαλήθευσης υπογραφών για την εξασφάλιση της ακεραιότητας των images. Επιπλέον, παρουσιάζεται η χρήση του εργαλείου cosign για την υπογραφή και επαλήθευση των images, καθώς επίσης και η ανάπτυξη και χρήση ενός εργαλείου watcher script που αυτοματοποιεί τον έλεγχο και την διαγραφή μη υπογεγραμμένων εικόνων.

4.1 Δημιουργία και ρύθμιση ενός private registry



Αρχικά, ορίζεται νέος χρήστης και κωδικός. Αυτό χρειάζεται ώστε να απαιτείται authentication σε Push και pull. Χωρίς authentication, οποιοσδήποτε στο ίδιο δίκτυο θα μπορούσε να κάνει push κακόβουλα images.

```
stry2/auth/htpasswd
tax@tax:~$ HOST_IP=$(ip route get 1.1.1.1 | awk '{print $7; exit}')
tax@tax:~$ openssl req -x509 -newkey rsa:4096 -sha256 -nodes \
  -keyout ~/registry2/certs/domain.key \
  -out ~/registry2/certs/domain.crt \
  -days 365 \
  -subj "/CN=registry2.local" \
  -addext "subjectAltName=DNS:registry2.local,IP:${HOST_IP}"
.+.+..+......+....+......+..+.......+......+...........+++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++*.....+...+.........+.....+.+...........
.+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*............
....+..........+...........+.+..+...+....+.....+...+...+....+.........+.......+..
....+........+.+......+.................+..........+..+.+...................+..
..........+......+....................+...................+.+..........+...+..+..
....+......+....+...........+...+.+...........+..............+...+..+.+.....+.+..
.+...+...+..............+..........+.+.............+................+..+.....
..+.....................+..+++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
...............+....+..+...+......+...+...+...............+++++++++++++++++++++
```

Η εντολή ip route get 1.1.1.1 εντοπίζει την τοπική IP στο δίκτυο Lan και στην συνέχεια η διεύθυνση αυτή αποθηκεύεται στη μεταβλητή host_IP. Έπειτα δημιουργείται self-signed TLS πιστοποιητικό για το registry με χρήση openssl req. Επιπροσθέτως, ορίζεται που θα αποθηκευτεί το ιδιωτικό κλειδί και το πιστοποιητικό και δίνεται η διάρκεια ισχύς του πιστοποιητικού. Επιπλέον, καταχωρούνται τα ονόματα/διευθύνσεις με τα οποία θα γίνεται πρόσβαση.

```
tax@tax:~$ docker rm -f registry2 2>/dev/null || true
docker run -d --name registry2 --restart=always \
  -p 5001:5000 \
  -v ~/registry2/data:/var/lib/registry \
  -v ~/registry2/certs:/certs:ro \
  -v ~/registry2/auth:/auth:ro \
  -e REGISTRY_HTTP_ADDR=:5000 \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
  -e REGISTRY_AUTH=htpasswd \
  -e REGISTRY_AUTH_HTPASSWD_REALM="Registry2 Realm" \
  -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
  registry:2
docker ps --filter name=registry2
c33a66f047ee793a306f3a4ffc777c50b47209c2b65733cb844513945b253faa
CONTAINER ID   IMAGE        COMMAND                  CREATED                   STATUS                   PORTS
  NAMES
c33a66f047ee   registry:2   "/entrypoint.sh /etc…"   Less than a second ago   Up Less than a second   0.0.0.0:5001->5000/tcp, [::]:5001->5000/tcp
  registry2
tax@tax:~$
```

Η παραπάνω εντολή ξεκινάει ένα docker registry container με όνομα registry2, στην αρχή γίνεται Port mapping, το 5001 είναι ο host και το 5000 ο container (ουσιαστικά η πόρτα που ακούει ο registry μέσα στον container). Στην συνέχεια γίνονται mount τα δεδομένα του registry και του φακέλου με τα TLS αρχεία. Έπειτα, γίνεται ρύθμιση HTTP server μέσα στο registry και με την προσθήκη των TLS αρχείων ο registry λειτουργεί με https. Τέλος, με την εντολή docker ps βλέπουμε όλα τα container που τρέχουν, των οποίων το όνομα περιέχει το registry2. Εδώ θα μπούνε αργότερα τα images.

```
tax@tax:~$ # /etc/hosts
sudo sed -i '/registry2\.local/d' /etc/hosts
echo "${HOST_IP} registry2.local" | sudo tee -a /etc/hosts
[sudo] password for tax:
10.0.2.15 registry2.local
tax@tax:~$
```

Αυτές οι δύο γραμμές φτιάχνουν την εγγραφή στο /etc/hosts για να αναγνωρίζει το μηχάνημα το όνομα registry2, ώστε κάθε φορά που γράφεται https://registry2.local:5001 ο υπολογιστής να ξέρει ότι πρέπει να συνδεθεί στην IP του host

```
tax@tax:~$ sudo mkdir -p /etc/docker/certs.d/registry2.local:5001/
tax@tax:~$ sudo cp ~/registry2/certs/domain.crt /etc/docker/certs.d/registry2.local:5001/ca.crt
tax@tax:~$ sudo systemctl restart docker
```

Δημιουργεί έναν φάκελο στο /etc/docker/certs.d/ που αντιστοιχεί στο registry. Στην συνέχεια, αντιγράφει το self-signed certificate που φτιάχτηκε μέσα στον φάκελο που στήθηκε για τον docker και γίνεται μετονομασία του σε ca.crt, γιατί ο docker περιμένει έτσι να λέγεται το Trusted certificate.

Τέλος, πραγματοποιείται restart στον docker daemon για να φορτώσει τη νέα ρύθμιση και να πάρει υπόψη το πιστοποιητικό.

```
tax@tax:~$ NODE=$(kubectl get nodes -o jsonpath='{.items[0].metadata.name}')
tax@tax:~$ docker exec -it "$NODE" bash -lc "sed -i '/registry2\.local/d' /etc/hosts; echo '${HOST_IP} registry2.local' >> /etc/hosts; tail -n1 /e
tc/hosts"
sed: cannot rename /etc/sed14VgKz: Device or resource busy
10.0.2.15 registry2.local
```

Παίρνει το όνομα του πρώτου node απο το cluster και το αποθηκεύει στην μεταβλητή NODE (βρίσκει το όνομα του node που θα χρησιμοποιηθεί για την εκτέλεση εντολών μέσα σε αυτό).

Στην συνέχεια εκτελείται μια εντολή μέσα στο node και πλέον είναι γνωστό ότι το όνομα registry2.local αντιστοιχεί στη διεύθυνση του Host όπου τρέχει το registry. Οπότε όταν το Node θα πάει να τραβήξει το Image ξέρει που θα συνδεθεί και το Host name ταιριάζει με το TLS cert.

```
tax@tax:~$ docker exec -it "$NODE" bash -lc "\
  mkdir -p /etc/containerd/certs.d/registry2.local:5001 && \
  cat >/etc/containerd/certs.d/registry2.local:5001/hosts.toml <<'EOF'
server = \"https://registry2.local:5001\"
[host.\"https://registry2.local:5001\"]
  capabilities = [\"pull\", \"resolve\"]
  ca = [\"/etc/containerd/certs.d/registry2.local:5001/ca.crt\"]
EOF
"
docker cp ~/registry2/certs/domain.crt "$NODE":/etc/containerd/certs.d/registry2.local:5001/ca.crt
Successfully copied 3.58kB to trust-cluster-control-plane:/etc/containerd/certs.d/registry2.local:5001/ca.crt
tax@tax:~$ docker exec -it "$NODE" bash -lc "pkill -SIGHUP containerd || pkill containerd; sleep 2; (containerd >/var/log/containerd.log 2>&1 &)"
```

Δημιουργεί τον φάκελο ρυθμίσεων του container για το συγκεκριμένο registry. Χωρίς τον φάκελο αυτόν ο container δεν θα έχει θέση να ψάξει για CA ή ρυθμίσεις TLS.

