Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εκτίμηση Ευπαθειών στο Hyperledger Fabric** **Hyperledger Fabric Vulnerability Assessment**
Ονοματεπώνυμο Φοιτητή	**Παναγιώτης Μάρκοβιτς**
Πατρώνυμο	**Δημήτριος**
Αριθμός Μητρώου	**ΜΠΣΠ/ 17044**
Επιβλέπων	**Κωνσταντίνος Πατσάκης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Ιούλιος 2020**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Κωνσταντίνος Πατσάκης Επίκουρος Καθηγητής	Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Ευάγγελος Σακκόπουλος Επίκουρος Καθηγητής

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω καταρχήν τον επίκουρο καθηγητή κ. Κωνσταντίνο Πατσάκη για την επίβλεψη αυτής της διπλωματικής εργασίας και για την εμπιστοσύνη που μου έδειξε, αναθέτοντάς μου ένα ιδιαίτερα ενδιαφέρον και απαιτητικό θέμα ερευνητικής μελέτης. Ακόμα, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στην οικογένεια και τους φίλους μου για την αδιάκοπη ενθάρρυνση και ψυχολογική στήριξη που μου παρείχαν καθ'όλη τη διάρκεια των σπουδών μου.

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η εκτίμηση ευπαθειών ασφάλειας στην καινούρια έκδοση (2.x) του Hyperledger Fabric, μιας δημοφιλούς πλατφόρμας για την ανάπτυξη εφαρμογών DLT (Distributed Ledger Technology), η οποία κυκλοφόρησε τον Ιανουάριο του 2020. Συγκεκριμένα, το Hyperledger Fabric αποτελεί λογισμικό ανοικτού κώδικα (open source software) το οποίο παρέχει την υποδομή για την κατασκευή private permissioned blockchain δικτύων και είναι σχεδιασμένο επί το πλείστον για χρήση στον τομέα των επιχειρήσεων.

Στα πλαίσια της έρευνάς μας, θα εξετάσουμε κάθε οντότητα σε ένα Hyperledger Fabric δίκτυο ξεχωριστά για να διαπιστώσουμε πιθανές ευπάθειες, με κύριο γνώμονα τα τρία βασικά κριτήρια ασφάλειας, την Εμπιστευτικότητα (Confidentiality), την Ακεραιότητα (Integrity) και την Διαθεσιμότητα (Availability). Παράλληλα, θα συγκρίνουμε τα ευρήματα μας με την ήδη υπάρχουσα βιβλιογραφία που αφορά προηγούμενες εκδόσεις του Hyperledger Fabric, ώστε να διαπιστώσουμε αν γνωστές ευπάθειες αυτών έχουν πλέον διορθωθεί.

Συνεπώς, η παρούσα εργασία αποτελεί μια προσπάθεια για ολοκληρωμένη και σφαιρική διερεύνηση των χαρακτηριστικών του Hyperledger Fabric από την σκοπιά της ασφάλειας, συνδυάζοντας, επεκτείνοντας και επιβεβαιώνοντας τις μέχρι τώρα δημοσιευμένες αναλύσεις και πειράματα με σκοπό την αξιολόγηση του συνολικού επιπέδου ασφάλειας στην καινούρια έκδοση.

Abstract

The scope of this thesis is the vulnerability assessment of the new version of Hyperledger Fabric (2.x), a popular platform for the development of DLT applications (Distributed Ledger Technology), which was release in January 2020. More specifically, Hyperledger Fabric is an open source software that provides the infrastructure for building private permissioned blockchain networks and is designed for use in enterprise contexts.

In the context of our research, we will inspect each entity of a Hyperledger Fabric network separatelt, in order to identify possible vulnerabilities, with our primary consideration being the three basic security principles, Confidentiality, Integrity and Availability. At the same time, we will compare our findings with those documented in related works that have been conducted regarding previous Hyperledger Fabric versions, in order to determine whether known vulnerabilities have been mitigated in the new version.

Consequently, this thesis constitutes an attempt to inspect the components of Hyperledger Fabric from a security point of view in an exhaustive manner, combining, extending and verifying the existing publications and experiments in the process, with the goal of evaluating the overall security level in the latest release.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1. Εισαγωγή

Πριν από περίπου μια δεκαετία, ένα άγνωστο άτομο ή ομάδα ατόμων με το ψευδώνυμο Satoshi Nakamoto, το οποίο δημιούργησε το Bitcoin[1], περιέγραψε πως η τεχνολογία blockchain[2], μια κατανεμημένη peer-to-peer συνδεδεμένη δομή, θα μπορούσε να χρησιμοποιηθεί για να λύσει το πρόβλημα της διατήρησης της σειράς των συναλλαγών και να αποφευχθεί το πρόβλημα double-spending[3]. Το Bitcoin ταξινομεί τις συναλλαγές και τις ομαδοποιεί σε μια δομή περιορισμένου μεγέθους που ονομάζεται *block*, αναθέτοντας τους ένα κοινό timestamp. Οι κόμβοι του δικτύου (*miners*) είναι υπεύθυνοι για τη σύνδεση μεταξύ των blocks σε χρονολογική σειρά[4], με κάθε block να περιέχει το hash του προηγούμενου block ώστε να δημιουργηθεί ένα blockchain[5]. Κατά συνέπεια, η blockchain δομή επιτυγχάνει την υλοποίηση ενός ισχυρού και ελέγξιμου μητρώου που καταγράφει όλες τις συναλλαγές.

Η ιδέα του blockchains προκάλεσε έντονες αναταραχές στον κόσμο των επιχειρήσεων όσον αφορά τις παραδοσιακές επιχειρησιακές διαδικασίες, καθώς οι εφαρμογές και οι συναλλαγές, οι οποίες μέχρι πρότινος απαιτούσαν μια κεντροποιημένη αρχιτεκτονική ή κάποιο έμπιστο διαμεσολαβητή για να επαληθεύει την ορθή λειτουργία τους, πλέον μπορούσαν να λειτουργήσουν με έναν αποκεντρωμένο τρόπο με το ίδιο επίπεδο βεβαιότητας. Τα εγγενή χαρακτηριστικά της αρχιτεκτονικής και του σχεδιασμού του blockchain παρέχουν ιδιότητες όπως διαφάνεια, ανθεκτικότητα, ελεγξιμότητα και ασφάλεια[6,7]. Μια δομή blockchain μπορεί να θεωρηθεί ως μια κατανεμημένη βάση δεδομένων[8] η οποία είναι οργανωμένη ως μια λίστα από ταξινομημένα blocks, όπου τα blocks που έχουν κατατεθεί είναι αμετάβλητα. Γίνεται λοιπόν εμφανές ότι μια τέτοια δομή είναι ιδανική στον τραπεζικό τομέα, όπου πολλές τράπεζες μπορούν να συνεργαστούν μέσω της χρήσης ενός κοινού blockchain, όπου θα καταθέτουν τις συναλλαγές των πελατών τους. Με αυτό τον τρόπο, πέρα από διαφάνεια, το blockchain ενισχύει την ελεγξιμότητα των συναλλαγών. Πολυάριθμες επιχειρήσεις επενδύουν στην εν λόγω τεχνολογία, καθώς διακρίνουν τη δυνατότητα να αποκεντροποιήσουν τις αρχιτεκτονικές τους και να ελαχιστοποιήσουν τα κόστη των συναλλαγών τους, ενώ ταυτόχρονα τις καθιστούν πιο ασφαλείς, διαφανείς και, σε ορισμένες περιπτώσεις, ταχύτερες.

Ο αριθμός από cryptocurrencies[9] τονίζει τη σημαντικότητα του Blockchain, ο οποίος ξεπερνάει τα 1900 και αυξάνεται ραγδαία[10]. Αυτός ο ρυθμός ανάπτυξης θα μπορούσε σύντομα να οδηγήσει σε προβλήματα συμβατότητας[11] λόγω της ανομοιογένειας[12] των cryptocurrency εφαρμογών[13,14]. Επιπροσθέτως, οι πιθανές εφαρμογές του blockchain συνεχώς επεκτείνονται, καθώς χρησιμοποιούνται σε άλλους τομείς πέρα από cryptocurrencies, με τα *Smart Contracts (SCs)* να παίζουν κεντρικό ρόλο. Τα SCs, όπως ορίστηκαν το 1994 από τον Szabo ως: *"ένα ηλεκτρονικό πρωτόκολλο συναλλαγών που εκτελεί τους όρους ενός συμβολαίου"* [15], μας επιτρέπουν να μεταφράσουμε ρήτρες συμβολαίων σε ενσωματώσιμο κώδικα, ελαχιστοποιώντας κατά συνέπεια συμμετοχή εξωτερικών μερών και λήψη ρίσκων. Επομένως, ένα SC αποτελεί ένα συμφωνητικό μεταξύ ομάδων που παρ' ότι δεν εμπιστεύονται ο ένας τον άλλο, εγγυάται ότι οι συμφωνημένοι όροι επιβάλλονται αυτόματα. Με άλλα λόγια, στα πλαίσια της τεχνολογίας blockchain, τα SCs είναι scripts που εκτελούνται με έναν αποκεντρωμένο τρόπο και αποθηκεύονται στο blockchain[7], χωρίς να εξαρτώνται από κάποια έμπιστη άρχη. Συγκεκριμένα, τα συστήματα που βασίζονται σε blockchain που υποστηρίζουν λειτουργικότητα SCs επιτρέπουν πολύ πιο περίπλοκες διεργασίες και αλληλεπιδράσεις, εγκαθιδρύοντας ένα νέο πρότυπο με πρακτικά αναρίθμητες εφαρμογές.

Ως αποτέλεσμα, η τεχνολογία blockchain γίνεται συνεχώς πιο επίκαιρη[17]. Σχεδόν 1000 (33%) από τα C-suite στελέχη δηλώνουν ότι αναλογίζονται τη χρήση ή ασχολούνται ήδη ενεργά με blockchains[18]. Οι ερευνητές και οι προγραμματιστές έχουν ήδη επίγνωση των δυνατοτήτων της νέας αυτής τεχνολογίας και εξερευνουν μια ποικιλία από πιθανές εφαρμογές σε ένα ευρύ φάσμα από τομείς[7,42]. Βάσει του κοινού στο οποίο απευθύνονται, μπορούμε να διαχωρίσουμε τα blockchains σε τρεις γενιές[17]: Το Blockchain 1.0 που περιλαμβάνει εφαρμογές που επιτρέπουν συναλλαγές με ψηφιακές συναλλαγές με cryptocurrencies, το Blockchain 2.0 που περιλαμβάνει SCs και ένα σύνολο από εφαρμογές που επεκτείνονται πέρα από συναλλαγές με cryptocurrency, και το Blockchain 3.0 που περιλαμβάνει εφαρμογές σε τομείς πέρα από τις δύο πρώτες εκδόσεις, όπως στη διακυβέρνηση, την υγεία, την επιστήμη και το IoT (Internet of Things).

Μολονότι τα blockchains θεωρούνται ιδιαίτερα ασφαλή και με εγγύηση προστασίας της ιδιωτικότητας εκ φύσεως, αυτή η εντύπωση δεν είναι καθολικά ακριβής, καθώς υπάρχουν αρκετές επιθέσεις που μπορούν να τα εκμεταλλευτούν με διάφορους τρόπους. Κάποιες από αυτές είναι αρκετά τεχνικές και ενδέχεται να μπορούν να πραγματοποιηθούν μόνο σε συγκεκριμένα blockchains. Οφείλουμε να λάβουμε υπόψη μας ότι τα blockchains είναι μια τεχνολογία που παρ' ότι αναπτύσσεται εδώ και μια δεκαετία, η πλειονότητα των χρηστών τους μόλις πρόσφατα άρχισαν να εμπλέκονται με αυτά και πολλές φορές τα αντιμετωπίζουν ως ένα "blackbox". Συνεπώς, θεωρούμε αναγκαίο βήμα την αποσαφήνιση των διάφορων πτυχών της ασφάλειας στις διάφορες τεχνολογίες blockchain, την ανάλυση των απειλών που αυτές αντιμετωπίζουν, τις επιθέσεις που μπορούν να πραγματοποιηθούν σε αυτές, το προσδόκιμο αντίκτυπο, καθώς και πιθανά αντίμετρα ασφαλείας.

Μια ιδιαίτερα δημοφιλής πλατφόρμα για την ανάπτυξη blockchain εφαρμογών στον τομέα των επιχειρήσεων τα τελευταία χρόνια είναι το Hyperledger Fabric, ένα λογισμικό ανοικτού κώδικα (open source software) το οποίο παρέχει την υποδομή για την κατασκευή private permissioned blockchain δικτύων. Το Hyperledger Fabric αποτελεί ένα από τα projects του Hyperledger υπό την αιγίδα του Linux Foundation[19,20]. Έχει χρησιμοποιηθεί σε πάνω από 400 πρωτότυπα, proofs-of-concept, και παραγωγικά συστήματα κατανεμημένων ledgers σε μεγάλο φάσμα τομέων και use cases. Το Fabric εισάγει μια καινοτόμα αρχιτεκτονική blockchain που αποσκοπεί σε ενισχυμένη ανθεκτικότητα, ευελιξια, επεκτασιμότητα και εμπιστευτικότητα. Σχεδιασμένο ως ένα κλιμακωτό και επεκτάσιμο permissioned blockchain γενικής χρήσης, είναι το πρώτο σύστημα blockchain που υποστηρίζει την εκτέλεση κατανεμημένων εφαρμογών γραμμένων σε standard γλώσσες προγραμματισμού, με τρόπο που τους επιτρέπει να εκτελούνται με συνέπεια κατά μήκος πολλών κόμβων, δίνοντας την εντύπωση της εκτέλεσης σε ένα ενιαίο κατανεμημένο blockchain υπολογιστή.

Το Hyperledger Fabric περιλαμβάνει αρθρωτά building blocks για κάθε ένα από τα δομικά του στοιχεία :

- Μια υπηρεσία ταξινόμησης (ordering service), η οποία μεταδίδει τις ενημερώσεις κατάστασης στους peers του δικτύου και διαμορφώνει τη συναίνεση όσον αφορά τη σειρά των συναλλαγών.
- Ένα πάροχο υπηρεσίας συνδρομής (membership service provider), ο οποίος είναι υπεύθυνος για τη συσχέτιση των peers με κρυπτογραφικές ταυτότητες. Διατηρεί την permissioned ιδιότητα του Fabric.
- Μια προαιρετική υπηρεσία peer-to-peer επικοινωνίας (gossip service) disseminates που διαδίδει τα blocks που παρέχει η υπηρεσία ταξινόμησης σε όλους τους peers.
- Τα Smart contracts στο Fabric εκτελούνται σε ένα περιβάλλον container για λόγους απομόνωσης. Μπορούν να συγγραφούν σε standard γλώσσες προγραμματισμούν αλλά δεν έχουν άμεση πρόσβαση στην κατάσταση του ledger.
- Κάθε peer διατηρεί τοπικά ένα αντίγραφο του ledger υπό τη μορφή ενός append-only blockchain και ενός snapshot της πιο πρόσφατης κατάστασης σε ένα key-value store.

Λόγω της συνεχούς και ενεργής ανάπτυξης και βελτίωσης του Hyperledger Fabric από τότε που κυκλοφόρησε, με τη λειτουργικότητα να αλλάζει συχνά για να προσαρμοστεί στις όλο και πιο εκτενείς ανάγκες της αγοράς και τα trends στην ανάπτυξη λογισμικού, είναι επόμενο να υπάρχουν αρκετές παρερμηνείες από τους χρήστες και τους προγραμματιστές σχετικά με διάφορες πτυχές του, καθώς και μια ασαφής εικόνα όσον αφορά το επίπεδο ασφάλειας, παρά την πραγματοποίηση αρκετών προσπαθειών αποτίμησης απειλών και πιθανών ευπαθειών. Για το λόγο αυτό, με αφορμή την κυκλοφορία της καινούριας έκδοσης (2.x) του Hyperledger Fabric τον Ιανουάριο του 2020, θα επιχειρήσουμε με την παρούσα διπλωματική εργασία να πραγματοποιήσουμε μια πλήρη εκτίμηση απειλών και ευπαθειών ασφάλειας.

Στα πλαίσια της έρευνάς μας, θα εξετάσουμε κάθε οντότητα σε ένα Hyperledger Fabric δίκτυο ξεχωριστά για να διαπιστώσουμε πιθανές ευπάθειες, με κύριο γνώμονα τα τρία βασικά κριτήρια ασφάλειας, την Εμπιστευτικότητα (Confidentiality), την Ακεραιότητα (Integrity) και την Διαθεσιμότητα (Availability). Παράλληλα, θα συγκρίνουμε τα ευρήματα μας με την ήδη

υπάρχουσα βιβλιογραφία που αφορά προηγούμενες εκδόσεις του Hyperledger Fabric, ώστε να διαπιστώσουμε αν γνωστές ευπάθειες αυτών έχουν πλέον διορθωθεί.