Η εντολή cat δημιουργεί ένα αρχείο ρυθμίσεων hosts.toml μέσα στον φάκελο, όπου μέσα εκεί δηλώνεται η βασική URL του registry, ορίζονται τα capabilities ώστε το Node να μπορεί να κάνει pull και αναζήτηση manifest και υποδεικνύεται που βρίσκεται το πιστοποιητικό που πρέπει να εμπιστεύεται.

Στην συνέχεια, αντιγράφει το self-signed certificate μέσα στο node και το τοποθετεί εκεί που δείχνει το host.toml ως CA, ώστε να ξέρει ότι μπορεί να κάνει pull από εκεί. Τέλος γίνεται restart ο container ώστε να πάρει τις ρυθμίσεις.

```
tax@tax:~$ sudo cp ~/registry2/certs/domain.crt /usr/local/share/ca-certificates/registry2.crt
sudo update-ca-certificates
[sudo] password for tax:
Updating certificates in /etc/ssl/certs...
rehash: warning: skipping ca-certificates.crt,it does not contain exactly one certificate or CRL
rehash: warning: skipping duplicate certificate in registry.local.pem
1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
tax@tax:~$
```

Πραγματοποιείται αντιγραφή του self-signed certificate (domain.crt) μέσα στον φάκελο /usr/local/share/ca-certificate και μετονομάζεται σε registry2.crt. Με αυτό δηλώνεται ότι το πιστοποιητικό (registry2.crt) είναι μια CA που θεωρείται έμπιστη. Όλα τα προγράμματα του υπολογιστή πλέον θα την θεωρούν έμπιστη και όχι μονο ο Docker, άρα το registry2.local θα θεωρείται ως έγκυρο HTTPS endpoint. Έτσι επιτυγχάνεται το cosign να εμπιστευτεί το certificate του registry2 στο host. Τέλος, γίνεται ενημέρωση της λίστας έμπιστων πιστοποιητικών.

4.2 Ενσωμάτωση private registry στο Kubernetes και εφαρμογή πολιτικών ασφαλείας

 https://doi.org/10.26219/heal.aueb.9571

```
tax@tax:~$ kubectl create namespace signed-images
kubectl label namespace signed-images policy.sigstore.dev/include=true --overwrite
namespace/signed-images created
namespace/signed-images labeled
tax@tax:~$ kubectl -n signed-images delete secret regcred2 --ignore-not-found
kubectl -n signed-images create secret generic regcred2 \
  --from-file=.dockerconfigjson=$HOME/.docker/config.json \
  --type=kubernetes.io/dockerconfigjson
secret/regcred2 created
tax@tax:~$
```

Δημιουργείται ένα νέο namespace στο kubernetes με όνομα signed-images, ώστε να έχει ξεχωριστό χώρο όπου θα εφαρμόζονται οι πολιτικές. Στην συνέχεια ενεργοποιείται ο Policy controller σε αυτό το namespace. Ακολουθεί η διαγραφή παλαιού τυχόν secret και στην συνέχεια δημιουργείται νέο Imagepullsecret. Έτσι έχει τοποθετηθεί μέσα του ένα Imagepullsecret με τα login στοιχεία για το registry.

```
tax@tax:~$ nano ~/secure-container/sa-myapp2-puller.yaml
tax@tax:~$ kubectl apply -f ~/secure-container/sa-myapp2-puller.yaml
serviceaccount/sa-myapp2-puller unchanged
tax@tax:~$
```

```
  GNU nano 7.2                                              sa-myapp2-puller.yaml *
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-myapp2-puller
  namespace: signed-images
imagePullSecrets:
  - name: regcred2
```

Με την εντολή nano sa-myapp2-puller.yaml δημιουργείται το service account με όνομα sa-myapp2-puller το οποίο κουβαλάει το secret (regcred2) για να μπορεί το Pod να κατεβάσει image από το Private registry. Οπότε αντί να τοποθετούνται secrets σε κάθε pod, αυτά συνδέονται με το SA (service account) και το χρησιμοποιούν όλα τα pods που το αναφέρουν.

```
GNU nano 7.2                                   /home/tax/secure-container/policy2.yaml
apiVersion: policy.sigstore.dev/v1beta1
kind: ClusterImagePolicy
metadata:
  name: cip-allow-tntezos-github-2
spec:
  images:
    - glob: "registry2.local:5001/**"
  mode: enforce
  authorities:
    - keyless:
        url: https://fulcio.sigstore.dev
        identities:
          - issuer: "https://github.com/login/oauth"
            subject: "t.ntenezos@gmail.com"
      ctlog:
        url: https://rekor.sigstore.dev
```

Το kind καθορίζει τον τύπο του αντικειμένου που στην συγκεκριμένη περίπτωση είναι ClusterImagePolicy (CIP) και ισχύει για ολόκληρο το cluster. Στο metadata ορίζεται το όνομα του αντικειμένου. Στο spec καθορίζονται οι κανόνες της πολιτικής , όπου στην προκειμένη περίπτωση αυτή η πολιτική καλύπτει όλα τα image από το registry2.local:5001. Σημαντικό είναι και το Mode το οποίο όταν είναι enforce, αν το Image δεν είναι επαληθευμένο το Pod απορρίπτεται. Τέλος, τα authorities καθορίζουν ποιοι υπογράφοντες είναι έγκυροι.

https://doi.org/10.26219/heal.aueb.9571

```
tax@tax:~$ nano ~/secure-container/policy2.yaml
tax@tax:~$ kubectl apply -f ~/secure-container/policy2.yaml
kubectl describe clusterimagepolicy cip-allow-tntezos-github-2
clusterimagepolicy.policy.sigstore.dev/cip-allow-tntezos-github-2 created
Name:         cip-allow-tntezos-github-2
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  policy.sigstore.dev/v1beta1
Kind:         ClusterImagePolicy
Metadata:
  Creation Timestamp:  2025-10-01T22:20:28Z
  Finalizers:
    clusterimagepolicies.policy.sigstore.dev
  Generation:        1
  Resource Version:  186671
  UID:               de3d3cc9-e255-4661-b881-0eda2ab1d6a9
Spec:
  Authorities:
    Ctlog:
      URL:  https://rekor.sigstore.dev
    Keyless:
      Identities:
        Issuer:   https://github.com/login/oauth
        Subject:  t.ntenezos@gmail.com
      URL:        https://fulcio.sigstore.dev
    Name:         authority-0
  Images:
    Glob:  registry2.local:5001/**
  Mode:    enforce
Status:
  Conditions:
    Last Transition Time:  2025-10-01T22:20:28Z
    Status:                True
```

Εδώ φαίνεται ο clusterimagepolicy (CIP), ο οποίος περιγράφει τους κανόνες πολιτικής για το ποιες εικόνες (Images) επιτρέπεται να τρέχουν.

```
tax@tax:~$ kubectl -n cosign-system create configmap registry2-ca \
  --from-file=registry2.crt=$HOME/registry2/certs/domain.crt
configmap/registry2-ca created
tax@tax:~$ kubectl -n cosign-system patch deploy policy-controller-webhook --type='json' -p='[
  {"op":"add","path":"/spec/template/spec/volumes/-","value":{"name":"registry2-ca","configMap":{"name":"registry2-ca"}}},
  {"op":"add","path":"/spec/template/spec/containers/0/volumeMounts/-","value":{"name":"registry2-ca","mountPath":"/etc/ssl/certs/registry2.crt","
subPath":"registry2.crt","readOnly":true}},
  {"op":"add","path":"/spec/template/spec/containers/0/env/-","value":{"name":"SSL_CERT_FILE","value":"/etc/ssl/certs/registry2.crt"}}
]'
Warning: spec.template.spec.containers[0].env[7]: hides previous definition of "SSL_CERT_FILE", which may be dropped when using apply
deployment.apps/policy-controller-webhook patched
tax@tax:~$ kubectl -n cosign-system rollout status deploy/policy-controller-webhook
Waiting for deployment "policy-controller-webhook" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "policy-controller-webhook" rollout to finish: 1 old replicas are pending termination...
deployment "policy-controller-webhook" successfully rolled out
tax@tax:~$
```

Δημιουργήθηκε ένα configmap registry2-ca στο namespace cosign-system που περιέχει το self-signed cert του registry ως αρχείο registry2.crt. Αυτό αποσκοπεί στο να καταστήσει το πιστοποιητικό διαθέσιμο στο pod του policy-controller, ώστε να εμπιστεύεται το TLS του registry κατά την επαλήθευση των υπογραφών. Τέλος, εφαρμόστηκε ένα Patch και πραγματοποιήθηκε restart ώστε να πάρει την νέα ρύθμιση.

4.3 Υπογραφή και επαλήθευση container images μέσω sigstore

```
tax@tax:~$ cat > ~/secure-container/Dockerfile-diplomatiki-demo3 <<'EOF'
FROM alpine:3.19
RUN echo "this is diplomatiki-demo3" > /hello.txt
CMD ["/bin/sh","-c","cat /hello.txt; date; sleep 3600"]
EOF
tax@tax:~$ docker build -f ~/secure-container/Dockerfile-diplomatiki-demo3 \
  -t registry2.local:5001/diplomatiki-demo3:1.0 ~/secure-container
[+] Building 0.4s (6/6) FINISHED                                  docker:default
 => [internal] load build definition from Dockerfile-diplomatiki-demo3     0.0s
 => => transferring dockerfile: 178B                                       0.0s
 => [internal] load metadata for docker.io/library/alpine:3.19             0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => CACHED [1/2] FROM docker.io/library/alpine:3.19                        0.0s
 => [2/2] RUN echo "this is diplomatiki-demo3" > /hello.txt                0.2s
 => exporting to image                                                     0.0s
 => => exporting layers                                                    0.0s
 => => writing image sha256:756c17fcec4115c40e7049b643cd844ea560490988cdd  0.0s
 => => naming to registry2.local:5001/diplomatiki-demo3:1.0                0.0s
tax@tax:~$ docker login registry2.local:5001 -u tax2 -p MyStrongPass2
docker push registry2.local:5001/diplomatiki-demo3:1.0
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
The push refers to repository [registry2.local:5001/diplomatiki-demo3]
eca90d8699cb: Pushed
0b44b2151d78: Mounted from diplomatiki-demo2
1.0: digest: sha256:febc7243e444633d09b412fad9f826046f6dc102386ee99577f221b69ac92101 size: 735
tax@tax:~$
```

Δημιουργήθηκε ένα Dockerfile αρχείο με όνομα Dockerfile-diplomatiki-demo3 μέσα στον φάκελο secure container. Στο Dockerfile καταγράφεται η δημιουργία ενός απλού αρχείου μέσα στο image και στην συνέχεια όταν το container ξεκινά θα εμφανίζει το περιεχόμενο του αρχείου και θα μένει ανοιχτό για 1 ώρα. Έπειτα, γίνεται Build το image και δίνεται το tag (το όνομα και τον προορισμό στο registry). Στην συνέχεια γίνεται είσοδος στο private registry και μετά Push το image. Ο Docker client επικοινωνεί με το registry μέσω HTTPS (port 5001).

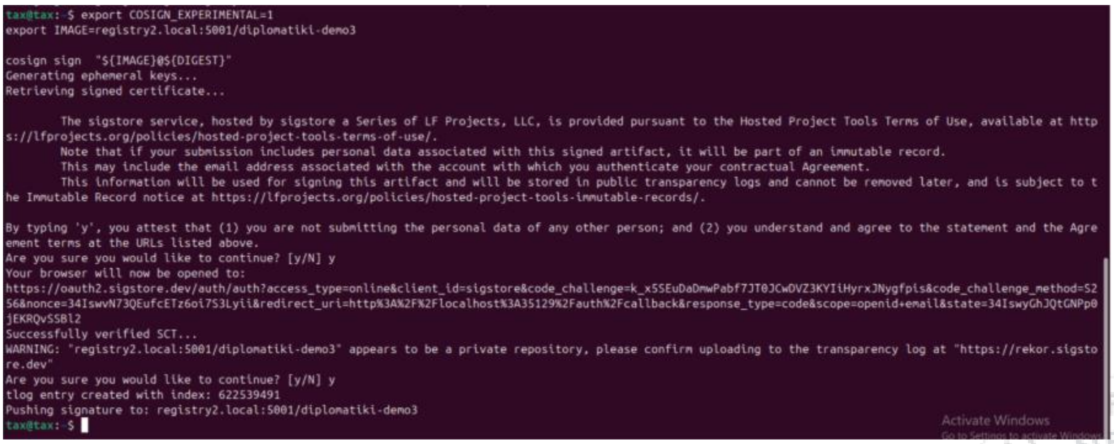
```
tax@tax:~$ curl -v -u tax2:MyStrongPass2 \
  --cacert ~/registry2/certs/domain.crt \
  -H 'Accept: application/vnd.docker.distribution.manifest.v2+json' \
  https://registry2.local:5001/v2/diplomatiki-demo3/manifests/1.0 2>&1 | \
  grep -i "Docker-Content-Digest" | \
  awk '{print $3}' | tr -d '\r'
sha256:febc7243e444633d09b412fad9f826046f6dc102386ee99577f221b69ac92101
tax@tax:~$ export DIGEST=sha256:febc7243e444633d09b412fad9f826046f6dc102386ee99577f221b69ac92101
tax@tax:~$
```

https://doi.org/10.26219/heal.aueb.9571

Εκτελείται η εντολή curl με authentication προς το Registry. Στο curl δηλώνεται ότι πρέπει να εμπιστευτεί το self signed πιστοποιητικό του registry για να θεωρηθεί η HTTPS σύνδεση ασφαλής. Έπειτα με την παράμετρο accept στέλνετε custom HTTP header για να ζητήσει το manifest του image και στην συνέχεια είναι το URL του endpoint που γίνεται το αίτημα. Τέλος εξάγουμε το hash του image και γίνεται αποθήκευση του σε μια μεταβλητή DIGEST.