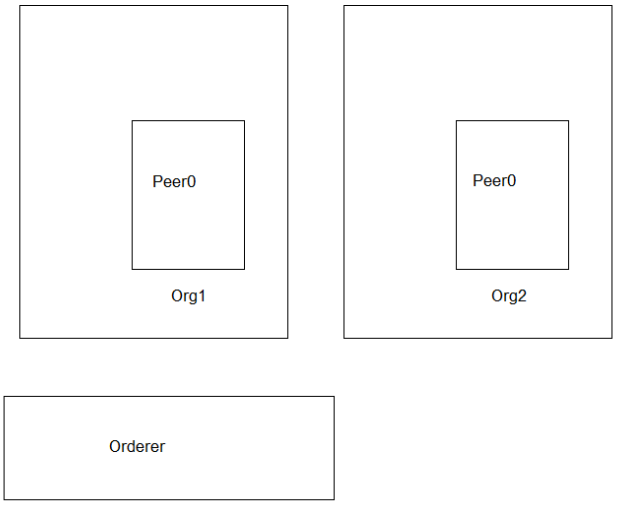
Συμπερασματικά, η παρούσα εργασία αποτελεί μια προσπάθεια για ολοκληρωμένη και σφαιρική διερεύνηση των χαρακτηριστικών του Hyperledger Fabric από την σκοπιά της ασφάλειας, συνδυάζοντας, επεκτείνοντας και επιβεβαιώνοντας τις μέχρι τώρα δημοσιευμένες αναλύσεις και πειράματα με σκοπό την αξιολόγηση του συνολικού επιπέδου ασφάλειας στην καινούρια έκδοση.

2. Περιβάλλον Εκτέλεσης

Όλα μας τα πειράματα εκτελέστηκαν σε ένα Ubuntu 18.04.5 LTS VM με kernel 4.15.0-117-generic, στο οποίο έχουν ανατεθεί 5 GB RAM και 6 πυρήνες από το φυσικό μηχάνημα με 8GB RAM, επεξεργαστή Intel ® Core™ i7-8750H CPU @ 2.20 GHz και Windows 10 build 1909 host λειτουργικό σύστημα.

Το blockchain network που χρησιμοποιούμε είναι το Test Network που παρέχουν τα fabric samples ως sample δίκτυο στην έκδοση 2.x του Hyperledger Fabric (2.2 στην περίπτωσή μας). Όλα τα scripts που παραθέτουμε ξεκινούν με αρχικό working directory το ***/home/pmarkovits/fabric-samples/test-network***[21,22], εκτός από τις περιπτώσεις όπου διευκρινίζεται διαφορετικά.

Το δίκτυό μας αποτελείται από ένα κανάλι mychannel με δύο οργανισμούς Org1 και Org2, με ένα peer κόμβο ο καθένας, και έναν orderer κόμβο τύπου Raft[23], όπως διαφαίνεται και από την παρακάτω εικόνα :



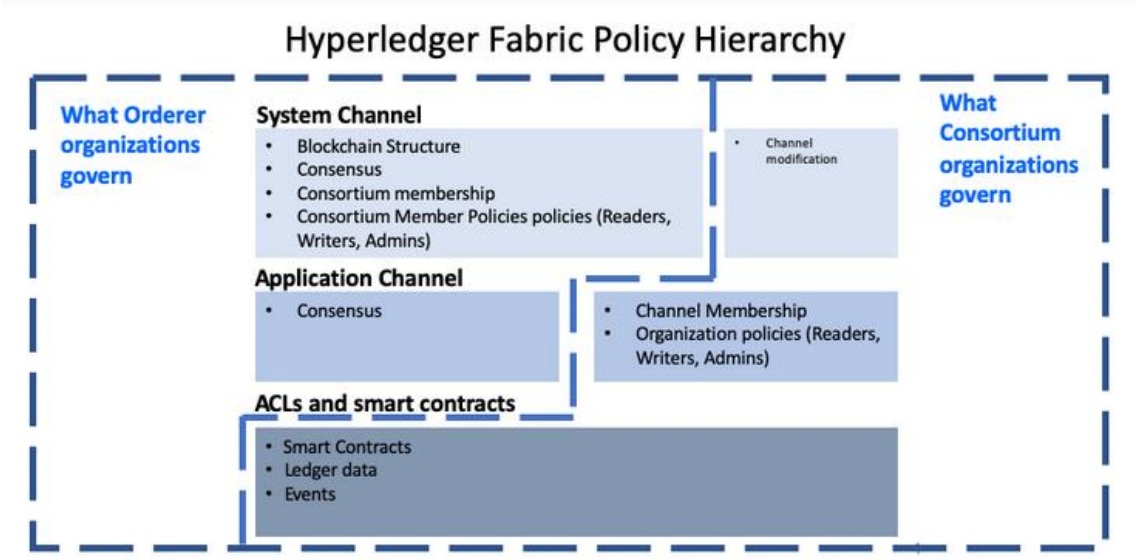
Εικόνα 2.1. Test Network, κανάλι mychannel

Κάθε οργανισμός χρησιμοποιεί για τη διαχείριση των εφαρμογών και των κόμβων του τους εξής ρόλους :

- Ένα client με όνομα admin στον αντίστοιχο Fabric CA Server για τη δημιουργία και διαχείριση άλλων ταυτοτήτων (default όταν δεν χρησιμοποιείται LDAP Server)
- Ένα peer με όνομα peer0
- Ένα client με όνομα user1 για τη διαχείριση της εφαρμογής του οργανισμού και την κλήση των chaincodes

- Τέλος, έναν admin με όνομα org1admin/org2admin αντίστοιχα

Στην πραγματικότητα, τα προνόμια κάθε ρόλου εντός του καναλίου καθορίζονται από τα policies που έχουν οριστεί στο channel configuration[41].



Εικόνα 2.2. Hyperledger Fabric Policy Hierarchy

3. Node & Transaction Authentication

3.1 Key Compromise

Το Fabric CA Module υποστηρίζει τη δημιουργία X.509 πιστοποιητικών και κλειδιών χρησιμοποιώντας τον Elliptic Curve Digital Signature Algorithm (ECDSA). Οι πιθανές επιλογές σχετικά με το μήκος του κλειδιού, την ελλειπτική καμπύλη που χρησιμοποιείται από τον ECDSA και τον αλγόριθμο υπογραφής παρατίθενται στον παρακάτω πίνακα :

Μήκος Κλειδιού	ASN1 OID	Αλγόριθμος Υπογραφής
256	prime256v1	ecdsa-with-SHA256
384	secp384r1	ecdsa-with-SHA384
521	secp521r1	ecdsa-with-SHA512

Πίνακας 1. Επιλογές για τη δημιουργία X.509 πιστοποιητικών και κλειδιών

Τόσο το μήκος των κλειδιών, όσο και οι επιλεγμένες ελλειπτικές καμπύλες και οι αλγόριθμοι κατακερματισμού που χρησιμοποιούνται περιλαμβάνονται στα πιο πρόσφατα specification documents από τους οργανισμούς NIST[24,25] και ENISA[26].

3.2 Transaction Replay Attack

Υπάρχουν πολλαπλοί μηχανισμοί που αποτρέπουν πιθανά transaction replay attacks. Αρχικά, το ID ενός transaction, στα πλαίσια του σχετικού channel, είναι μοναδικό και προκύπτει από το HEX-encoded SHA-256 hash του συνδυασμού της ταυτότητας του καταθέτη (συνήθως ένα X.509 πιστοποιητικό) και ενός τυχαίου nonce[27,28]. Ακόμα, δεν είναι εφικτό ένας κόμβος να καταθέσει ξανά το ίδιο transaction εφόσον έχει γίνει endorsed, διότι το endorsement περιλαμβάνει ένα read/write set με μοναδικό αναγνωριστικό κατάστασης για όλες τις μεταβλητές. Οι κόμβοι επικοινωνούν αποκλειστικά μέσω transactions, και το μόνο άλλο message flow στο channel είναι τα admin updates, τα οποία δε μπορούν να γίνουν replayed, καθώς βασίζονται στο τωρινό configuration, το οποίο μετά το εκάστοτε update είναι διαφορετικό και για την κατάθεση ενός νέου update απαιτείται η εκ νέου προσκόμιση του configuration.

Τέλος, υπάρχει προαιρετικά η δυνατότητα για χρήση ακόμα ενός τύπου μοναδικού αναγνωριστικού, του binding[29,30], για να αποτραπούν πιθανά replay attacks σε περιπτώσεις όπου σε ένα chaincode είναι απαραίτητη η αυθεντικοποίηση μιας ταυτότητας ανεξάρτητης από τον καταθέτη του transaction. Το binding προκύπτει από το HEX-encoded SHA-256 hash του συνδυασμού της ταυτότητας του καταθέτη, ενός τυχαίου nonce και του epoch timestamp τη στιγμή του chaincode invocation. Υπογράφοντας το συνδυασμό του transaction payload και του binding, ένας χρήστης μπορεί να παρέχει την απαραίτητη εξουσιοδότηση για να συμμετέχει ένας ενδιάμεσος σε ένα transaction, εκ μέρους του (ένα broker agency για παράδειγμα).

3.3 Source non-repudiation

Όλα τα είδη επικοινωνίας στο Hyperledger Fabric χρησιμοποιούν ECDSA, γεγονός που αποτρέπει non-repudiation attacks.

Σημείωση : Τα συμπεράσματα της ενότητας 2.4 προκύπτουν από τη σχετική βιβλιογραφία, όμως λόγω ελλιπούς documentation στην έκδοση 2.0 του Hyperledger Fabric, δεν καταφέραμε να απενεργοποιήσουμε το TLS στα Peer Nodes του Test Network που χρησιμοποιούμε, ώστε να τα επιβεβαιώσουμε πειραματικά.

3.4 Man-in-the-middle και Session Hijacking Attacks

Αν το TLS (v.1.2) στα Peer Nodes ή/και στα Orderer Nodes είναι απενεργοποιημένο, ένας επιτιθέμενος θα μπορούσε να παρεμβληθεί στην επικοινωνία μεταξύ τους, όμως λόγω των μηχανισμών που αναφέραμε στην ενότητα 2.2 και της χρήσης του ECDSA, δε θα ήταν εφικτή η τροποποίηση του traffic.

3.5 DDoS και DoS Attacks

Μια επίθεση DDoS ή DoS στα πλαίσια ενός Hyperledger Fabric channel είναι εφικτή είτε αν είναι απενεργοποιημένο το TLS client authentication στους κόμβους είτε αν ο επιτιθέμενος να αποτελεί μέλος του channel. Διακρίνουμε λοιπόν δύο περιπτώσεις, αν ο επιτιθέμενος δεν έχει στην κατοχή του ένα έγκυρο πιστοποιητικό που να μπορούν να αναγνωρίσουν οι Membership Service Providers των κόμβων αλλά το TLS client authentication είναι απενεργοποιημένο, η επίθεση μπορεί να πραγματοποιηθεί αλλά τα αντίστοιχα transaction requests θα απορριφθούν άμεσα λόγω χρήσης του ECDSA. Στην περίπτωση που ο επιτιθέμενος κατέχει μια έγκυρη ταυτότητα, μπορεί να πραγματοποιήσει πλήρως μια τέτοια επίθεση. Φυσικά, όπως θα αναλύσουμε σε επόμενες ενότητες, οι εκδόσεις > 2.0 του Hyperledger Fabric υποστηρίζουν πλήρως certificate revocation, ή ακόμα και identity revocation, το οποίο ακυρώνει όλα τα certificates μιας οντότητας και αποκλείει τη δυνατότητα έκδοσης νέων certificates από αυτή. Συνεπώς, με αυτό τον τρόπο μπορεί να αποτραπεί σχετικά άμεσα μια επίθεση DoS.

Στη συνέχεια, θα εξετάσουμε το αντίκτυπο μιας επίθεσης DoS, βασιζόμενοι στη μεθοδολογία παλαιότερης δημοσίευσης[31] πάνω σε προηγούμενη έκδοση του Hyperledger Fabric.

Το δίκτυο στο οποίο θα διεξάγουμε τις μετρήσεις μας είναι το Test Network, με το deployed chaincode το fabcar. Έχουμε δηλαδή δύο οργανισμούς, 1 και 2, που ο καθένας διαχειρίζεται ένα κόμβο peer. Ακόμα, έχουμε ένα κόμβο orderer τύπου Raft, και μια Couch DB state database που αποθηκεύει την κατάσταση των clients. Θα χρησιμοποιήσουμε το benchmark tool Hyperledger Caliper[32,33] έκδοση 0.3.2, ώστε να μετρήσουμε την απόδοση του δικτύου σε κανονικές συνθήκες και όταν δέχεται επίθεση DoS στους κόμβους endorsers. Επιπροσθέτως, θα χρησιμοποιήσουμε το εργαλείο hping320, έκδοση 3.0.0-alpha-2, για να εφαρμόσουμε DoS στους κόμβους στόχους, στέλνοντας συνεχώς SYN packets.

Συγκεκριμένα, θα πραγματοποιήσουμε 4 benchmark rounds των 1000 transactions δημιουργίας Car asset με κλιμακούμενο ρυθμό κατάθεσης, ξεκινώντας από 50 transactions το δευτερόλεπτο και αυξάνοντας το ρυθμό κατά 50 μέχρι να φτάσουμε 200 transactions το δευτερόλεπτο στο τελευταίο round. Εξετάζουμε την απόδοση του δικτύου σε τρία σενάρια : Α) υπό κανονικές συνθήκες, Β) με τον peer του οργανισμού να δέχεται επίθεση DoS και Γ) τους peers και των δύο οργανισμών να δέχονται επίθεση DoS. Θα επαναλάβουμε την εκτέλεση συνολικά 5 φορές για να υπολογίσουμε το μέσο όρο (μ) των παραμέτρων απόδοσης που παρέχει το Caliper, το throughput και το latency. Το throughput εκφράζει τον αριθμό από transactions που ανανέωσαν το ledger ανά δευτερόλεπτο (transactions per second – tps), ενώ το latency εκφράζει το απαιτούμενο χρόνο για να περάσει ένα transaction από το proposal phase στο commit phase. Τέλος, πριν από κάθε εκτέλεση επανεκκινούμε το δίκτυο, έτσι ώστε να εξασφαλίσουμε ότι το μήκος του ledger δεν επηρεάζει τις μετρήσεις μας.

Θα θέλαμε να αναφέρουμε σε αυτό το σημείο ότι ο αρχικός μας σχεδιασμός για το πείραμα ήταν να εκτελεστούν 20 benchmarks rounds, ξεκινώντας από 10 tps και αυξάνοντας το ρυθμό κατά 10 μέχρι να φτάσουμε τα 200 tps. Ωστόσο, η εκτέλεση του πειράματος προκαλούσε σταθερά crash των peer docker containers κατά τον έβδομο ή όγδοο γύρο, γεγονός που μπορεί να αποτελεί ένδειξη για πιθανή ευπάθεια. Για το λόγο αυτό, τροποποιήσαμε το σχεδιασμό μας όπως περιγράφεται παραπάνω.