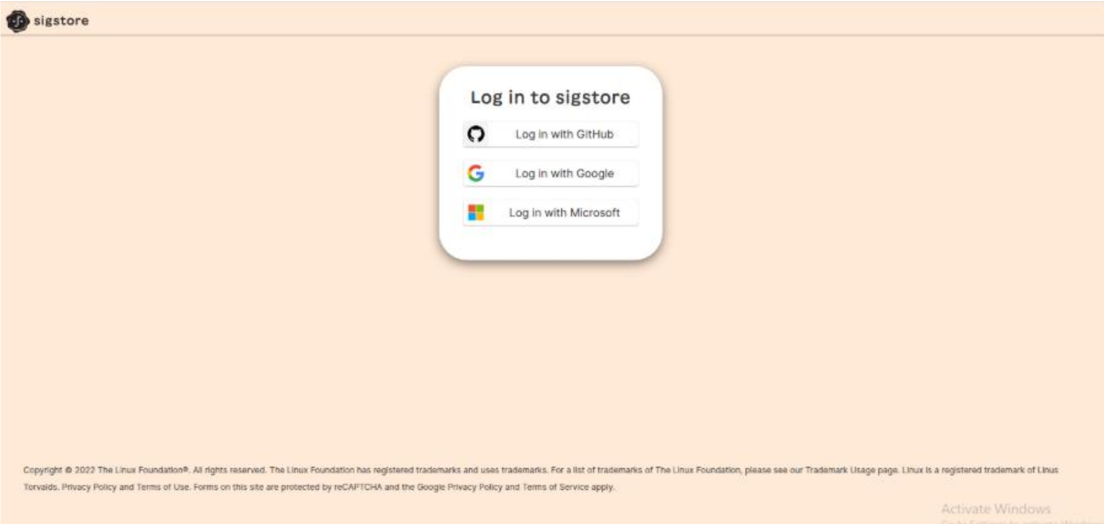
```
tax@tax:~$ cat > ~/secure-container/deploy-diplomatiki-demo3.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: diplomatiki-demo3-deploy
  namespace: signed-images
spec:
  replicas: 1
  selector:
    matchLabels:
      app: diplomatiki-demo3
  template:
    metadata:
      labels:
        app: diplomatiki-demo3
    spec:
      serviceAccountName: sa-myapp2-puller
      containers:
        - name: demo
          image: registry2.local:5001/diplomatiki-demo3@${DIGEST}
          imagePullPolicy: IfNotPresent
          command: ["sh","-c","echo 'unsigned -> should be DENIED'; sleep 3600"]
EOF

kubectl apply -f ~/secure-container/deploy-diplomatiki-demo3.yaml
Error from server (BadRequest): error when creating "/home/tax/secure-container/deploy-diplomatiki-demo3.yaml": admission webhook "policy.sigstore.dev" denied
the request: validation failed: failed policy: cip-allow-tntezos-github-2: spec.template.spec.containers[0].image
registry2.local:5001/diplomatiki-demo3@sha256:febc7243e444633d09b412fad9f826046f6dc102386ee99577f221b69ac92101 signature keyless validation failed for authorit
y authority-0 for registry2.local:5001/diplomatiki-demo3@sha256:febc7243e444633d09b412fad9f826046f6dc102386ee99577f221b69ac92101: no signatures found
tax@tax:~$
```

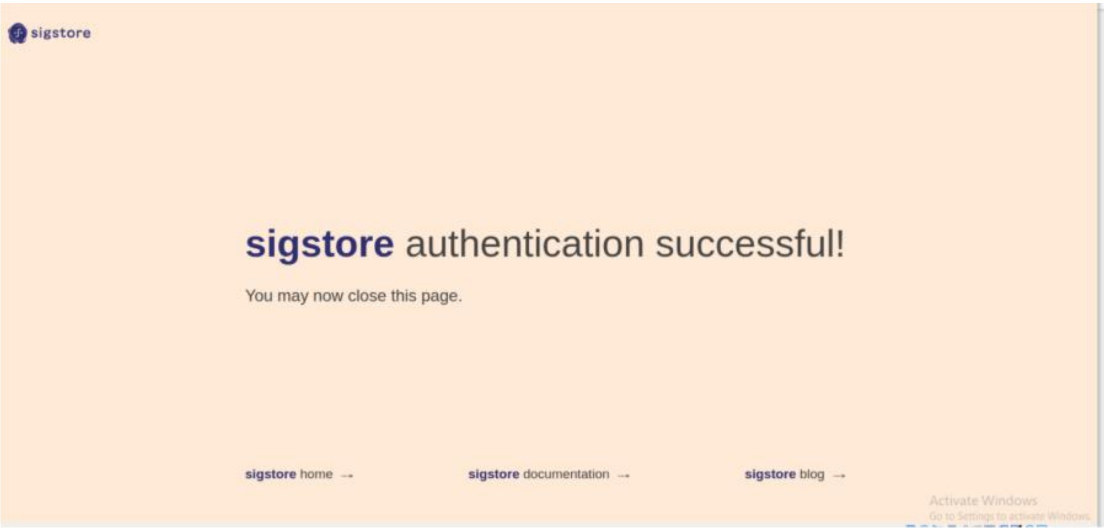
Δημιουργείται ένα deployment αρχείο (deploy-diplomatiki-demo3.yaml) που ορίζει ένα Pod το οποίο τρέχει το image diplomatiki-demo3 από το private registry. Το image χρησιμοποιεί το digest που βρέθηκε προηγουμένως. Το deployment τοποθετείται στο namespace signed-images όπου είναι ενεργός ο Sigstore Policy controller. Με την εντολή (kubectl apply –f........) ο policy controller ελέγχει το Image και απορρίπτει την δημιουργία του pod επειδή το Image δεν είναι υπογεγραμμένο.



```
tax@tax:~$ export COSIGN_EXPERIMENTAL=1
export IMAGE=registry2.local:5001/diplomatiki-demo3

cosign sign  "${IMAGE}@${DIGEST}"
Generating ephemeral keys...
Retrieving signed certificate...

        The sigstore service, hosted by sigstore a Series of LF Projects, LLC, is provided pursuant to the Hosted Project Tools Terms of Use, available at http
s://lfprojects.org/policies/hosted-project-tools-terms-of-use/.
        Note that if your submission includes personal data associated with this signed artifact, it will be part of an immutable record.
        This may include the email address associated with the account with which you authenticate your contractual Agreement.
        This information will be used for signing this artifact and will be stored in public transparency logs and cannot be removed later, and is subject to t
he Immutable Record notice at https://lfprojects.org/policies/hosted-project-tools-immutable-records/.

By typing 'y', you attest that (1) you are not submitting the personal data of any other person; and (2) you understand and agree to the statement and the Agre
ement terms at the URLs listed above.
Are you sure you would like to continue? [y/N] y
Your browser will now be opened to:
https://oauth2.sigstore.dev/auth/auth?access_type=online&client_id=sigstore&code_challenge=k_x5SEuDaDmwPabf7JT0JCwDVZ3KYIiHyrxJNygfpis&code_challenge_method=S2
56&nonce=34IswvN73QEufcETz6oi7S3Lyii&redirect_uri=http%3A%2F%2Flocalhost%3A35129%2Fauth%2Fcallback&response_type=code&scope=openid+email&state=34IswyGhJQtGNPp0
jEKRQvSSBl2
Successfully verified SCT...
WARNING: "registry2.local:5001/diplomatiki-demo3" appears to be a private repository, please confirm uploading to the transparency log at "https://rekor.sigsto
re.dev"
Are you sure you would like to continue? [y/N] y
tlog entry created with index: 622539491
Pushing signature to: registry2.local:5001/diplomatiki-demo3
tax@tax:~$
```