Αρχικά, εγκαθιστούμε, ρυθμίζουμε και εκτελούμε το Caliper με τις παρακάτω εντολές :

```
# setup network, channel and chaincode
cd ../fabcar
./startFabric.sh
cd ../..

# download and setup caliper
git clone https://github.com/hyperledger/caliper-benchmarks.git
cd caliper-benchmarks
git checkout v0.3.2
npm init -y
npm install --only=prod @hyperledger/caliper-cli@0.3.2

cd /home/pmarkovits/caliper-benchmarks/networks/fabric/config_solo_raft
./generate.sh
```

```
cd –

# run caliper in fabric version 2.x compatible mode
export CALIPER_FABRIC_SKIPCREATECHANNEL_MYCHANNEL=true
npx caliper launch master \
--caliper-bind-sut fabric:2.1.0\
--caliper-cwd . \
--caliper-flow-only-test \
--caliper-fabric-gateway-usegateway \
--caliper-benchconfig benchmarks/samples/fabric/fabcar/config.yaml \
--caliper-networkconfig networks/fabric/test-network.yaml
```

όπου ***config.yaml*** είναι το παρακάτω benchmark scenario :

```
---
test:
  workers:
    type: local
    number: 1
  rounds:
  - label: create50
    txNumber: 1000
    rateControl:
      type: fixed-rate
      opts:
        tps: 50
    callback: benchmarks/samples/fabric/fabcar/createCar.js
  - label: create100
    txNumber: 1000
    rateControl:
      type: fixed-rate
      opts:
        tps: 100
    callback: benchmarks/samples/fabric/fabcar/createCar.js
  - label: create150
```

```
    txNumber: 1000
    rateControl:
      type: fixed-rate
      opts:
        tps: 150
    callback: benchmarks/samples/fabric/fabcar/createCar.js
  - label: create200
    txNumber: 1000
    rateControl:
      type: fixed-rate
      opts:
        tps: 200
    callback: benchmarks/samples/fabric/fabcar/createCar.js
monitor:
  type:
  - docker
  docker:
    name:
    - all
  interval: 1
```

και ***test-network.yaml*** το εξής δίκτυο :

```
name: Fabric
version: "1.0"
mutual-tls: false

caliper:
    blockchain: fabric
    command:
        start: export FABRIC_VERSION=2.1.0;export FABRIC_CA_VERSION=1.4.4;sleep 3s

info:
    Version: 2.1.0
    Size: 2 Orgs with 1 Peer
```

```
    Orderer: Raft
    Distribution: Single Host
    StateDB: CouchDB

clients:
    client0.org1.example.com:
        client:
            organization: Org1
            credentialStore:
                path: /tmp/hfc-kvs/org1
                cryptoStore:
                    path: /tmp/hfc-cvs/org1
            clientPrivateKey:
                path: /home/pmarkovits/fabric-samples/test-network/organizations/
             peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/keystore/
             bd97be4df58934fc7ff461f31bd47a43b850879326fb7606207a96653c705688_sk
            clientSignedCert:
                path: /home/pmarkovits/fabric-samples/test-network/
             organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/
             msp/signcerts/cert.pem

    client0.org2.example.com:
        client:
            organization: Org2
            credentialStore:
                path: /tmp/hfc-kvs/org2
                cryptoStore:
                    path: /tmp/hfc-cvs/org2
            clientPrivateKey:
                path: /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/
             org2.example.com/users/User1@org2.example.com/msp/keystore/
             8b60a6f03a50280a2bdc6e054af3658adb522b22e487bafd42f27940ae83d371_sk
            clientSignedCert:
                path: /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/
                org2.example.com/users/User1@org2.example.com/msp/signcerts/cert.pem
```

```
channels:
  mychannel:
    created: true
    definition:
      capabilities: []
      consortium: 'SampleConsortium'
      msps: ['Org1MSP', 'Org2MSP']
      version: 0
    orderers:
    - orderer.example.com
    peers:
      peer0.org1.example.com:
        eventSource: true
      peer0.org2.example.com:
        eventSource: true

    chaincodes:
    - id: fabcar
      version: v1
      language: golang
      path: /home/pmarkovits/fabric-samples/fabcar/go

organizations:
  Org1:
    mspid: Org1MSP
    peers:
    - peer0.org1.example.com
    certificateAuthorities:
    - ca-org1
    adminPrivateKey:
      path: /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/
       org1.example.com/users/Admin@org1.example.com/msp/keystore/
       6ae734f41b6290d9558449e9c62bcfb6d8a544da37e1148e4be1622d8a4a1f47_sk
    signedCert:
      path: /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/
       org1.example.com/users/Admin@org1.example.com/msp/signcerts/cert.pem
```

```
   Org2:
      mspid: Org2MSP
      peers:
      - peer0.org2.example.com
      certificateAuthorities:
      - ca-org2
      adminPrivateKey:
         path: /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/
          org2.example.com/users/Admin@org2.example.com/msp/keystore/
          dc30fba46cce5b888f91731fc5d5286c9a73bea320be76f667714843e9243778_sk
      signedCert:
         path: /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/
          org2.example.com/users/Admin@org2.example.com/msp/signcerts/cert.pem

orderers:
   orderer.example.com:
      url: grpcs://localhost:7050
      grpcOptions:
         ssl-target-name-override: orderer.example.com
      tlsCACerts:
         path: /home/pmarkovits/fabric-samples/test-network/organizations/
          ordererOrganizations/example.com/orderers/orderer.example.com/msp/
          tlscacerts/tlsca.example.com-cert.pem
peers:
   peer0.org1.example.com:
      url: grpcs://localhost:7051
      grpcOptions:
         ssl-target-name-override: peer0.org1.example.com
         grpc.keepalive_time_ms: 600000
      tlsCACerts:
         path: /home/pmarkovits/fabric-samples/test-network/organizations/
          peerOrganizations/org1.example.com/msp/tlscacerts/ca.crt

   peer0.org2.example.com:
      url: grpcs://localhost:9051
      grpcOptions:
         ssl-target-name-override: peer0.org2.example.com
```

```
        grpc.keepalive_time_ms: 600000
    tlsCACerts:
      path: /home/pmarkovits/fabric-samples/test-network/organizations/
       peerOrganizations/org2.example.com/msp/tlscacerts/ca.crt

certificateAuthorities:
  ca-org1:
    url: https://localhost:7054
    httpOptions:
      verify: false
    tlsCACerts:
      path: /home/pmarkovits/fabric-samples/test-network/organizations/
       peerOrganizations/org1.example.com/tlsca/tlsca.org1.example.com-cert.pem
    registrar:
    - enrollId: admin
      enrollSecret: adminpw

  ca-org2:
    url: https://localhost:8054
    httpOptions:
      verify: false
    tlsCACerts:
      path: /home/pmarkovits/fabric-samples/test-network/organizations/
       peerOrganizations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem
    registrar:
    - enrollId: admin
      enrollSecret: adminpw
```

Σημείωση : Κάθε φορά που κάνουμε επανεκκίνηση στο δίκτυο, τα ιδιωτικά κλειδιά στο παραπάνω αρχείο πρέπει να ανανεώνονται, καθώς παράγονται επιτόπου και συνεπώς είναι διαφορετικά σε κάθε εκτέλεση. Το ίδιο ισχύει και για τις IP των docker containers στο blockchain δίκτυο.

Στη συνέχεια, έχοντας ταυτοποιήσει τις IP των peer docker containers στο blockchain δίκτυο με τις εντολές ***docker exec -it <peer node name> sh*** και ***ifconfig***, επαναλαμβάνουμε την εκτέλεση του caliper benchmark για το σενάριο **B**, κατά τη διάρκεια του οποίου εκτελείται ως background process η εντολή

```
sudo hping3 <organisation 1 peer node IP> --rand-source -S -p 7051 –flood &
```

Έπειτα, επαναλαμβάνουμε μια τελευταία φορά την εκτέλεση του benchmark για το σενάριο Γ, κατά τη διάρκεια του οποίου εκτελούνται ως background processes οι εντολές

```
sudo hping3 <organisation 1 peer node IP> --rand-source -S -p 7051 –flood &
sudo hping3 <organisation 2 peer node IP> --rand-source -S -p 9051 –flood &
```

Μετά την επανάληψη του πειράματος 5 φορές για όλα τα σενάρια, ο μέσος όρος των παραμέτρων απόδοσης ήταν ο ακόλουθος :

Σενάριο Α

Round Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
create50	997	3	49.9	3.63	0.09	0.67	48.56
create100	993	7	99.6	8.20	0.38	5.20	56.00
create150	999	1	149.9	11.02	0.17	5.62	60.16
create200	1000	0	193.0	11.36	0.35	5.76	65.10

Σενάριο Β

Round Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
create50	996	4	49.9	5.94	0.31	2.50	40.66
create100	999	1	98.5	11.49	0.26	6.00	49.42
create150	999	1	143.1	16.01	0.25	7.76	48,36
create200	1000	0	174.9	16.14	0.29	7.58	53.12

Σενάριο Γ

Round Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
create50	997	3	50,1	4.65	0.12	1.28	45.38

create100	999	1	99.9	9.23	0.35	5.00	53.80
create150	1000	0	147.4	12.57	0.25	6.48	56,18
create200	999	1	190,0	15.89	0.39	7.43	56.00

Παρατηρούμε ότι το latency και το throughput μειώνονται σημαντικά όταν οι peer κόμβοι δέχονται επίθεση DoS, χωρίς όμως μεγάλες διαφορές ανάμεσα στα σενάρια επίθεσης. Μια πιθανή εξήγηση είναι το γεγονός ότι οι δύο κόμβοι ανήκουν σε διαφορετικό οργανισμό, ενώ αν είχαμε στην τοπολογία μας παραπάνω από ένα κόμβο ανά οργανισμό, το αντίκτυπο της επίθεσης θα ήταν μεγαλύτερο.

4. Fabric CA Module

Το Hyperledger Fabric διαθέτει ένα module που διαχειρίζεται την έκδοση πιστοποιητικών, το Fabric CA module[10,11]. Οποιαδήποτε επικοινωνία με το CA module διεξάγεται μέσω REST API, με τις εξής διαθέσιμες επιλογές όσον αφορά τη χρήση TLS (v.1.2) :

- Απενεργοποιημένο
- Client authentication
- Server Authentication
- Mutual authentication

4.1 Password Brute Force Attack

Για να καταχωρηθεί μια νέα ταυτότητα σε έναν Fabric CA server, απαιτείται η παροχή των credentials μέσω του προαναφερθέντος REST API. Συνεπώς, αν είναι απενεργοποιημένο το TLS authentication, ένας επιτιθέμενος που δεν ανήκει στο δίκτυο μπορεί να εφαρμόσει brute force attack για να ανακαλύψει τα credentials μιας ταυτότητας και να γίνει enroll με αυτά.

Για να επιβεβαιώσουμε τα παραπάνω, θα επιχειρήσουμε να εφαρμόσουμε brute force attack ώστε να γίνουμε enrolled ως ο administrator ρόλος του οργανισμού 1 στο κανάλι mychannel του test network, του sample network που παρέχει το Hyperledger Fabric v.2 ως tutorial. Σκοπός μας είναι να διαπιστώσουμε σε ποιες περιπτώσεις και μέχρι ποιο όριο επιτρέπονται συνεχείς προσπάθειες authentication χωρίς να μπορεί να πιστοποιηθεί το αν ανήκουμε στο δίκτυο. Αξίζει να αναφέρουμε στο σημείο αυτό ότι το Fabric CA server CLI επιτρέπει μέχρι 10 ανεπιτυχείς προσπάθειες ανά identity, πριν αυτό να κλειδωθεί. Η τιμή αυτή μπορεί να ρυθμιστεί με την παράμετρο **--cfg.identities.passwordattempts** κατά την εκκίνηση του server. Ωστόσο, στο σχετικό documentation δεν διευκρινίζεται αν μετά την πάροδο κάποιου χρονικού διαστήματος, το συγκεκριμένο identity ξεκλειδώνεται για άλλες 10 προσπάθειες. Συνεπώς, για να εξετάσουμε όλες τις παραμέτρους, θα δοκιμάσουμε 1 σωστό κωδικό adminpw και 11 λανθασμένους σύντομους κωδικούς adminpw0-11, θα περιμένουμε 10 λεπτά και θα ξαναδοκιμάσουμε, σε σενάρια με αυξανόμενο επίπεδο ασφάλειας. Το script που θα χρησιμοποιήσουμε είναι το παρακάτω :

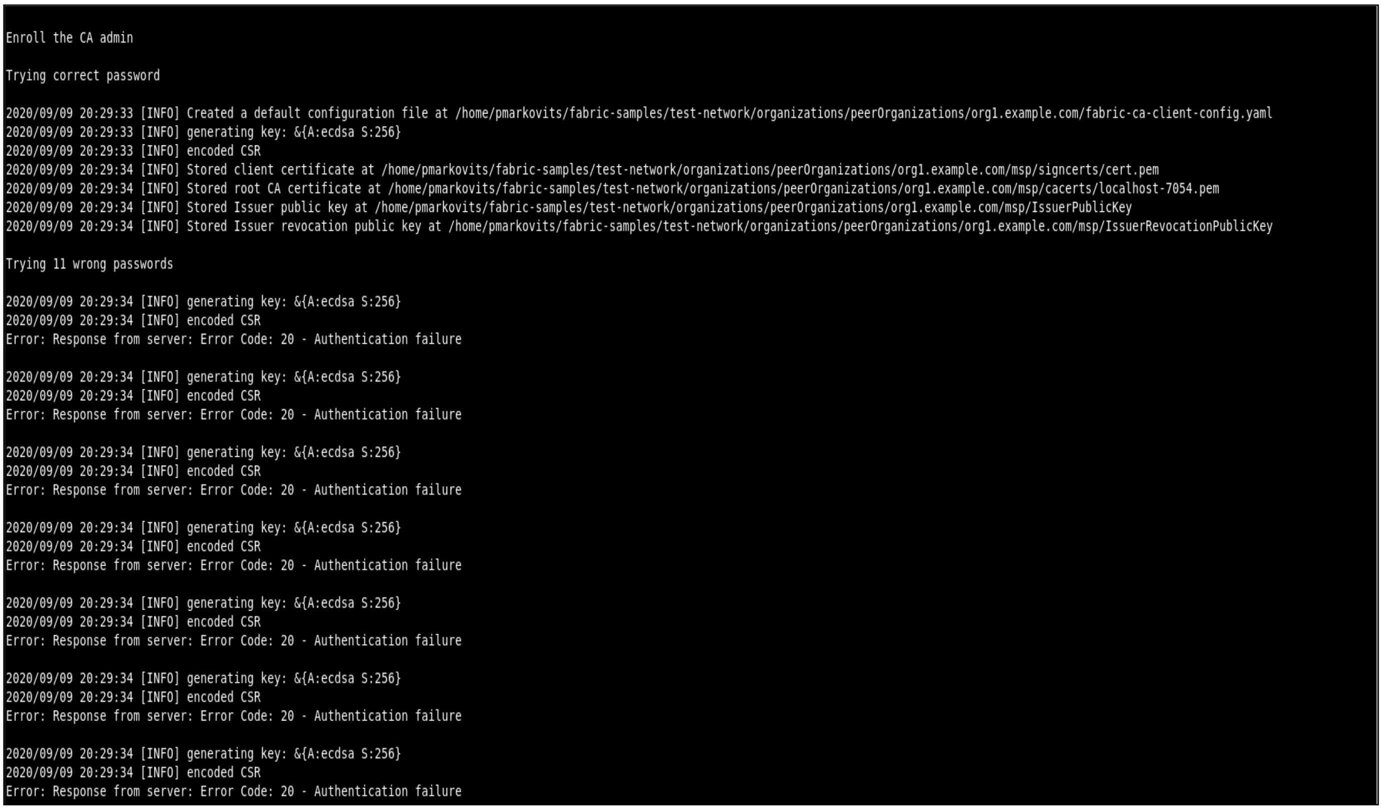
```
IMAGE_TAG="latest" docker-compose -f docker/docker-compose-ca.yaml up -d 2>&1
sleep 10
echo
echo "Enroll the CA admin"
echo
mkdir -p organizations/peerOrganizations/org1.example.com/
export FABRIC_CA_CLIENT_HOME=${PWD}/organizations/
peerOrganizations/org1.example.com/
export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=${PWD}/configtx
export VERBOSE=false

echo "Trying correct password"
echo
fabric-ca-client enroll -u http://admin:adminpw@localhost:7054
echo
echo "Trying 11 wrong passwords"
echo
for i in {1..11}; do
        fabric-ca-client enroll -u http://admin:adminpw$i@localhost:7054
done

echo "Waiting for 10 minutes"
sleep 10m

echo
echo “Trying again”
echo
for I in {0..15}; do
        fabric-ca-client enroll -u http://admin:adminpw$i@localhost:7054
done
```

```
# shut down network
./network.sh "down"
```

Script 4.2. fabric-ca_brute-force.sh , Server TLS disabled

Αρχικά, επιχειρούμε την επίθεση με το TLS του Fabric CA server απενεργοποιημένο.

Όπως ήταν αναμενόμενο, μπορούμε να γίνουμε enroll χωρίς να πιστοποιήσουμε την ταυτότητα του client που χρησιμοποιούμε, καθώς και να επιχειρήσουμε ένα brute-force attack μέχρι 10 προσπάθειες, όπως διαφαίνεται στις παρακάτω εικόνες :



```
Enroll the CA admin

Trying correct password

2020/09/09 20:29:33 [INFO] Created a default configuration file at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2020/09/09 20:29:33 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:33 [INFO] encoded CSR
2020/09/09 20:29:34 [INFO] Stored client certificate at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/signcerts/cert.pem
2020/09/09 20:29:34 [INFO] Stored root CA certificate at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/cacerts/localhost-7054.pem
2020/09/09 20:29:34 [INFO] Stored Issuer public key at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/IssuerPublicKey
2020/09/09 20:29:34 [INFO] Stored Issuer revocation public key at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/IssuerRevocationPublicKey

Trying 11 wrong passwords

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure
```

Εικόνα 4.3. Επιτυχές enrollment, εκκίνηση brute-force attack

```
2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:34 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:34 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:35 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:35 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 20:29:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:29:35 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

Waiting for 10 minutes

Trying again

2020/09/09 20:39:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:39:35 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 20:39:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:39:35 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 20:39:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:39:35 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 20:39:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:39:35 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 20:39:35 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 20:39:35 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached
```

Εικόνα 4.2. Lock out μετά τις 10 προσπάθειες, χωρίς ανανέωση μετά από 10 λεπτά

Στη συνέχεια, κάνουμε πλήρη εκκίνηση στο δίκτυό μας ώστε να παραχθεί το απαραίτητο κρυπτογραφικό υλικό, έχοντας πρώτα ενεργοποιήσει το TLS Server Authentication για τον Fabric CA server του οργανισμού 1. Τροποποιούμε κατάλληλα το script μας ώστε κάθε αίτηση για enrollment να παρέχει το πιστοποιητικό του Root CA στο οποίο απευθύνεται.

```
#IMAGE_TAG="latest" docker-compose -f docker/docker-compose-ca.yaml up -d 2>&1
 ./network.sh "up" “createChannel” "-ca"
echo
sleep 10
echo
echo "Enroll the CA admin"
echo
#mkdir -p organizations/peerOrganizations/org1.example.com/
export FABRIC_CA_CLIENT_HOME=${PWD}/organizations/
peerOrganizations/org1.example.com/
export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=${PWD}/configtx
export VERBOSE=false

echo "Trying correct password"
```

```
echo

fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca-org1
--tls.certfiles ${PWD}/organizations/fabric-ca/org1/tls-cert.pem
echo
echo "Trying 11 wrong passwords"
echo
for i in {1..11}; do
        fabric-ca-client enroll -u https://admin:adminpw$i@localhost:7054
        --caname ca-org1
        --tls.certfiles ${PWD}/organizations/fabric-ca/org1/tls-cert.pem
done

echo "Waiting for 10 minutes"
sleep 10m

echo
echo "Trying again"
echo
for i in {0..15}; do
        fabric-ca-client enroll -u https://admin:adminpw$i@localhost:7054
        --caname ca-org1
        --tls.certfiles ${PWD}/organizations/fabric-ca/org1/tls-cert.pem
done

# shut down network
./network.sh "down"
```

Script 4.2. fabric-ca_brute-force.sh , Server TLS enabled

Ομοίως με το πρώτο σενάριο, μπορούμε να γίνουμε enroll χωρίς να πιστοποιήσουμε την ταυτότητα του client που χρησιμοποιούμε, εφόσον το Client Authentication είναι απενεργοποιημένο, καθώς και να επιχειρήσουμε ένα brute-force attack μέχρι 10 προσπάθειες.

```
########### Create Org1 Identities #####################
########################################################

Enroll the CA admin

Trying correct password

2020/09/09 23:02:54 [INFO] TLS Enabled
2020/09/09 23:02:54 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:54 [INFO] encoded CSR
2020/09/09 23:02:54 [INFO] Stored client certificate at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/signcerts/cert.pem
2020/09/09 23:02:54 [INFO] Stored root CA certificate at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2020/09/09 23:02:54 [INFO] Stored Issuer public key at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/IssuerPublicKey
2020/09/09 23:02:54 [INFO] Stored Issuer revocation public key at /home/pmarkovits/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/msp/IssuerRevocationPublicKey

Trying 11 wrong passwords

2020/09/09 23:02:54 [INFO] TLS Enabled
2020/09/09 23:02:54 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:54 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 23:02:54 [INFO] TLS Enabled
2020/09/09 23:02:54 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:54 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 23:02:54 [INFO] TLS Enabled
2020/09/09 23:02:54 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:54 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 23:02:55 [INFO] TLS Enabled
2020/09/09 23:02:55 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:55 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 23:02:55 [INFO] TLS Enabled
2020/09/09 23:02:55 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:55 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 23:02:55 [INFO] TLS Enabled
2020/09/09 23:02:55 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:55 [INFO] encoded CSR
Error: Response from server: Error Code: 20 - Authentication failure
```

Εικόνα 4.3. Επιτυχές enrollment, εκκίνηση brute-force attack, Server TLS Authentication enabled

```
Error: Response from server: Error Code: 20 - Authentication failure

2020/09/09 23:02:55 [INFO] TLS Enabled
2020/09/09 23:02:55 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:02:55 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

Waiting for 10 minutes

Trying again

2020/09/09 23:12:55 [INFO] TLS Enabled
2020/09/09 23:12:55 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:12:55 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 23:12:55 [INFO] TLS Enabled
2020/09/09 23:12:55 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:12:55 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 23:12:56 [INFO] TLS Enabled
2020/09/09 23:12:56 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:12:56 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 23:12:56 [INFO] TLS Enabled
2020/09/09 23:12:56 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:12:56 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 23:12:56 [INFO] TLS Enabled
2020/09/09 23:12:56 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:12:56 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 23:12:56 [INFO] TLS Enabled
2020/09/09 23:12:56 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:12:56 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached

2020/09/09 23:12:56 [INFO] TLS Enabled
2020/09/09 23:12:56 [INFO] generating key: &{A:ecdsa S:256}
2020/09/09 23:12:56 [INFO] encoded CSR
Error: Response from server: Error Code: 73 - Incorrect password entered 10 times, max incorrect password limit of 10 reached
```

Εικόνα 4.4. Lock out μετά τις 10 προσπάθειες, χωρίς ανανέωση μετά από 10 λεπτά

Τέλος, κάνουμε επανεκκίνηση στο δίκτυό μας, έχοντας πρώτα ενεργοποιήσει και το TLS Client Authentication για τον Fabric CA server του οργανισμού 1. Οποιαδήποτε απόπειρα authentication πλέον αποτυγχάνει αυτόματα, αφού δεν παρέχουμε το απαραίτητο πιστοποιητικό για το authentication του Fabric CA Client μας.

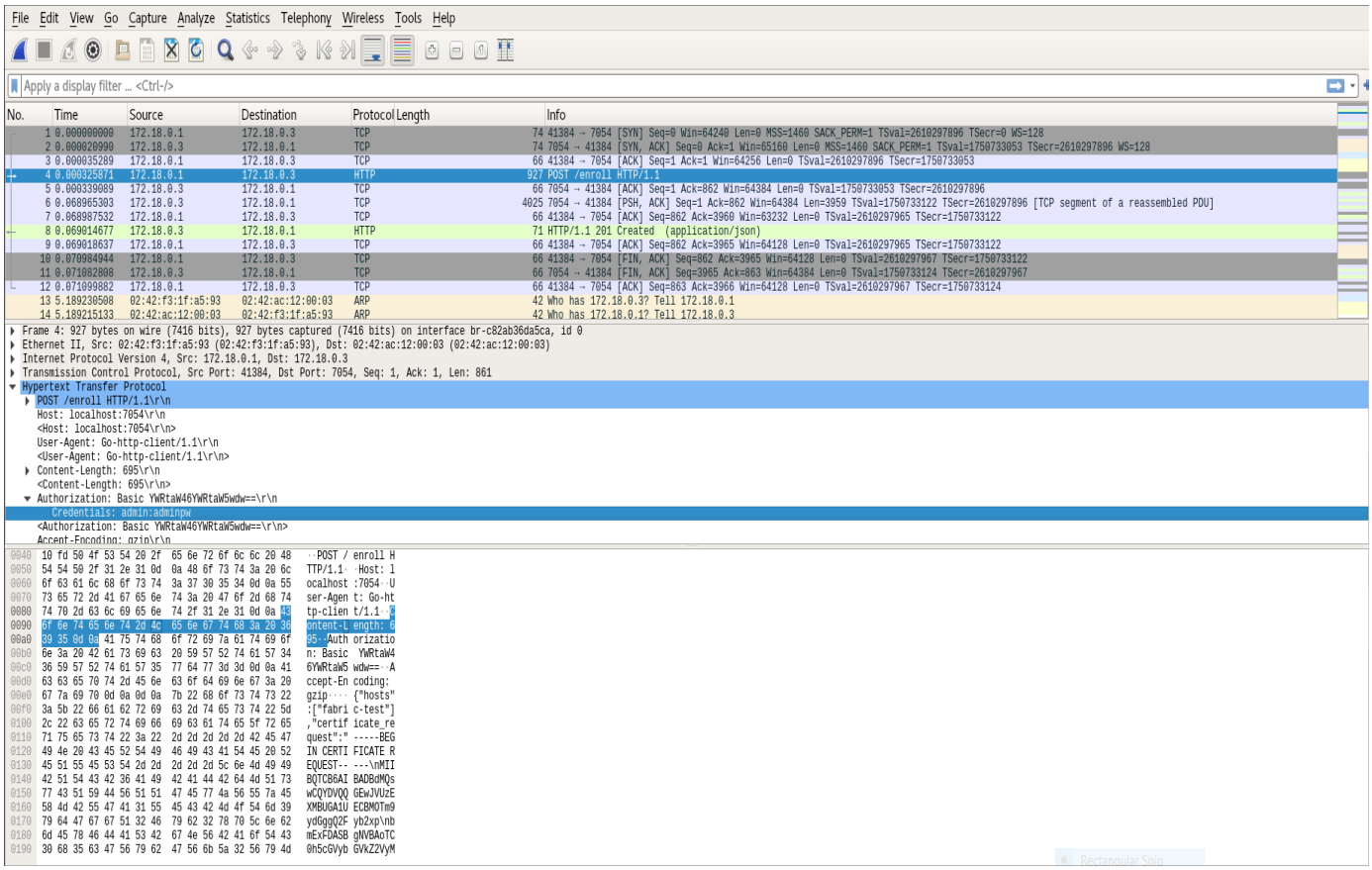
Εικόνα 4.5. Απόρριψη των authentication attempts λόγω του Client TLS Authentication policy

4.2 Dictionary Attack

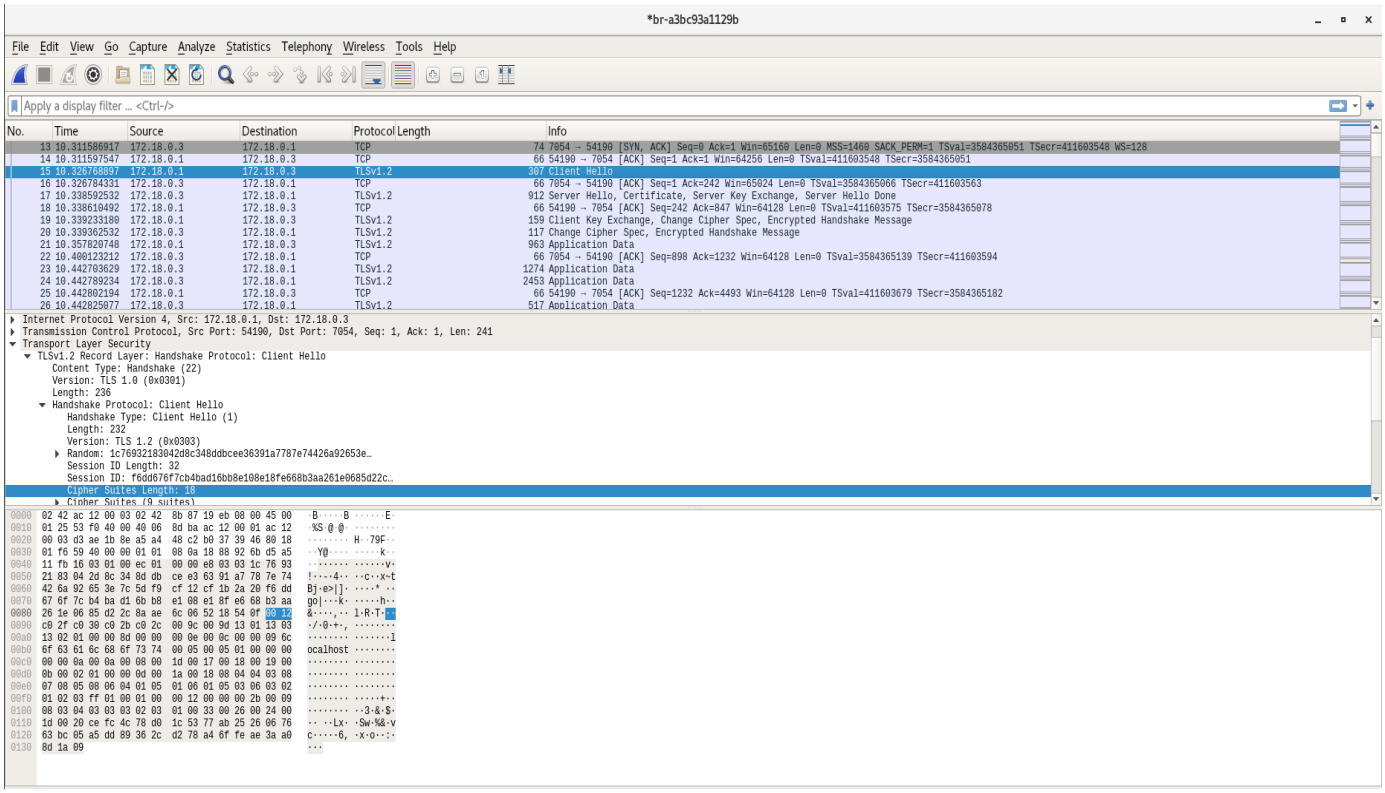
Ομοίως με το brute force attack, το Fabric CA module είναι ευάλωτο σε dictionary attacks αν δεν είναι υποχρεωτικό το client authentication.

4.3 Sniffing Attack

Ένας επιτιθέμενος μπορεί να αποκτήσει το username και password ενός identity μέσω της κίνησης του δικτύου, αν το TLS είναι απενεργοποιημένο, όπως φαίνεται και στις παρακάτω εικόνες:



Εικόνα 3.6. Credentials Sniffing με απενεργοποιημένο το Server και Client TLS Authentication



Εικόνα 4.7. Credentials Sniffing με ενεργοποιημένο το Server TLS Authentication

Παρατηρούμε ότι όταν το Server TLS Authentication είναι ενεργοποιημένο, ολόκληρη η επικοινωνία μεταξύ client και server είναι κρυπτογραφημένη και κατά συνέπεια δε γνωρίζουμε σε ποιο request και περιέχονται τα identity credentials, ούτε μπορούμε να τα ανακτήσουμε. Για το sniffing demonstration χρησιμοποιήσαμε την εφαρμογή *Wireshark*[38] *έκδοση 3.2.5*.

4.4 TLS Keys Compromise

Στα πλαίσια της έρευνάς μας, δεν μπορέσαμε να βρούμε πληροφορίες σχετικά με τις πιθανές επιλογές για την παραγωγή των TLS κλειδιών. Εικάζουμε λοιπόν ότι η πιθανότητα υποκλοπής τους καθορίζεται από τις επιλογές του εκάστοτε χρήστη.

4.5 Replay Attack

Το Fabric CA είναι ευάλωτο σε replay attack, στην περίπτωση που το TLS είναι απενεργοποιημένο, καθώς η επικοινωνία με αυτό πραγματοποιείται REST API το οποίο δεν είναι κρυπτογραφημένο.

4.6 Man-in-the-middle και Session Hijacking Attacks

Αν το TLS είναι απενεργοποιημένο, είναι εφικτό να πραγματοποιηθεί MITM ή session hijacking attack, καθώς η επικοινωνία βασίζεται σε REST API, που είναι stateless και μη κρυπτογραφημένα.

4.7 DDoS και DoS Attacks

Όπως και στην περίπτωση των peer κόμβων του δικτύου *(ενότητα 2.5)*, μια επίθεση DDoS ή DoS είναι εφικτή μόνο αν το TLS client Authentication είναι απενεργοποιημένο ή ο επιτιθέμενος αποτελεί μέλος του δικτύου. Σε αντίθεση όμως με τους peer κόμβους, ανάλογα με το πόσοι CA κόμβοι χρησιμοποιούνται ανά οργανισμό και αν ο κόμβος που δέχεται επίθεση είναι Root CA, ενδέχεται να είναι δύσκολο να γίνουν revoke τα certificates και το identity που χρησιμοποιεί ο επιτιθέμενος.