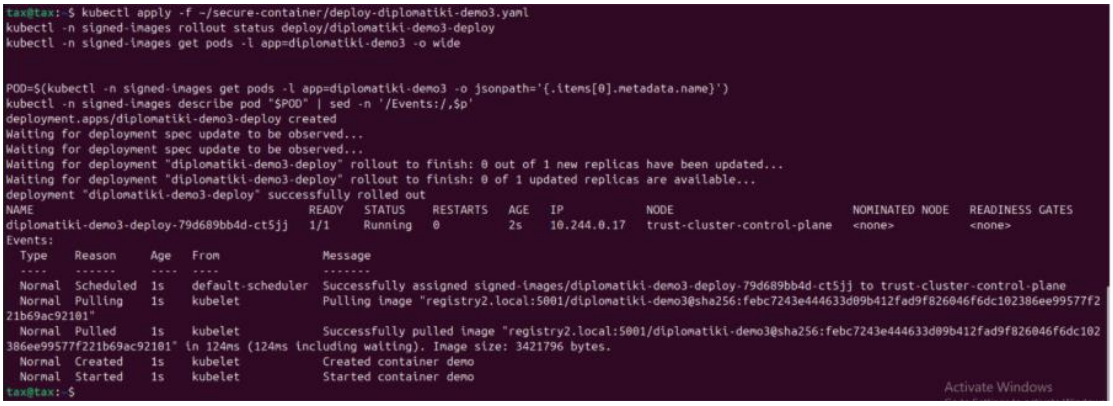
Γίνεται ενεργοποίηση του keyless signing mode του cosign και στην συνέχεια γίνεται αποθήκευση του ονόματος του image σε μια μεταβλητή για ευκολία. Τέλος, υπογράφεται το image. Κατά την διαδικασία υπογραφής πραγματοποιείται σύνδεση με την υπηρεσία Fulcio, όπου ζητείται επιβεβαίωση και ανοίγει ο browser ώστε να γίνει αυθεντικοποίηση μέσω OIDC. Εδώ ουσιαστικά εκδίδεται ένα σύντομης διάρκειας πιστοποιητικό, το αποθηκεύει στο Rekor και στέλνει την υπογραφή στο registry.





 https://doi.org/10.26219/heal.aueb.9571

Εκτελείται η εντολή verify image ώστε να επιβεβαιωθεί ότι το image όντως υπογράφθηκε και ότι είναι αποθηκευμένη στο registry και προέρχεται από την σωστή OIDC.



Πλέον παρατηρείται ότι το ίδιο yaml που είχε πριν απορριφθεί τρέχει. Εμφανίζεται η πρόοδος του deployment και στο τέλος δείχνει ότι το Pod δημιουργήθηκε και ξεκίνησε επιτυχώς. Στη συνέχεια φαίνονται τα Pod που δημιουργήθηκαν με τις πληροφορίες τους. Τέλος έγινε εμφάνιση των events που σχετίζονται με το pod, ώστε να φανούν οι ενέργειες που έκανε το kubernetes για να το δημιουργήσει και να το εκκινήσει. Αναλυτικά, το pod έγινε scheduled, έπειτα το image έγινε pull απο το registry με HTTPS και στην συνέχεια το pull επετράπη από τον Policy controller γιατί η υπογραφή ήταν έγκυρη. Τέλος, το container εκτελείται κανονικά.

https://doi.org/10.26219/heal.aueb.9571

4.4 Υλοποίηση watcher για αυτόματο έλεγχο / διαγραφή Images

watcher-config.yaml

```yaml
registry:
  base_url: "https://registry2.local:5001"
  username: "tax2"
  password: "MyStrongPass2"
  ca_cert: "/home/tax/registry2/certs/domain.crt"

scope:
  include_repos: []
  exclude_repos: []
  ignore_tags: ["latest"]
  max_repos_per_scan: 1000

cosign:
  experimental: true
  certificate_identity: "t.ntenezos@gmail.com"
  certificate_oidc_issuer: "https://github.com/login/oauth"
  extra_args: []

actions:
  on_fail: "delete"
  dry_run: false
  notify:
    slack_webhook: ""

runtime:
  poll_interval_sec: 30
  state_file: "/home/tax/image-watcher-state.json"
  request_timeout_sec: 10
  verbose: true
```

Αυτό είναι το αρχείο ρυθμίσεων του watcher script. Αρχικά, η ενότητα registry δηλώνει που βρίσκεται το registry, πως γίνεται log in και ποιο Cert να εμπιστεύεται. Έπειτα η ενότητα scope ορίζει σε ποια repositories θα κάνει έλεγχο ή θα αγνοεί και δίνεται ένας αριθμός για το πόσα repositories θα ελέγξει ανά κύκλο. Στην συνέχεια, η ενότητα cosign ρυθμίζει το cosign για να κάνει έλεγχο των υπογραφών, δηλαδή το χρειάζεται ο watcher ώστε να κάνει cosign verify για κάθε image και να απορρίπτει όσα δεν έχουν υπογραφή από το mail που δίνεται. Η ενότητα actions καθορίζει τι θα κάνει εάν βρει ο watcher μη υπογεγραμμένο image. Στην συγκεκριμένη περίπτωση έχει ζητηθεί να διαγράφει αυτόματα. Τέλος, στην ενότητα runtime ορίζεται κάθε πότε θα ελέγχει, κρατάει αρχείο κατάστασης, δηλαδή ποια Image έχει ήδη ελέγξει και μας δείχνει και τα logs.

```python
import json, os, time, subprocess
from pathlib import Path
from typing import Optional, List, Set, Dict
import requests, yaml

class RegistryClient:
    def __init__(self, base_url: str, username: Optional[str], password: Optional[str],
                 ca_cert, timeout: int = 10, verbose: bool = False):
        self.base_url = base_url.rstrip("/")
        self.auth = (username, password) if username and password else None
        self.verify = ca_cert if isinstance(ca_cert, str) and ca_cert else (True if ca_cert is True else False)
        self.timeout = timeout
        self.verbose = verbose
        self.s = requests.Session()
        if self.auth:
            self.s.auth = self.auth

    def _log(self, *a):
        if self.verbose:
            print("[registry]", *a)

    def get_catalog(self, n=1000) -> List[str]:
        url = f"{self.base_url}/v2/_catalog"
        self._log("GET", url)
        r = self.s.get(url, params={"n": str(n)}, verify=self.verify, timeout=self.timeout)
        r.raise_for_status()
        return r.json().get("repositories", [])

    def get_tags(self, repo: str) -> List[str]:
        url = f"{self.base_url}/v2/{repo}/tags/list"
        self._log("GET", url)
        r = self.s.get(url, verify=self.verify, timeout=self.timeout)
        if r.status_code == 404:
            return []
        r.raise_for_status()
        return r.json().get("tags", []) or []
```

Αρχικά, δημιουργείται η κλάση RegistryClient. Στήνεται HTTPS client προς ένα registry, με μηχανισμό αυθεντικοποίησης (username/password) και έλεγχο TLS (ca_cert). Η κλάση αυτή είναι υπεύθυνη για την σύνδεση και επικοινωνία με το docker registry. Μέσα της περιλαμβάνει μεθόδους, όπως την get_catalog, η οποία επιστρέφει τη λίστα όλων των repositories και την get_tag η οποία εμφανίζει όλα τα διαθέσιμα tags μιας εικόνας. Η _log είναι απλά για να υπάρχει μια καθαρή εικόνα των μηνυμάτων στο terminal όταν τρέξει το script.