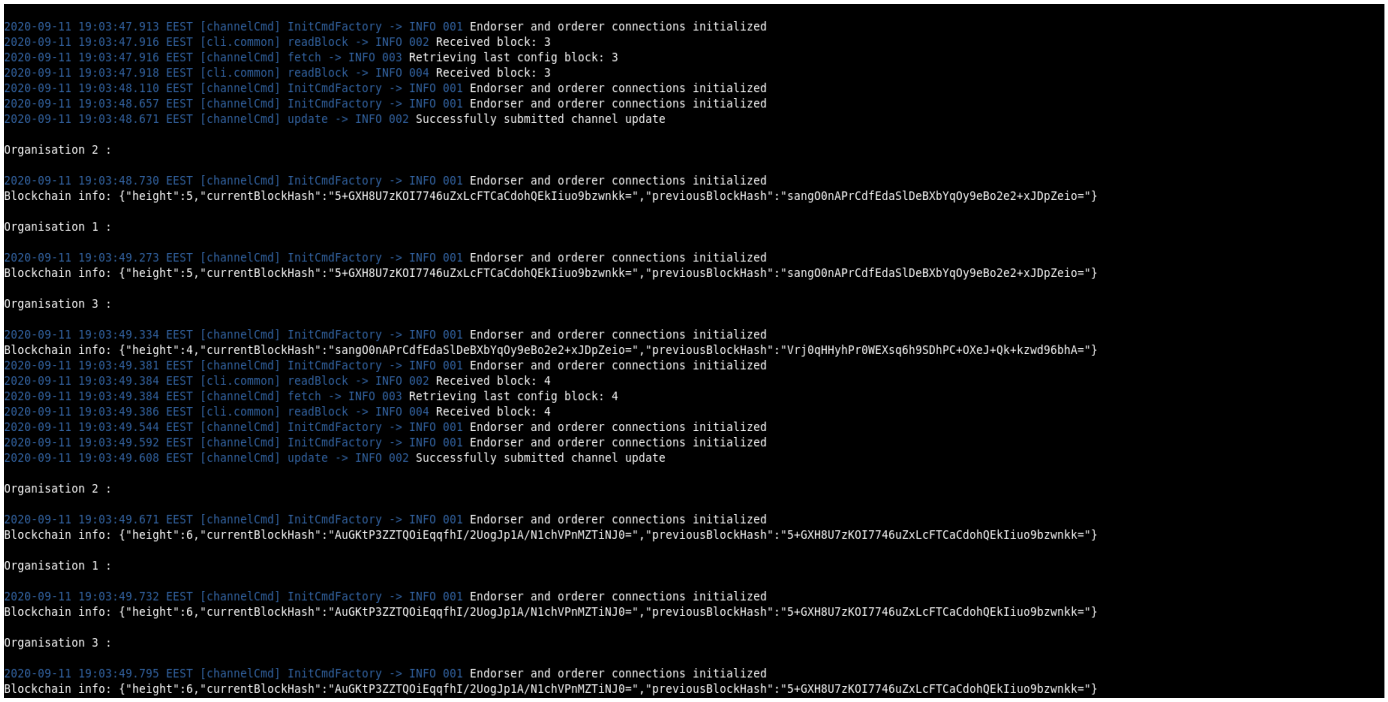
5. Channel Authentication

Το κανάλι δε μπορεί να ταυτοποιηθεί με κάποιο τρόπο απέναντι στους χρήστες, και κατά συνέπεια δεν υπάρχει εγγύηση ότι το κανάλι ή το δίκτυο δεν έχει τροποποιηθεί. Μια πιθανή επίθεση, η οποία είχε εντοπιστεί στα πλαίσια ενός security assessment report[39] που αφορά προηγούμενη έκδοση του Hyperledger Fabric, είναι η παρακάτω :

- Η πλειονότητα των οργανισμών σε ένα κανάλι αποφασίζει την από-προσάρτηση ενός επιλεγμένου οργανισμού-θύμα
- Στη συνέχεια, τροποποιείται το network/channel configuration για να προστεθεί ένα νέο root ή ενδιάμεσο CA ή να αλλάξει κάποιο policy
- Ο οργανισμός θύμα προστίθεται ξανά στο κανάλι και υιοθετεί το νέο configuration χωρίς να ειδοποιείται για την αλλαγή.

Ακολουθώντας την παρακάτω μεθοδολογία, διαπιστώνουμε ότι η παραπάνω επίθεση είναι ακόμα εφικτή στην έκδοση 2.2 :

- Δημιουργούμε ένα κανάλι mychannel με τους 3 default οργανισμούς του test network
- Ο MSP Admin του καναλιού, που ανήκει στον οργανισμό 1, ανακτά το channel configuration
- Το channel configuration αποκωδικοποιείται με τη χρήση του configtxlator binary
- Ο οργανισμός 3 αφαιρείται από το channel configuration, και αλλάζει το endorsement policy του καναλιού ώστε να εξαρτάται αποκλειστικά από τα peer nodes του οργανισμού 1
- Το αρχικό και το τροποποιημένο configuration κωδικοποιείται με το configtxlator
- Το κωδικοποιημένο περιεχόμενο περικλείεται σε ένα wrapper και συμπεριλαμβάνεται σε ένα config transaction
- Ο MSP Admin υπογράφει το transaction block και ο peer του οργανισμού 2 το συνυπογράφει και στη συνέχεια καταθέτει το transaction, που εφόσον ήδη είναι endorsed από την πλειονότητα των οργανισμών του καναλιού, γίνεται δεκτό και οι αλλαγές τίθενται σε ισχύ
- Η διαδικασία επαναλαμβάνεται για να προσθέσει ξανά τον οργανισμό 3 στο κανάλι



Εικόνα 5.1. Channel block information after each transaction

Παρατηρούμε ότι μετά το πρώτο transaction, το ύψος του blockchain και το αντίστοιχο block του οργανισμού 3 δεν έχει ανανεωθεί. Μετά το δεύτερο transaction και την εκ νέου προσθήκη του οργανισμού στο κανάλι, το ledger του ενημερώνεται έτσι ώστε να είναι συνεπές με αυτά των υπόλοιπων οργανισμών. Κατά τη διάρκεια της διαδικασίας, ο peer του οργανισμού 3 χάνει τη σύνδεση του με το κανάλι και κατά την επανασύνδεση υιοθετεί το νέο endorsement policy χωρίς να γνωρίζει ότι είχε αφαιρεθεί προσωρινά.

```
# set up test network and channel
./network.sh "up" "createChannel" "-ca"

#set up third organisation
cd addOrg3/
./addOrg3.sh "up" "-c" "mychannel" "-ca"
cd ..

# use organisation 1 who has an admin role in the channel
export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/
organizations/peerOrganizations/org1.example.com/peers/
```

```
peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/
organizations/peerOrganizations/org1.example.com/users/
Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

# retrieve latest configuration block
peer channel fetch config config_block.pb
-o localhost:7050 --ordererTLSHostnameOverride orderer.example.com
--tls –cafile ${PWD}/organizations/ordererOrganizations/example.com/
orderers/orderer.example.com/msp/
tlscacerts/tlsca.example.com-cert.pem -c mychannel

configtxlator proto_decode --input config_block.pb --type common.Block
| jq .data.data[0].payload.data.config > config.json

# save & format organisation 3 MSP configuration for later use
cat config.json | jq '.channel_group.groups.Application.groups.Org3MSP' > file.json.tmp
&& cp file.json.tmp missing_config.json && rm file.json.tmp

sed -i '1s/^/"Org3MSP": /' missing_config.json

cat missing_config.json | tr -d "\n\r " > file.json.tmp
&& cp file.json.tmp  missing_config.json && rm file.json.tmp

# save & format organisation 1 endorsement policy for later use
cat config.json
| jq '.channel_group.groups.Application.groups.Org1MSP.policies.Endorsement'
> file.json.tmp && cp file.json.tmp endorsement_config.json && rm file.json.tmp

sed -i '1s/^/"Endorsement": /' endorsement_config.json

cat endorsement_config.json | tr -d "\n\r " > file.json.tmp
```

```
&& cp file.json.tmp endorsement_config.json && rm file.json.tmp

# remove organisation 3 MSP configuration and channel endorsement policy
jq 'del(.channel_group.groups.Application.groups.Org3MSP)' config.json
| jq 'del(.channel_group.groups.Application.policies.Endorsement)'
> modified_config.json

# append organisation 1 endorsement policy as channel endorsement policy
#so that only organisation 1 can endorse transactions in the channel
cat modified_config.json | jq '.channel_group.groups.Application.policies
+= {'$(cat endorsement_config.json)'}' >file.json.tmp
&& cp file.json.tmp modified_config.json
&& rm file.json.tmp

# encode both config.json and modified_config.json files to PB format,
# compute the difference of the two files, saved as update.pb
configtxlator proto_encode --input config.json --type common.Config --output config.pb

configtxlator proto_encode --input modified_config.json --type common.Config
--output modified_config.pb

configtxlator compute_update --channel_id mychannel --original config.pb
--updated modified_config.pb --output update.pb

# add back envelope for update.pb : first decoded into update.json,
# then envelope is added, the result is encoded back to update_in envelope.pb
configtxlator proto_decode --input update.pb --type common.ConfigUpdate
| jq . > update.json

echo
'{"payload":{"header":{"channel_header":{"channel_id":"mychannel","type":2}},"data":
{"config_update":'$(cat update.json)'}}}' | jq . > update_in_envelope.json
```

```
configtxlator proto_encode --input update_in_envelope.json
--type common.Envelope --output update_in_envelope.pb

# sign the update_in_envelope.pb as organisation 1. Organisation 2
# submits the update, which includes its signature as well,
# achieving majority endorsement
peer channel signconfigtx -f update_in_envelope.pb

export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/
org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=localhost:9051

peer channel update -f update_in_envelope.pb
-o localhost:7050 --ordererTLSHostnameOverride orderer.example.com
--tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/
orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
 -c mychannel

# check channel block information for each organisation,
# to ensure that organisation 3 has been removed
echo
echo "Organisation 2 :"
echo
peer channel getinfo -c mychannel

export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
```

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

echo
echo "Organisation 1 :"
echo
peer channel getinfo -c mychannel

export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org3MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp
export CORE_PEER_ADDRESS=localhost:11051

echo
echo "Organisation 3 :"
echo
peer channel getinfo -c mychannel

# we will now repeat the above procedure in reverse,
# adding organisation 3 back to the channel
export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
```

```
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

peer channel fetch config config_block.pb
-o localhost:7050 --ordererTLSHostnameOverride orderer.example.com
--tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -c mychannel

configtxlator proto_decode --input config_block.pb --type common.Block
| jq .data.data[0].payload.data.config > new_config.json

# we add back organisation 3 MSP configuration that we saved previously
cat modified_config.json | jq '.channel_group.groups.Application.groups +=
{"$(cat missing_config.json)"}' > file.json.tmp
&& cp file.json.tmp new_modified_config.json
&& rm file.json.tmp

configtxlator proto_encode --input new_config.json
--type common.Config --output new_config.pb

configtxlator proto_encode --input new_modified_config.json
--type common.Config --output new_modified_config.pb

configtxlator compute_update --channel_id mychannel --original new_config.pb
--updated new_modified_config.pb --output new_update.pb

configtxlator proto_decode --input new_update.pb
--type common.ConfigUpdate | jq . > new_update.json
```

```
echo
'{"payload":{"header":{"channel_header":{"channel_id":"mychannel","type":2}},"data":
{"config_update":'$(cat new_update.json)'}}}' | jq . > new_update_in_envelope.json

configtxlator proto_encode --input new_update_in_envelope.json
--type common.Envelope --output new_update_in_envelope.pb

peer channel signconfigtx -f new_update_in_envelope.pb

export PATH= ${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org2MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
export CORE_PEER_ADDRESS=localhost:9051

peer channel update -f new_update_in_envelope.pb
-o localhost:7050 --ordererTLSHostnameOverride orderer.example.com
--tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -c mychannel

# check channel block information for each organisation,
# to ensure that organisation 3 has been added back
echo
echo "Organisation 2 :"
echo
peer channel getinfo -c mychannel

export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
```

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

echo
echo "Organisation 1 :"
echo
peer channel getinfo -c mychannel

export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org3MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp
export CORE_PEER_ADDRESS=localhost:11051

echo
echo "Organisation 3 :"
echo
peer channel getinfo -c mychannel

# cleanup all generated files except the initial configuration
# and the two updates in json format, in order to compare them
# and validate our experiment

#rm config.json modified_config.json new_modified_config.json
```

```
rm config.pb config_block.pb missing_config.json
endorsement_config.json modified_config.pb new_config.*
new_modified_config.pb update.* update_in_envelope.*
new_update.* new_update_in_envelope.*

# shut down network
./network.sh "down"
```

Script 5.1. remove_org3.sh

6. Επαύξηση δικαιωμάτων

Στα πλαίσια της έρευνάς μας, δεν εντοπίσαμε κάποια ευπάθεια όσον αφορά τα node authorisation και chaincode policy schemes. Ωστόσο, σε περίπτωση που χρησιμοποιείται μόνο ένας κόμβος orderer στο κανάλι και ένας επιτιθέμενος αποκτήσει τον έλεγχό του, τότε θα μπορεί να παρακάμψει το chaincode policy.

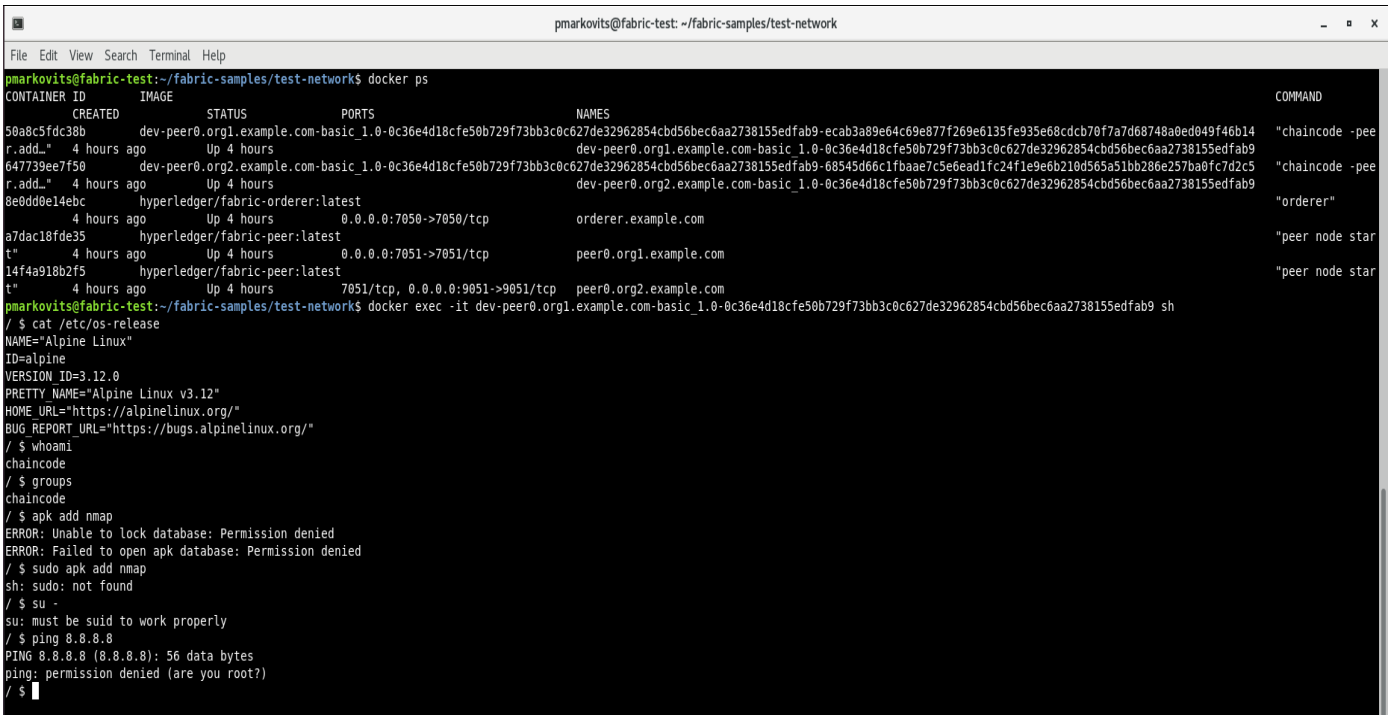
Σε παλαιότερο security assessment management report από τη Nettitude[40] πάνω στην έκδοση 1.1 του Hyperledger Fabric, είχε διαπιστωθεί ότι το sandboxing στα chaincode containers ήταν ελλιπές, καθιστώντας το σύστημα ευάλωτο σε πιθανή επίθεση επαύξησης δικαιωμάτων. Χαρακτηριστικά, η Nettitude μπόρεσε να επιδείξει την εκτέλεση κακόβουλου chaincode, το οποίο πραγματοποιεί nmap scans.

Για να διαπιστώσουμε αν τα παραπάνω εξακολουθούν να ισχύουν στην τωρινή έκδοση, εξετάσαμε τα chaincode containers ως προς δύο παραμέτρους :

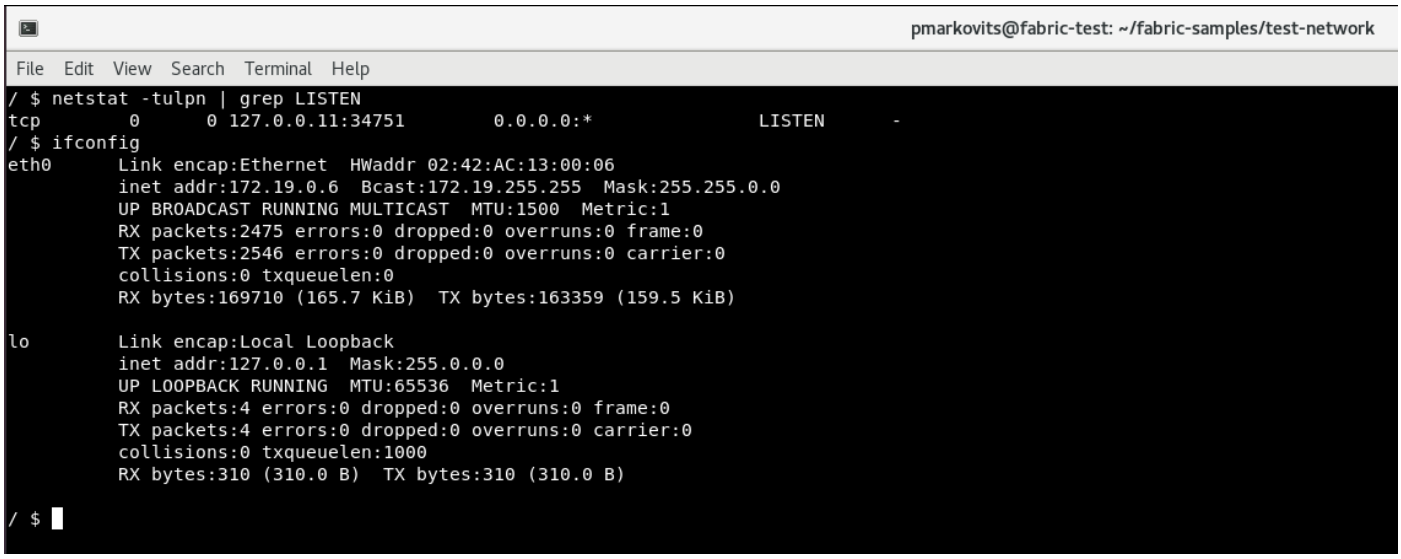
- Δικαιώματα χρήστη
- Απομόνωση από το υπόλοιπο δίκτυο

Αρχικά, συνδεόμαστε στο chaincode container του peer 0 του οργανισμού 1. Παρατηρούμε ότι το περιβάλλον του container είναι Alpine Linux v3.12. Ο χρήστης που εκτελεί το chaincode στο container λέγεται chaincode και ανήκει αποκλειστικά στο group chaincode. Ο χρήστης chaincode δε μπορεί να αποκτήσει root privileges, καθώς δεν υπάρχει sudo binary στο container και ο root χρήστης είναι hard coded. Ακόμα, δε μπορεί να χρησιμοποιήσει τον apk package manager ούτε να κάνει ping.

Ακόμα, με χρήση των εντολών netstat και ifconfig, παρατηρούμε ότι μόνο ένα port είναι ανοιχτό – διαφορετικό σε κάθε container - και τα διαθέσιμα interfaces είναι το loopback interface lo και το eth0 που συνδέει το container με το υπόλοιπο κανάλι.

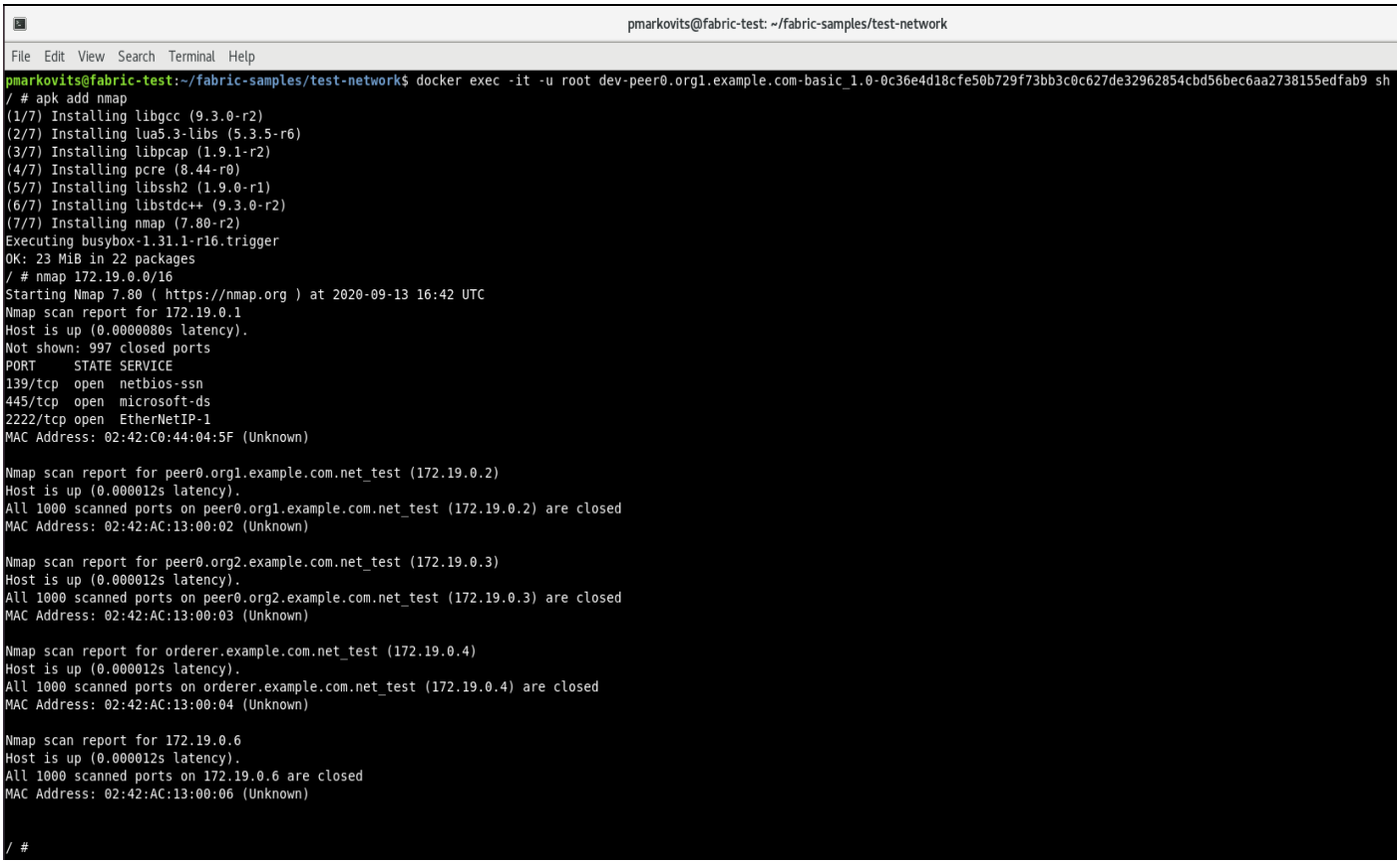


Εικόνα 6.1. chaincode user privileges



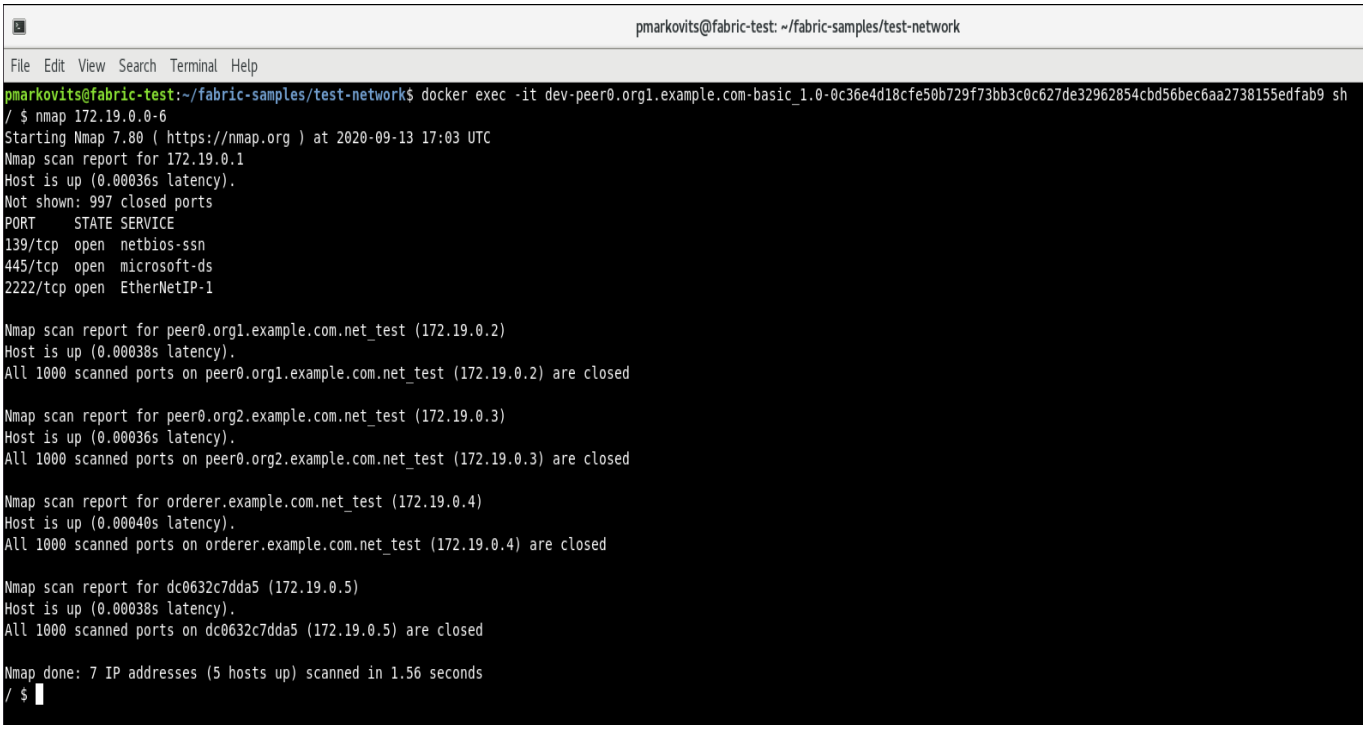
Εικόνα 6.2. chaincode network interfaces & open ports

Τέλος, συνδεόμαστε στο chaincode container ως root, εγκαθιστούμε το nmap και εκτελούμε ένα βασικό network scan, για να εξετάσουμε το worst case scenario στο οποίο ο επιτιθέμενος είτε άμεσα ως μέρος του chaincode είτε έμμεσα μέσω bash commands που εκτελούνται από το chaincode, καταφέρει να περάσει το nmap στο container και επιχειρήσει να χαρτογραφήσει το δίκτυο. Εκτελούμε το scan μια φορά ως root και μια ως chaincode, για να συγκρίνουμε τα αποτελέσματα.



Εικόνα 6.3. nmap scan as root user

Παρατηρούμε ότι το chaincode container του peer 0 του οργανισμού 2 εντοπίζεται μόνο με nmap scan στον chaincode χρήστη, ενώ η 172.19.0.6 εντοπίζεται μόνο με nmap scan ως root. Τα μόνα ανοιχτά ports εντοπίζονται στο gateway του δικτύου των docker containers, που είναι το VM στο οποίο είναι στημένο το blockchain network, και πρόκειται για ports που χρησιμοποιούνται για επικοινωνία με τον host του VM (Windows 10).



Εικόνα 6.4. nmap scan as chaincode user

Συμπεραίνουμε λοιπόν ότι αν μπορούσε ο επιτιθέμενος να εκτελέσει nmap scan στο chaincode container, θα αποκτούσε δυνητικά πληροφορίες για τον host του δικτύου και ενδεχομένως το αν πρόκειται για virtual ή physical machine, δε θα μπορούσε όμως να προχωρήσει σε κάποια άλλη επιθετική ενέργεια, καθώς θα είχε πρόσβαση μόνο ως ο χρήστης chaincode, ο οποίος έχει επαρκώς περιορισμένα προνόμια.

7. Access Granting & Revoking Vulnerabilities

Όσον αφορά το μοντέλο παροχής και ανάκλησης πρόσβασης στο δίκτυο, εντοπίστηκαν δύο ευπάθειες :

- Δεν υποστηρίζεται η ανάκληση των TLS certificates που έχουν εκδοθεί, και κατά συνέπεια μπορεί να υπάρξει επικοινωνία μεταξύ κόμβων, ακόμα και αν το αντίστοιχο MSP certificate ενός από τους χρήστες έχει ανακληθεί.
- Τα enrollment certificates (MSP) δεν λαμβάνουν υπόψη τους την καθορισμένη ημερομηνία λήξης. Επομένως, σε επίπεδο MSP ένα expired certificate εξακολουθεί να θεωρείται έγκυρο.

8. Security Single Points of Failure

Ανάλογα με το configuration του εκάστοτε δικτύου, προκύπτουν τα παρακάτω πιθανά single points of failure :

- Αν το σύστημα χρησιμοποιεί μόνο ένα κόμβο orderer, σε περίπτωση που αποκτήσει τον έλεγχό του ο επιτιθέμενος, θα μπορεί να εμποδίσει την κατάθεση συγκεκριμένων transactions ή να εγκρίνει κακόβουλα transactions.

- Αν το ίδιο CA χρησιμοποιείται για την έκδοση TLS και MSP certificates, σε περίπτωση που αποκτήσει τον έλεγχό του ο επιτιθέμενος, τότε θα μπορεί να εκδώσει TLS και MSP certificates και να προσποιηθεί ότι είναι ένας νέος κόμβος στο κανάλι, ο οποίος καλύπτει όλα τα security requirements.

9. Default Parameters Vulnerabilities

Αρχικά, εξετάζουμε τις default TLS παραμέτρους στο Test Network[1,2], το sample network που αντικαθιστά το First Network στην έκδοση 2.x του Hyperledger Fabric. Παρατηρούμε ότι τόσο στους peers όσο και στους orderers, το TLS server authentication είναι ενεργοποιημένο, όμως το client authentication είναι απενεργοποιημένο, γεγονός που καθιστά το δίκτυο ευάλωτο σε επιθέσεις DoS και DDoS.

Στη συνέχεια, εξετάζουμε τις default TLS παραμέτρους του Fabric CA module[34,35]. Τόσο ως μέρος του Test Network όσο και ως standalone module, το Fabric CA έχει το TLS ενεργοποιημένο για το CA server αλλά όχι για τον client, καθιστώντας το δίκτυο ευάλωτο σε πολλές από τις επιθέσεις που αναφέραμε στις προηγούμενες ενότητες. Παραδόξως, το επίσημο documentation ισχυρίζεται ότι το TLS είναι απενεργοποιημένο και για τον server, κάτι που εμφανώς δεν ισχύει :

tls

```
tls:
  # Enable TLS (default: false)
  enabled: true
  # TLS for the server's listening port
  certfile:
  keyfile:
  clientauth:
    type: noclientcert
    certfiles:
```

Configure this section to enable TLS communications for the CA. After TLS is enabled, all nodes that transact with the CA will also need to enable TLS.

- **`tls.enabled`**: For a secure production environment, TLS should be enabled for secure communications between nodes by setting `enabled: true` in the `tls` section of the config file. (Note that it is disabled by default which may be acceptable for a test network but for production it needs to be enabled.) This setting will configure `server-side TLS`, meaning that TLS will guarantee the identity of the *server* to the client and provides a two-way encrypted channel between them.

Εικόνα 9.1. default fabric-ca server tls configuration

Παρόμοιες ασυνέπειες συναντήσαμε και σε άλλα σημεία του documentation, στα πλαίσια των πειραμάτων μας. Για παράδειγμα, μέχρι πρότινος το Test Network tutorial ισχυριζόταν ότι το default chaincode που γίνεται deploy με την εντολή **./network.sh deployCC** είναι το *fabcar* (όπως στο First Network της προηγούμενης έκδοσης), ενώ στην πραγματικότητα το default chaincode είναι το *asset-transfer-basic* ή αλλιώς *basic*.

Τέλος, παρατηρήσαμε ότι όλα τα logs που τυπώνονται στο terminal σχετικά με το δίκτυο και το chaincode, χρησιμοποιούν το standard error stream (stderr) αντί για το standard input stream (stdin), γεγονός που μπορεί να οδηγήσει στην απώλεια πληροφοριών τις οποίες ο χρήστης αποθηκεύει από το stdin ή σχεδιαστικά λάθη σε applications που αυτός αναπτύσσει.

10. Chaincode Fuzzing

Στην παρούσα ενότητα, θα επιχειρήσουμε να βρούμε πιθανές ευπάθειες στις βασικές CRUD λειτουργίες ενός απλού Go ή javascript chaincode, με χρήση τεχνικών fuzzing. Η μεθοδολογία μας βασίζεται σε προηγούμενη δημοσίευση[36] που αφορά παλαιότερη έκδοση του Hyperledger Fabric, ***Fuzzing Hyperledger Fabric*** από τους Ankit Sharma et al. Επιλέξαμε το default chaincode του Test Network, *asset-transfer-basic*, για τη διεξαγωγή των πειραμάτων μας.