https://doi.org/10.26219/heal.aueb.9571

```
    def get_manifest_digest(self, repo: str, ref: str) -> Optional[str]:
        url = f"{self.base_url}/v2/{repo}/manifests/{ref}"
        headers = {"Accept": "application/vnd.docker.distribution.manifest.v2+json"}
        self._log("HEAD", url)
        r = self.s.head(url, headers=headers, verify=self.verify, timeout=self.timeout)
        if r.status_code == 404:
            return None
        r.raise_for_status()
        return r.headers.get("Docker-Content-Digest")

    def delete_manifest(self, repo: str, digest: str) -> bool:
        url = f"{self.base_url}/v2/{repo}/manifests/{digest}"
        self._log("DELETE", url)
        r = self.s.delete(url, verify=self.verify, timeout=self.timeout)
        if r.status_code in (200, 201, 202):
            return True
        if r.status_code == 404:
            self._log("manifest not found", repo, digest)
            return False
        self._log("delete failed", r.status_code, r.text)
        return False

class StateStore:
    def __init__(self, path: str):
        self.path = Path(path)
        self.seen: Set[str] = set()
        if self.path.exists():
            try:
                self.seen = set(json.loads(self.path.read_text()).get("seen_digests", []))
            except Exception:
                self.seen = set()

    def save(self):
        self.path.parent.mkdir(parents=True, exist_ok=True)
        self.path.write_text(json.dumps({"seen_digests": sorted(self.seen)}, indent=2))
```

Περιλαμβάνει επίσης την get_manifest_digest η οποία μέσω https head αιτήματος, για κάθε tag παίρνει το αντίστοιχο digest κάθε εικόνας που βρίσκεται στο repository και την delete_manifest, η οποία επιτρέπει τη διαγραφή ενός image από το registry.

Στην συνέχεια, υπάρχει η κλάση StateStore η οποία είναι μια κλάση που έχει ως λειτουργία να θυμάται ποια images έχουν ήδη ελεγχθεί, ώστε να μην τα ξανά ελέγχει σε κάθε κύκλο. Η αποθήκευση γίνεται σε ένα json αρχείο. Η κλάση περιλαμβάνει μεθόδους όπως τη Load, η οποία φορτώνει τα δεδομένα από το αρχείο, τη save, που αποθηκεύει τις νέες εγγραφές. Επίσης, έχουμε τις has και add, οι οποίες ελέγχουν αν ένα digest υπάρχει ήδη ή προσθέτουν καινούργιο. Αυτό γίνεται ώστε να ελέγχει ο watcher μόνο τα νέα τροποποιημένα images στο registry.

```
    def has(self, key: str) -> bool: return key in self.seen
    def add(self, key: str): self.seen.add(key)

def notify(msg: str, slack_webhook: Optional[str] = None):
    print("[notify]", msg)
    if slack_webhook:
        try:
            requests.post(slack_webhook, json={"text": msg}, timeout=5)
        except Exception as e:
            print("[notify] slack error:", e)

def cosign_verify(image_ref: str, cert_id: Optional[str], cert_issuer: Optional[str],
                  experimental: bool, extra_args: Optional[List[str]] = None) -> bool:
    env = os.environ.copy()
    if experimental: env["COSIGN_EXPERIMENTAL"] = "1"
    cmd = ["cosign", "verify", image_ref]
    if cert_id: cmd += ["--certificate-identity", cert_id]
    if cert_issuer: cmd += ["--certificate-oidc-issuer", cert_issuer]
    if extra_args: cmd += extra_args
    try:
        p = subprocess.run(cmd, capture_output=True, text=True, env=env, timeout=90)
        ok = (p.returncode == 0)
        print(f"[cosign] {'OK' if ok else 'FAIL'} verify {image_ref}")
        if not ok:
            print(p.stdout.strip() or "")
            print(p.stderr.strip() or "")
        return ok
    except Exception as e:
        print("[cosign] error:", e); return False

def load_config(path: str) -> Dict:
    with open(path, "r") as f: return yaml.safe_load(f)
```

Έπειτα υπάρχουν οι βοηθητικές συναρτήσεις notify, cosign_verify και load_config. Η συνάρτηση notify χρησιμοποιείται για την εμφάνιση ενημερωτικών μηνυμάτων στο terminal. Στη συνέχεια, η συνάρτηση cosign verify είναι υπεύθυνη για την επαλήθευση της υπογραφής για κάθε image μέσω της εντολής cosign verify. Χρησιμοποιεί τα στοιχεία του repository, του digest και του πιστοποιητικού, με σκοπό να ελέγξει αν η υπογραφή της εικόνας είναι έγκυρη, και επιστρέφει το ανάλογο θετικό ή αρνητικό αποτέλεσμα. Με αυτό τον τρόπο ο watcher γνωρίζει ποιες εικόνες είναι υπογεγραμμένες και ποιες πρέπει να απορριφθούν. Τέλος η συνάρτηση load_config διαβάζει το αρχείο watcher-config.yaml και φορτώνει όλες τις παραμέτρους που καθορίζουν τη λειτουργία του script.

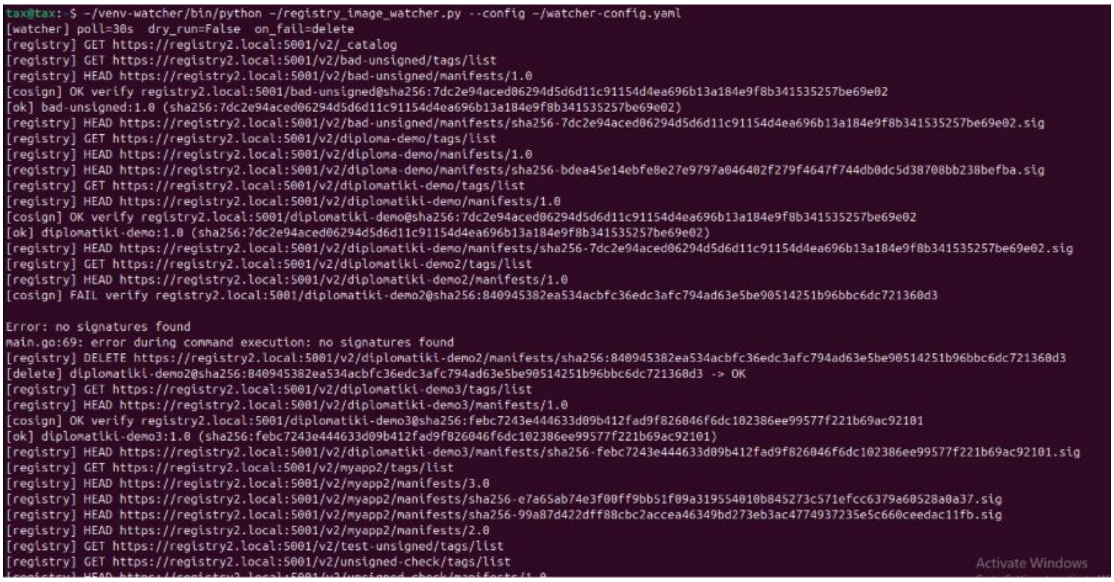
 https://doi.org/10.26219/heal.aueb.9571