10.1 Golang Chaincode

Προσθέτουμε τις παρακάτω συναρτήσεις στον πηγαίο κώδικα **smartcontract.go** :

```
func (s *SmartContract) CreateFuzzAsset(ctx contractapi.TransactionContextInterface, id string, color string, size int, owner string, appraisedValue int) error {
	exists, err := s.AssetExists(ctx, id)
	if err != nil {
		fmt.Println(err)
	}
	if exists {
		fmt.Errorf("the asset %s already exists", id)
	}

	var rval int
	var rstring string
	var asset Asset

	f:=fuzz.New().NilChance(0)

	for i := 0; i < 10000; i++ {

		if i < 3000 {
			f.Fuzz(&rval)
			asset = Asset{
			ID:             id,
			Color:          color,
			Size:           rval,
			Owner:          owner,
			AppraisedValue: appraisedValue,
			}
			assetJSON, err := json.Marshal(asset)
```

```
                if err != nil {
                        fmt.Println(err)
                }
                err = ctx.GetStub().PutState(id, assetJSON)
                if err != nil {
                        fmt.Println(err)
                }
                err = ctx.GetStub().PutState("rval", []byte(strconv.Itoa(rval)))
                if err != nil {
                fmt.Println(err)
        }
        } else if i < 6000 && i > 3000 {
                f.Fuzz(&rstring)
                asset = Asset{
                ID:             rstring,
                Color:          color,
                Size:           size,
                Owner:          owner,
                AppraisedValue: appraisedValue,
                }
                assetJSON, err := json.Marshal(asset)
                if err != nil {
                        fmt.Println(err)
                }
                err = ctx.GetStub().PutState(rstring, assetJSON)
                if err != nil {
                        fmt.Println(err)
                }
                err = ctx.GetStub().PutState("rstring", []byte(rstring))
                if err != nil {
                        fmt.Println(err)
                }
        } else if i > 6000 {
                f.Fuzz(&asset)
                assetJSON, err := json.Marshal(asset)
                if err != nil {
                        fmt.Println(err)
```

```
                        }
                        err = ctx.GetStub().PutState(id, assetJSON)
                        if err != nil {
                                fmt.Println(err)
                        }
                }
        }
        asset = Asset{
                        ID:             id,
                        Color:          color,
                        Size:           size,
                        Owner:          owner,
                        AppraisedValue: appraisedValue,
                }

        // Write the state to the ledger

        assetJSON, err := json.Marshal(asset)
        if err != nil {
                return err
        }
        return ctx.GetStub().PutState(id, assetJSON)
}

func (s *SmartContract) ReadFuzzAsset(ctx contractapi.TransactionContextInterface, id string)
(*Asset, error) {

        var rval string

        f:=fuzz.New().NilChance(0)
        var asset Asset

        for i := 0; i < 10000; i++ {

                f.Fuzz(&rval)
                fmt.Printf("trying to read asset %s \n", rval)
                assetJSON, err := ctx.GetStub().GetState(rval)
```

```
                if err != nil {
                        fmt.Errorf("failed to read from world state: %v", err)
                }
                if assetJSON == nil {
                        fmt.Errorf("the asset %s does not exist", rval)
                }
                err = json.Unmarshal(assetJSON, &asset)
                if err != nil {
                        fmt.Println(err)
                }
                fmt.Println(string(assetJSON))
        }
        assetJSON, err := ctx.GetStub().GetState(id)
        if err != nil {
                fmt.Errorf("failed to read from world state: %v", err)
        }
        if assetJSON == nil {
                fmt.Errorf("the asset %s does not exist", id)
        }
        err = json.Unmarshal(assetJSON, &asset)
        if err != nil {
                fmt.Println(err)
        }
        return &asset, nil
}

func (s *SmartContract) UpdateFuzzAsset(ctx contractapi.TransactionContextInterface, id string, color string, size int, owner string, appraisedValue int) error {

        var rval int
        var rstring string
        var asset Asset
        f:=fuzz.New().NilChance(0)

        for i := 0; i < 10000; i++ {
                f.Fuzz(&rstring)
                exists, err := s.AssetExists(ctx, rstring)
```

```
        if err != nil {
                fmt.Println(err)
        }
        if !exists {
                fmt.Errorf("the asset %s does not exist", rval)
        }

        if i < 3000 {
                f.Fuzz(&rval)
                asset = Asset{
                ID:             id,
                Color:          color,
                Size:           rval,
                Owner:          owner,
                AppraisedValue: appraisedValue,
                }
        } else if i < 6000 && i > 3000 {
                asset = Asset{
                ID:             rstring,
                Color:          color,
                Size:           size,
                Owner:          owner,
                AppraisedValue: appraisedValue,
                }
        } else if i > 6000 {
                var asset Asset
                f.Fuzz(&asset)
        }

        assetJSON, err := json.Marshal(asset)
        if err != nil {
                fmt.Println(err)
        }
        err = ctx.GetStub().PutState(id, assetJSON)
        if err != nil {
                fmt.Println(err)
        }
```

```
	}
	exists, err := s.AssetExists(ctx, id)
	if err != nil {
		return err
	}
	if !exists {
		return fmt.Errorf("the asset %s does not exist", id)
	}

	// overwriting original asset with new asset
	asset = Asset{
		ID:             id,
		Color:          color,
		Size:           size,
		Owner:          owner,
		AppraisedValue: appraisedValue,
	}
	assetJSON, err := json.Marshal(asset)
	if err != nil {
		return err
	}

	return ctx.GetStub().PutState(id, assetJSON)
}

func (s *SmartContract) DeleteFuzzAsset(ctx contractapi.TransactionContextInterface, id string) {

	var rval string

	f:=fuzz.New().NilChance(0)

	for i := 0; i < 10000; i++ {

		f.Fuzz(&rval)
		fmt.Printf("trying to delete asset %s \n", rval)
		exists, err := s.AssetExists(ctx, rval)
```

```
                if err != nil {
                        fmt.Println(err)
                }
                if !exists {
                        fmt.Errorf("the asset %s does not exist", rval)
                }
                err = ctx.GetStub().DelState(rval)
                if err != nil {
                        fmt.Println(err)
                }
        }
}
```

Οι παραπάνω συναρτήσεις βασίζονται στις αρχικές CRUD συναρτήσεις που υλοποιεί το *asset-transfer-basic* chaincode με την προσθήκη fuzzing λειτουργικότητας, η οποία πραγματοποιείται με χρήση της βιβλιοθήκης go-fuzz[37] από τη Google, η οποία αναθέτει τυχαίες τιμές τόσο σε απλές μεταβλητές όπως int και float, όσο και σε struct δομές class objects.

Συγκεκριμένα, στις συναρτήσεις CreateFuzzAsset και UpdateFuzzAsset επιχειρούμε να δημιουργήσουμε και να ανανεώσουμε αντίστοιχα πάνω από 10000 assets που αποτελούν fuzzed παραλλαγές του αρχικού που θα δημιουργούσε η αντίστοιχη CreateAsset/UpdateAsset συνάρτηση. Τις πρώτες 3000 φορές κάνουμε fuzz το size, μια μεταβλητή μέλος του Asset struct, και αποθηκεύουμε το Asset και την τυχαία τιμή rval (int) στο state store. Τις επόμενες 3000 φορές κάνουμε fuzz το id του Asset struct και αποθηκεύουμε το Asset και την τυχαία τιμή rstring (string) στο state store. Τέλος, κάνουμε fuzz την ίδια τη δομή Asset στο σύνολο της για 4000 φορές και την αποθηκεύουμε στο state store.

Στις συναρτήσεις ReadFuzzAsset και DeleteFuzzAsset επιχειρούμε να διαβάσουμε ή να διαγράψουμε αντίστοιχα 10000 assets, κάνοντας fuzz το id (string) βάσει του οποίου ταυτοποιείται το asset προς ανάγνωση/διαγραφή.

Στη συνέχεια, κάνουμε deploy το chaincode και εκτελούμε τις συναρτήσεις με τις παρακάτω εντολές :

```
# setup Test Network
./network.sh up createChannel

# deploy chaincode
./network.sh deployCC

# use organisation 1 peer
export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
```

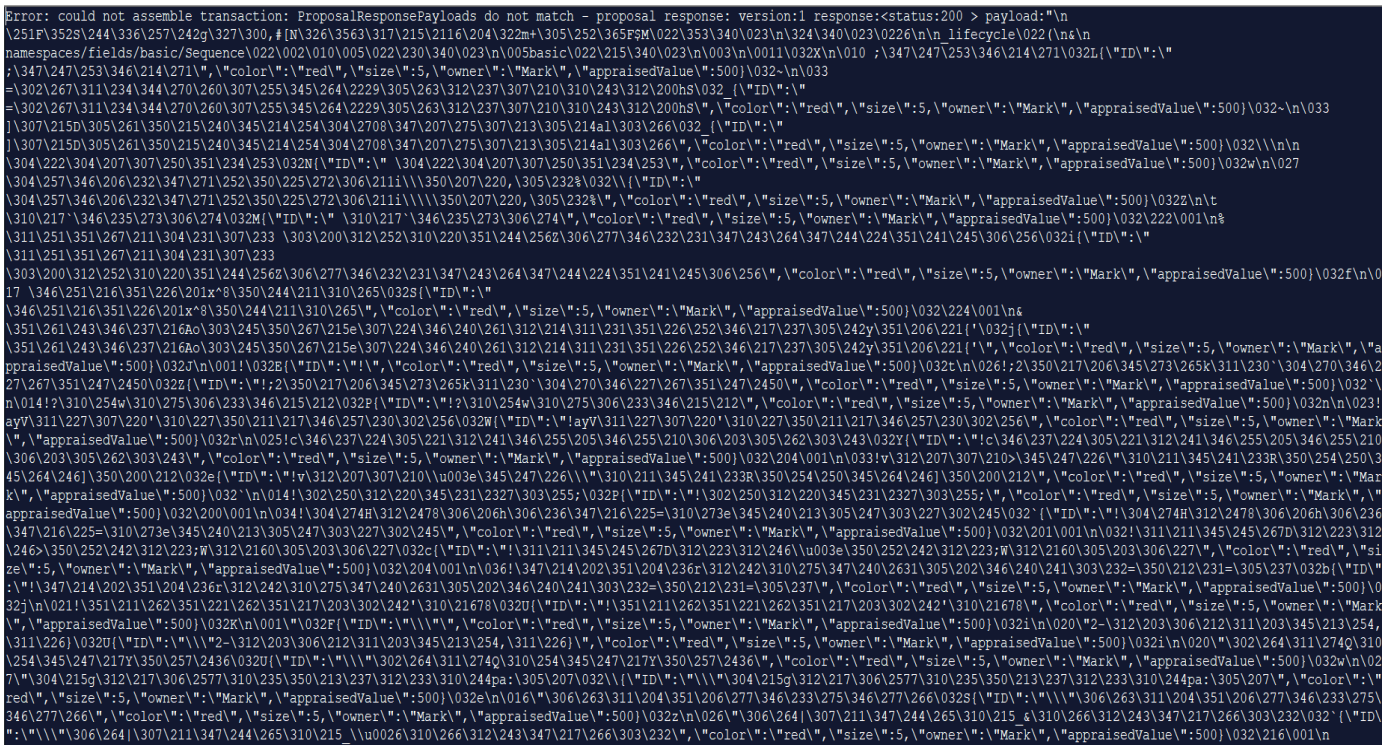
```
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/
peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/
peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

# execute fuzzed CRUD chaincode functions and save their output in separate files
peer chaincode invoke -o localhost:7050
 --ordererTLSHostnameOverride orderer.example.com
 --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
-C mychannel -n basic --peerAddresses localhost:7051
--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/
peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:9051
 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/
peer0.org2.example.com/tls/ca.crt
-c '{"Args":["CreateFuzzAsset","1","red","5","Mark","500"]}' &> create.txt

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride
orderer.example.com
 --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
-C mychannel -n basic --peerAddresses localhost:7051
 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/
peer0.org1.example.com/tls/ca.crt -c '{"Args":["ReadFuzzAsset","asset1"]}' &> read.txt

peer chaincode invoke -o localhost:7050
--ordererTLSHostnameOverride orderer.example.com
--tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
-C mychannel -n basic --peerAddresses localhost:7051
 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/
peer0.org1.example.com/tls/ca.crt --peerAddresses localhost:9051
 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/
peer0.org2.example.com/tls/ca.crt
 -c '{"Args":["UpdateFuzzAsset","1","yellow","6","John","600"]}' &> update.txt
```

```
peer chaincode invoke -o localhost:7050
--ordererTLSHostnameOverride orderer.example.com
 --tls --cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/
orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
-C mychannel -n basic --peerAddresses localhost:7051
 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/
peer0.org1.example.com/tls/ca.crt -c '{"Args":["DeleteFuzzAsset","asset1"]}' &> delete.txt

# save chaincode docker container logs
docker logs -f dev-peer0.org1.example.com-
basic_1.0-577a958fb29d881098c0c8c343ca3feab6a4d20dd7570c6cbc2c7861594d9e9f
 > chaincode_logs.txt
```

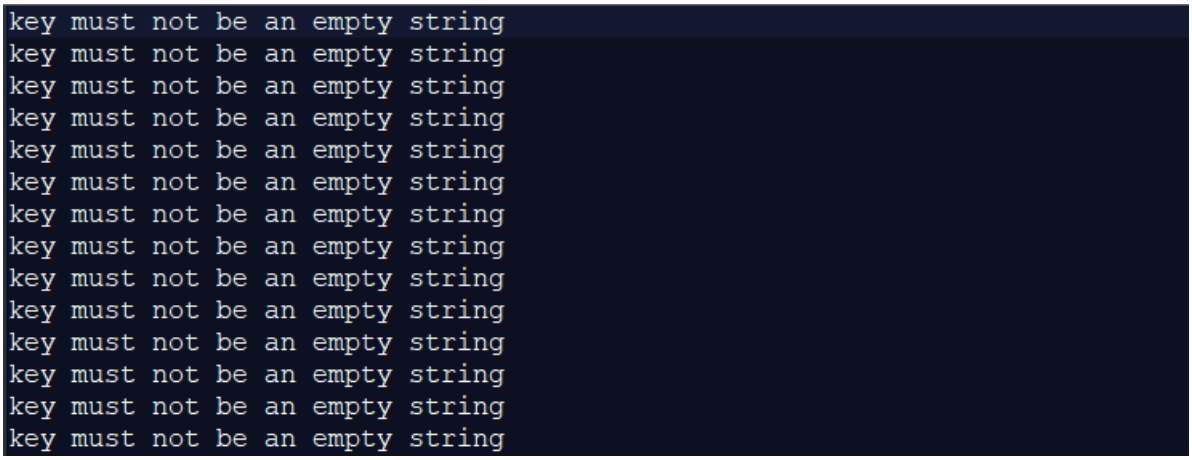
Εξετάζοντας τα logs που προκύπτουν από τα παραπάνω βήματα, παρατηρούμε τα εξής :

Create Function

Όπως ήταν αναμενόμενο, το transaction απορρίπτεται, καθώς τα payloads είναι διαφορετικά σε κάθε endorsing peer λόγω των τυχαίων τιμών που αναθέτει ο fuzzer. Αξίζει να αναφέρουμε στο σημείο αυτό ότι στη δημοσίευση των Sharma et al η ένδειξη αυτή ερμηνεύεται ως bug εξαιτίας του fuzzing, ενώ πρόκειται για ορθή συμπεριφορά που εγγυάται τη συνέπεια δεδομένων ανάμεσα στους peer ledgers και τη ντετερμινιστική συμπεριφορά του deployed chaincode. Δεν εντοπίζουμε κάποιο άλλο σχετικό σφάλμα στα docker container logs, πέρα από την απόρριψη των assets όπου το κλειδί είναι κενό ως αποτέλεσμα του fuzzing.