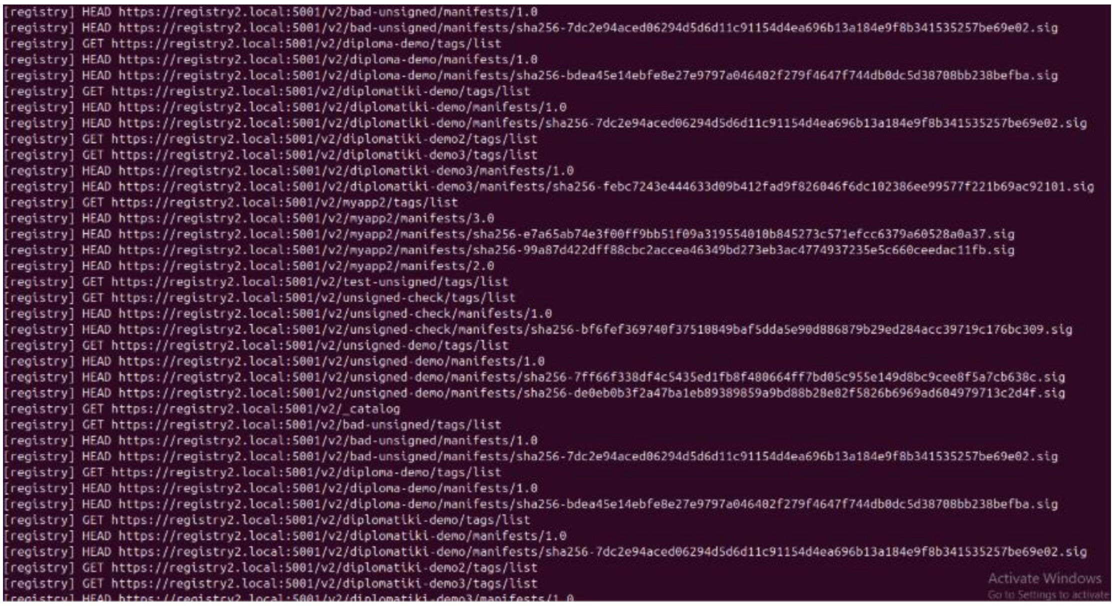
```python
def main():
    import argparse
    ap = argparse.ArgumentParser(description="Watch registry, verify images with cosign, act on failures.")
    ap.add_argument("--config", required=True)
    args = ap.parse_args()

    cfg = load_config(args.config)
    reg = cfg["registry"]; scope = cfg["scope"]; cos = cfg["cosign"]; act = cfg["actions"]; rt = cfg["runtime"]

    client = RegistryClient(reg["base_url"], reg.get("username"), reg.get("password"),
                            reg.get("ca_cert", True), rt.get("request_timeout_sec", 10), rt.get("verbose", False))

    include = scope.get("include_repos") or []
    exclude = set(scope.get("exclude_repos") or [])
    ignore_tags = set(scope.get("ignore_tags") or [])
    max_repos = scope.get("max_repos_per_scan", 1000)

    state = StateStore(rt.get("state_file", "./image-watcher-state.json"))
    poll = rt.get("poll_interval_sec", 60)

    on_fail = act.get("on_fail", "notify")
    dry_run = act.get("dry_run", True)
    slack = (act.get("notify") or {}).get("slack_webhook", "")

    cert_id = cos.get("certificate_identity")
    cert_issuer = cos.get("certificate_oidc_issuer")
    experimental = bool(cos.get("experimental", True))
    extra = cos.get("extra_args") or []

    print(f"[watcher] poll={poll}s  dry_run={dry_run}  on_fail={on_fail}")
```

Ακολουθεί η κύρια συνάρτηση main(), η οποία είναι το σημείο εκκίνησης του watcher. Στην αρχή διαβάζει το αρχείο watcher-config.yaml μέσω της συνάρτησης load_config. Από αυτό το αρχείο παίρνει πληροφορίες όπως η διεύθυνση του registry, τις επιλογές σύνδεσης (username,password, πιστοποιητικό tls), καθώς επίσης και τις ρυθμίσεις που αφορούν τη συμπεριφορά τους συστήματος σε περίπτωση αποτυχίας επαλήθευσης. Έπειτα δημιουργείται ένα αντικείμενο της κλάσης RegistryClient, το οποίο χρησιμοποιεί τις παραμέτρους του αρχείου για να συνδεθεί με το private registry. Επιπλέον, καθορίζονται τα repos που θα ελεγχθούν και τα tags που θα παραλειφθούν. Αρχικοποιείται επίσης η κλάση StateStore της οποίας η λειτουργία εξηγήθηκε προηγουμένως. Τέλος, φορτώνονται οι ρυθμίσεις που σχετίζονται με το cosign.

```
    while True:
        try:
            repos = include or client.get_catalog(n=max_repos)
            repos = [r for r in repos if r not in exclude]
            for repo in repos:
                tags = client.get_tags(repo)
                for tag in (tags or []):
                    if tag in ignore_tags: continue
                    digest = client.get_manifest_digest(repo, tag)
                    if not digest: continue
                    key = f"{repo}@{digest}"
                    if state.has(key): continue

                    host = client.base_url.split("://", 1)[1]
                    image_ref = f"{host}/{repo}@{digest}"

                    ok = cosign_verify(image_ref, cert_id, cert_issuer, experimental, extra)
                    if ok:
                        print(f"[ok] {repo}:{tag} ({digest})")
                    else:
                        msg = f"UNVERIFIED image: {repo}:{tag} ({digest})"
                        if on_fail in ("notify", "both", "none"):
                            notify(msg, slack if on_fail in ("notify", "both") else None)
                        if on_fail in ("delete", "both"):
                            if dry_run:
                                print(f"[delete][dry-run] would delete {repo}@{digest}")
                            else:
                                deleted = client.delete_manifest(repo, digest)
                                print(f"[delete] {repo}@{digest} -> {'OK' if deleted else 'FAIL'}")

                    state.add(key); state.save()
        except Exception as e:
            print("[watcher] loop error:", e)
        time.sleep(poll)

if __name__ == "__main__":
    main()
```

Στην τελευταία εικόνα, εκτελείται η κύρια διαδικασία ελέγχου του watcher. Για κάθε repository και tag πραγματοποιείται HTTPS HEAD αίτημα προς το registry, ώστε να εντοπιστεί το digest του image και να γίνει η επαλήθευση του μέσω cosign. Αν η υπογραφή είναι έγκυρη, το digest καταγράφεται στη StateStore, ενώ σε περίπτωση αποτυχίας (για παράδειγμα δεν επαληθεύτηκε ή δεν υπάρχει καθόλου) το image διαγράφεται από το registry. Έτσι, διασφαλίζεται ότι στο private registry παραμένουν μόνο υπογεγραμμένα images.





Κατά την εκτέλεση του watcher script, γίνεται συνεχής έλεγχος όλων των images που υπάρχουν στο registry2.local:5001. Το εργαλείο αυτό επικοινωνεί με το endpoint του registry μέσω HTTPS αιτημάτων και πραγματοποιεί έλεγχο σε κάθε Image digest με το cosign, ώστε να επαληθευτεί η εγκυρότητα της υπογραφής. Στις περιπτώσεις που εμφανίζει " [cosign] ok verify ", η υπογραφή του image έχει επιβεβαιωθεί και το αρχείο θεωρείται ασφαλές. Αντίθετα, σε περιπτώσεις που εμφανίζεται μήνυμα " fail verify " ή " signature not found ", σημαίνει ότι το Image αυτό δεν είναι υπογεγραμμένο ή έχει αλλοιωθεί και ο watcher το διαγράφει από την registry. Αυτό μπορούμε να το διακρίνουμε και από την πρώτη εικόνα όπου βρίσκει ένα μη υπογεγραμμένο image και το διαγράφει, ενώ στην δεύτερη εικόνα όταν ξανά ελέγχει δεν υπάρχει το μη υπογεγραμμένο image καθώς διαγράφηκε στον προηγούμενο κύκλο ελέγχου του.

Επίλογος

Η παρούσα διπλωματική εργασία ανέδειξε τη σημασία της ασφάλειας στην εφοδιαστική αλυσίδα λογισμικού και την ανάγκη εφαρμογής μηχανισμών ασφάλειας και επαλήθευσης κατά την διανομή containerized εφαρμογών. Η μεγάλη εξάρτηση των σύγχρονων εφαρμογών από εξωτερικά πακέτα, βιβλιοθήκες και υπηρεσίες τρίτων, αυξάνει τα πιθανά σημεία επίθεσης και δημιουργεί νέες αδυναμίες που μπορούν εύκολα να αξιοποιηθούν από κακόβουλους δράστες, όπως φάνηκε και από τα πρόσφατα περιστατικά επιθέσεων στο Npm.

Στο θεωρητικό μέρος παρουσιάστηκαν αναλυτικά οι βασικές συνιστώσες του Sigstore που το καθιστούν κατάλληλο για την ενίσχυση της εμπιστοσύνης, της ακεραιότητας και της διαφάνειας στην διανομή λογισμικού. Αναλύθηκε ο ρόλος του OpenID Connect στην ταυτοποίηση, η λειτουργία του Fulcio που εκδίδει προσωρινά πιστοποιητικά, η σημασία των transparency logs του Rekor για την καταγραφή και επαλήθευση των συμβάντων και τέλος ο μηχανισμός keyless signing που προσφέρει το cosign.

Στο πρακτικό μέρος υλοποιήθηκε ένα ασφαλές private registry με υποστήριξη TLS, το οποίο συνδέθηκε με το Kubernetes και προστατεύθηκε μέσω πολιτικών επαλήθευσης υπογραφών. Με αυτόν τον τρόπο διασφαλίστηκε ότι στο cluster μπορούν να εκτελεστούν μόνο images που έχουν ελεγχθεί και είναι αξιόπιστα. Επιπλέον, έγινε υλοποίηση ενός watcher script που παρακολουθεί διαρκώς το registry, ελέγχει την εγκυρότητα των υπογραφών για κάθε image και διαγράφει όσα δεν συμμορφώνονται με τις καθορισμένες πολιτικές.

Τέλος, η υλοποίηση που πραγματοποιήθηκε σε εργαστηριακό περιβάλλον, απέδειξε ότι οι ίδιες πρακτικές μπορούν να εφαρμοστούν και σε πιο σύνθετες, παραγωγικές υποδομές, διατηρώντας το ίδιο επίπεδο ασφαλείας. Επιπλέον, ο watcher μπορεί να εξελιχθεί ακόμη περισσότερο, είτε αποκτώντας πιο έξυπνες λειτουργίες, είτε να συνεργάζεται με άλλες υπηρεσίες στο cluster. Το σημαντικό όμως είναι ότι γίνεται πλέον ξεκάθαρο πως η διαφάνεια και η επαλήθευση δεν είναι κάτι “προαιρετικό”, αλλά βασική προϋπόθεση για να χτίζονται ασφαλείς, αξιόπιστες εφαρμογές.

https://doi.org/10.26219/heal.aueb.9571

Βιβλιογραφία

(CISA), C. a. (2025, September). *Widespread supply chain compromise impacting npm ecosystem*. Ανάκτηση από CISA Alerts: https://www.cisa.gov/news-events/alerts/2025/09/23/widespread-supply-chain-compromise-impacting-npm-ecosystem

(ICO), I. C. (2024). *Learning from the mistakes of others: Supply chain attacks.* Information Commissioners Office.

Andrey Pohiliko. *Registry-cli: Simple client for Docker Registry HTTP API v2*. Ανάκτηση από GitHub: https://github.com/andrey-pohilko/registry-cli/blob/master/registry.py

Chainguard Academy. *How to install Policy Controller*. Ανάκτηση από https://edu.chainguard.dev/open-source/sigstore/policy-controller/how-to-install-policy-controller/

Security Alliance. (2025). *NPM follow up*. Ανάκτηση από Security Alliance News: https://www.securityalliance.org/news/npm-follow-up

Chandramouli, K. &.-A. (2023). *Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines.* Gaithersburg: National Institute of Standards and Technology (NIST).

OpenSSF Foundation, O. S. (2023). *Running Sigstore as a managed service: A tour of sigstore public good instance*. Ανάκτηση από OpenSSF blog: https://openssf.org/blog/2023/10/03/running-sigstore-as-a-managed-service-a-tour-of-sigstores-public-good-instance/

OpenID Foundation. (2023). *The OpenID Handbook.* OpenID Foundation.

GitGurdian. *Supply chain attack:6 steps to harden your supply chain*. Ανάκτηση από GitGurdian Blog: https://blog.gitguardian.com/supply-chain-attack-6-steps-to-harden-your-supply-chain/

GitHub. (2025). *Our plan for a more secure npm supply chain*. Ανάκτηση από GitHub security Blog: https://github.blog/security/supply-chain-security/our-plan-for-a-more-secure-npm-supply-chain/

Red Hat. *What is a CI/CD pipeline*. Ανάκτηση από Red Hat: https://www.redhat.com/en/topics/devops/what-cicd-pipeline

Hostinger. *Kubernetes tutorial*. Ανάκτηση από Hostinger Tutorials: https://www.hostinger.com/tutorials/kubernetes-tutorial

Burnettk. *Delete_docker_registry_image*. Ανάκτηση από GitHub: https://github.com/burnettk/delete-docker-registry-image/blob/main/delete_docker_registry_image.py

Trend Micro. (2025). *NPM supply chain attack*. Ανάκτηση από Trend Micro Research: https://www.trendmicro.com/en_us/research/25/i/npm-supply-chain-attack.html

Kubernetes Project. *Kubernetes documendation home*. Ανάκτηση από Kubernetes.io: https://kubernetes.io/docs/home/

https://doi.org/10.26219/heal.aueb.9571

Sigstore. *Architecture Documentation*. Ανάκτηση από GitHub: https://github.com/sigstore/architecture-docs/blob/main/README.md

Sigstore. *Cosign*. Ανάκτηση από GitHub: https://github.com/sigstore/cosign/blob/main/README.md

sigstore. *How certificate issuing works*. Ανάκτηση από GitHub: https://github.com/sigstore/fulcio/blob/main/docs/how-certificate-issuing-works.md

sigstore. *Policy Controller overview*. Ανάκτηση από Sigstore Documendation: https://docs.sigstore.dev/policy-controller/overview/

Sonatype. (2025). *Ongoing npm software supply chain attack*. Ανάκτηση από Sonatype blog: https://www.sonatype.com/blog/ongoing-npm-software-supply-chain-attack-exposes-new-risks

Z. Newman, J. S.-A. (2022). Sigstore: Software Signing for Everyone. https://dl.acm.org/doi/pdf/10.1145/3548606.3560596.