Εικόνα 10.1. Fuzzed Create Function logs – create.txt



Εικόνα 10.2. Chaincode Docker Container Create Function logs – chaincode_logs.txt

Read Function

Παρατηρούμε ότι υπάρχει μόνο μια ένδειξη επιτυχούς ανάγνωσης, με αποτέλεσμα ένα κενό Asset struct, αντί για ενδείξεις ότι τα assets δεν υπάρχουν στο state store. Μάλιστα, εξετάζοντας παράλληλα τα docker container logs, είναι εμφανές ότι γίνεται προσπάθεια ανάγνωσης των ανύπαρκτων assets, με το unmarshalling να αποτυγχάνει καθώς το input δεν είναι έγκυρο JSON object.

```
ESC[34m2020-09-14 13:48:54.772 EEST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001ESC[0m Chaincode invoke successful. result: status:200
payload:"{\"ID\":\"\",\"color\":\"\",\"size\":0,\"owner\":\"\",\"appraisedValue\":0}"
```

Εικόνα 10.3. Fuzzed Read Function logs – read.txt

```
trying to read asset Âjə£傑4c飐ÖhƠĀ+嬶#ʧ
unexpected end of JSON input

trying to read asset eƸOŷȫj
unexpected end of JSON input

trying to read asset ě櫻7ʦɞQ7Ő詈靚:L舸"ŏkãℾ隨鑪
unexpected end of JSON input

trying to read asset 具I6DX
unexpected end of JSON input

trying to read asset 浉pA僻f̂
unexpected end of JSON input

trying to read asset Ŕ鵙ƒDĿjă粄ĆVȸŅ-Ġ木Hŭ喃
unexpected end of JSON input

trying to read asset
unexpected end of JSON input

trying to read asset ɾ茚埿
unexpected end of JSON input

trying to read asset
unexpected end of JSON input

trying to read asset
unexpected end of JSON input

trying to read asset 嫘0
unexpected end of JSON input

trying to read asset 6咺熿ł竹ƽ7俭ŋ^垓\
unexpected end of JSON input

trying to read asset 蜉Iʂɥǣ3
unexpected end of JSON input

trying to read asset ʃ彝.ALȑ
unexpected end of JSON input
```

Εικόνα 10.4. chaincode Docker Container Read Function logs – chaincode_logs.txt

Update Function

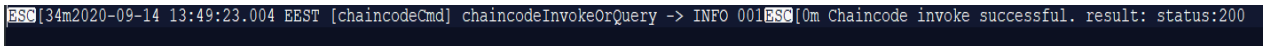
Παρατηρούμε ότι ο έλεγχος σχετικά με την ύπαρξη του asset πριν γίνει η ανανέωση, είναι πάντα επιτυχής και κατά συνέπεια το Chaincode δε προχωράει σε περαιτέρω ενέργειες.

```
Error: endorsement failure during invoke. response: status:500 message:"the asset 1 does not exist"
```

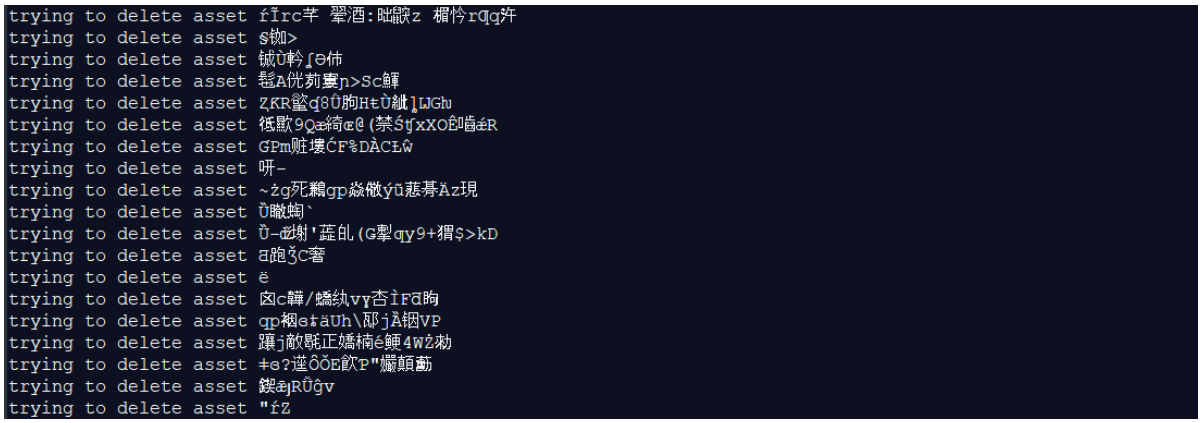
Εικόνα 10.5. Fuzzed Update Function logs – update.txt

Delete Function

Ομοίως με το Read function, ο έλεγχος ύπαρξης του asset αγνοείται, και κατά συνέπεια το chaincode προσπαθεί να διαγράψει τα ανύπαρκτα assets, χωρίς να προκύπτει κάποιο σφάλμα στα docker container logs. Ως αποτέλεσμα του chaincode invocation, λαμβάνουμε μόνο ένα μήνυμα ότι η συνάρτηση εκτελέστηκε με επιτυχία.



Εικόνα 10.6. Fuzzed Delete Function logs – delete.txt



Εικόνα 9.7. chaincode Docker Container Delete Function logs – chaincode_logs.txt

10.2 Javascript Chaincode

Προσθέτουμε τις παρακάτω συναρτήσεις στον πηγαίο κώδικα **assetTransfer.js** :

```
async CreateFuzzAsset(ctx, id, color, size, owner, appraisedValue) {
        for (var i = 0; i < 10000; i++) {
            if (i < 3000) {
                var Rval = Math.random()
                const asset = {
                    ID: id,
                    Color: color,
                    Size: Rval,
                    Owner: owner,
                    AppraisedValue: appraisedValue,
                };
                ctx.stub.putState(id, Buffer.from(JSON.stringify(asset)))
                ctx.stub.putState("Rval", Buffer.from(Rval.toString()))
            } else if (i < 6000 && i > 3000) {
                var Rstring = Math.random().toString()
                const asset = {
                    ID: Rstring,
                    Color: color,
```

```
                                        Size: size,
                                        Owner: owner,
                                        AppraisedValue: appraisedValue,
                                    };
                                ctx.stub.putState(Rstring, Buffer.from(JSON.stringify(asset)))
                                ctx.stub.putState("Rstring", Buffer.from(Rstring))
                            } else if (i > 6000) {
                                    var asset = Math.random().toString()
                                    ctx.stub.putState(id, Buffer.from(JSON.stringify(asset)))
                                }
                    }
    const asset2 = {
                            ID: id,
                            Color: color,
                            Size: size,
                            Owner: owner,
                            AppraisedValue: appraisedValue,
    };
    return ctx.stub.putState(id, Buffer.from(JSON.stringify(asset2)));
}

        async ReadFuzzAsset(ctx, id) {
                for (var i = 0; i < 10000; i++) {

                        var Rstring = Math.random().toString()
                         const assetJSON = await ctx.stub.getState(Rstring);
                        if (!assetJSON || assetJSON.length === 0) {
                                console.info(`The asset ${id} does not exist`);
                        }
                }
                const assetJSON = await ctx.stub.getState(id);
                if (!assetJSON || assetJSON.length === 0) {
                        throw new Error(`The asset ${id} does not exist`);
                }
                return assetJSON.toString();
        }
```

```
        async UpdateFuzzAsset(ctx, id, color, size, owner, appraisedValue) {
                for (var i = 0; i < 10000; i++) {
                        var Rval = Math.random()
                        var Rstring = Math.random().toString()
                        const exists = await this.AssetExists(ctx, Rstring);
                        if (!exists) {
                                console.info(`The asset ${id} does not exist`);
                        }
                        if (i < 3000) {
                                const updatedAsset = {
                                        ID: id,
                                        Color: color,
                                        Size: Rval,
                                        Owner: owner,
                                        AppraisedValue: appraisedValue,
                                };
                                ctx.stub.putState(id,
Buffer.from(JSON.stringify(updatedAsset)));
                        } else if (i < 6000 && i > 3000) {
                                const updatedAsset = {
                                        ID: id,
                                        Color: color,
                                        Size: size,
                                        Owner: owner,
                                        AppraisedValue: appraisedValue,
                                };
                                ctx.stub.putState(Rstring,
Buffer.from(JSON.stringify(updatedAsset)));
                         } else if (i > 6000) {
                                        ctx.stub.putState(Rstring,
Buffer.from(JSON.stringify(Rval)));
                                }
                }
                const exists = await this.AssetExists(ctx, id);
      if (!exists) {
         throw new Error(`The asset ${id} does not exist`);
      }
```

```
    // overwriting original asset with new asset
    const updatedAsset = {
        ID: id,
        Color: color,
        Size: size,
        Owner: owner,
        AppraisedValue: appraisedValue,
    };
    return ctx.stub.putState(id, Buffer.from(JSON.stringify(updatedAsset)));
}

async DeleteFuzzAsset(ctx, id) {
                for (var i = 0; i < 10000; i++) {
                        var Rstring = Math.random().toString()
                        const exists = await this.AssetExists(ctx, Rstring);
                        if (!exists) {
                                console.info(`The asset ${id} does not exist`);
                        }
                        ctx.stub.deleteState(Rstring);
                }
    const exists = await this.AssetExists(ctx, id);
    if (!exists) {
        throw new Error(`The asset ${id} does not exist`);
    }
    return ctx.stub.deleteState(id);
}
```

Οι παραπάνω συναρτήσεις βασίζονται στις αρχικές CRUD συναρτήσεις που υλοποιεί το *asset-transfer-basic* chaincode με την προσθήκη fuzzing λειτουργικότητας, η οποία πραγματοποιείται με χρήση της συνάρτησης *Math.random()*.

Ομοίως με το golang chaincode, στις συναρτήσεις CreateFuzzAsset και UpdateFuzzAsset επιχειρούμε να δημιουργήσουμε και να ανανεώσουμε αντίστοιχα πάνω από 10000 assets που αποτελούν fuzzed παραλλαγές του αρχικού που θα δημιουργούσε η αντίστοιχη CreateAsset/UpdateAsset συνάρτηση. Τις πρώτες 3000 φορές κάνουμε fuzz το size, μια μεταβλητή μέλος του Asset struct, και αποθηκεύουμε το Asset και την τυχαία τιμή rval (int) στο state store. Τις επόμενες 3000 φορές κάνουμε fuzz το id του Asset struct και αποθηκεύουμε το Asset και την τυχαία τιμή rstring (string) στο state store. Τέλος, κάνουμε fuzz την ίδια τη δομή Asset στο σύνολο της για 4000 φορές και την αποθηκεύουμε στο state store.

Στις συναρτήσεις ReadFuzzAsset και DeleteFuzzAsset επιχειρούμε να διαβάσουμε ή να διαγράψουμε αντίστοιχα 10000 assets, κάνοντας fuzz το id (string) βάσει του οποίου ταυτοποιείται το asset προς ανάγνωση/διαγραφή.

Το chaincode deployment & execution script είναι ίδιο με την προηγούμενη ενότητα, με μόνη διαφορά τη προσθήκη της παραμέτρου ***-ccl javascript*** στην εντολή ***./network.sh deployCC***.

Εξετάζοντας τα logs που προκύπτουν, παρατηρούμε τα εξής :

Create Function

Τα ευρήματα μας είναι ίδια με το Create Function στο golang chaincode.

Read, Update & Delete Functions

Σε αντίθεση με το golang chaincode, στο javascript chaincode οι συναρτήσεις Read, Update και Delete εντοπίζουν επιτυχώς ότι τα fuzzed assets δεν υπάρχουν, και επιστρέφουν αντίστοιχο μήνυμα σφάλματος. Εξετάζοντας παράλληλα τα docker container logs, υπάρχουν αποκλειστικά μηνύματα λάθους για μη υπαρκτά assets.

```
Error: endorsement failure during invoke. response: status:500 message:"error in simulation: transaction returned with failure: Error: The asset asset1 does not exist"
```

Εικόνα 10.8. Fuzzed Read, Update & Delete Function logs – read/update/delete.txt

```
> asset-transfer-basic@1.0.0 start /usr/local/src
> fabric-chaincode-node start "--peer.address" "peer0.org1.example.com:7052"

2020-09-14T19:17:23.459Z info [c-api:contracts-spi/bootstrap.js]                No metadata file supplied in contract, introspection will generate all the data
2020-09-14T19:17:23.461Z info [c-api:./lib/contract.js]                         Creating new Contract
2020-09-14T19:17:23.461Z info [c-api:./lib/contract.js]                         Creating new Contract "org.hyperledger.fabric"
2020-09-14T19:17:23.485Z info [c-api:lib/chaincode.js]                          Registering with peer peer0.org1.example.com:7052 as chaincode
"basic_1.0:c1cf71bede5ba730114576301947f13969959ab2787308dd26941a32b23352fa2"
2020-09-14T19:17:23.489Z info [c-api:fabric-shim/cli]
Command succeded

2020-09-14T19:17:23.537Z info [c-api:lib/handler.js]                            Successfully registered with peer node. State transferred to "established"
2020-09-14T19:17:23.537Z info [c-api:lib/handler.js]                            Successfully established communication with peer node. State transferred to "ready"
2020-09-14T19:17:52.418Z info [c-api:lib/handler.js]                            [mychannel-b46b104a] Calling chaincode Invoke() succeeded. Sending COMPLETED message
back to peer
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
The asset asset1 does not exist
```

Εικόνα 10.9. chaincode Docker Container logs – chaincode_logs.txt

11. Wormhole Attack

Στα πλαίσια ενός Hyperledger Fabric blockchain δικτύου, το κανάλι είναι ένα μέσο που διαχωρίζει μια ομάδα από ταυτότητες από τις υπόλοιπες. Όλα τα μέλη ενός καναλιού έχουν πρόσβαση στο ledger. Αρκεί ένα μέλος του καναλιού να είναι κακόβουλο και σε συνεργασία με κάποιον εξωτερικό επιτιθέμενο – ή να έχει πάρει υπό τον έλεγχό του την ταυτότητα ένας επιτιθέμενος – ώστε να είναι εφικτό ένα wormhole attack.

Για παράδειγμα, εντός ενός ιδιωτικού δικτύου, ένας κακόβουλος peer θα μπορούσε να δημιουργήσει ένα virtual private network (VPN) με εξωτερικό δίκτυο, ώστε να διαρρεύσει ευαίσθητες πληροφορίες όλων των άλλων μελών του καναλιού, χωρίς αυτοί να έχουν καμία επίγνωση της επίθεσης.

Συνεπώς, συμπεραίνουμε ότι ο μηχανισμός ελέγχου πρόσβασης του Hyperledger Fabric βασίζεται εξ ολοκλήρου στην υπόθεση ότι κάθε νόμιμο μέλος του καναλιού είναι καλόβουλο.

12. Επίλογος

Στο κεφάλαιο αυτό θα συνοψίσουμε τα συμπεράσματα της παρούσας εργασίας και θα προτείνουμε μερικές μελλοντικές επεκτάσεις της έρευνάς μας.

12.1 Συμπεράσματα

Στα πλαίσια της έρευνάς μας, επιβεβαιώσαμε τα ευρήματα της προϋπάρχουσας βιβλιογραφίας, σύμφωνα με τα οποία :

- Οι κόμβοι σε ένα Hyperledger Fabric δίκτυο δεν είναι ευάλωτοι σε key compromise, transaction replay επιθέσεις και source non-repudiation. Ακόμα, αν το Server TLS είναι απενεργοποιημένο, οι κόμβοι ενδέχεται να είναι ευάλωτοι σε man-in-the-middle και session hijacking επιθέσεις. Τέλος, αρκεί το TLS client authentication να είναι απενεργοποιημένο ή ο επιτιθέμενος να είναι χρήστης του δικτύου για να είναι εφικτή μια επίθεση denial of service.
- Όσον αφορά τα Fabric Certificate Authorities, είναι τεχνικά εφικτή μια επίθεση password brute force ή dictionary εφόσον το TLS client authentication είναι απενεργοποιημένο. Ωστόσο, δείξαμε ότι ο αριθμός των προσπαθειών που διαθέτει ο επιτιθέμενος είναι ιδιαίτερα περιορισμένος (default: 10), καθιστώντας την επίθεση πρακτικά αδύνατη, εκτός από την περίπτωση misconfiguration από πλευράς του διαχειριστή. Επιπροσθέτως, Αν το Server TLS είναι απενεργοποιημένο, δείξαμε ότι ένας επιτιθέμενος με πρόσβαση στο δίκτυο μπορεί να πραγματοποιήσει μια επίθεση sniffing και κατά συνέπεια να ανακτήσει τα credentials των registered/enrolled χρηστών. Αν δε ο εν λόγω χρήστης είναι ο διαχειριστής του CA server, ο επιτιθέμενος μπορεί στη συνέχεια να δημιουργήσει δικούς του χρήστες και να προχωρήσει σε περαιτέρω επιθέσεις στο δίκτυο ή/και σε συγκεκριμένα κανάλια σε αυτό. Ακόμα, με απενεργοποιημένο το Server TLS, ο CA server/client είναι ευάλωτος σε επιθέσεις replay, man-in-the-middle και session hijacking. Τέλος, ομοίως με τους κόμβους, το Fabric CA είναι ευάλωτο σε επίθεση denial of service αν το TLS client authentication είναι απενεργοποιημένο ή ο επιτιθέμενος να είναι χρήστης του δικτύου.
- Το κανάλι δεν πιστοποιεί την ταυτότητά του στο χρήστη, και συνεπώς ο χρήστης δε δύναται να γνωρίζει εάν το δίκτυο ή το κανάλι που χρησιμοποιεί έχει τροποποιηθεί. Για παράδειγμα, καταφέραμε να διαγράψουμε έναν οργανισμό από ένα κανάλι, να τροποποιήσουμε τις πολιτικές του και έπειτα να επαναπροσθέσουμε τον οργανισμό στο κανάλι, χωρίς αυτός να ειδοποιείται για τις εν λόγω αλλάγες ή για το γεγονός ότι είχε διαγραφεί προσωρινά.
- Εξακολουθεί και σε αυτή την έκδοση του Hyperledger Fabric να μην υποστηρίζεται η ανάκληση των TLS certificates που έχουν εκδοθεί, και κατά συνέπεια μπορεί να υπάρξει επικοινωνία μεταξύ κόμβων, ακόμα και αν το αντίστοιχο MSP certificate ενός από τους χρήστες έχει ανακληθεί. Επίσης, τα enrollment certificates (MSP) δεν λαμβάνουν υπόψη τους την καθορισμένη ημερομηνία λήξης. Επομένως, σε επίπεδο MSP ένα expired certificate εξακολουθεί να θεωρείται έγκυρο.
- Εντοπίζουμε δύο security single points of failure : αν το σύστημα χρησιμοποιεί μόνο ένα κόμβο orderer, σε περίπτωση που αποκτήσει τον έλεγχό του ο επιτιθέμενος, θα μπορεί να εμποδίσει την κατάθεση συγκεκριμένων transactions ή να εγκρίνει κακόβουλα transactions. Ακόμα, εφόσον το ίδιο CA χρησιμοποιείται για την έκδοση TLS και MSP certificates, σε περίπτωση που αποκτήσει τον έλεγχό του ο επιτιθέμενος, τότε θα μπορεί να εκδώσει TLS και MSP certificates και να

προσποιηθεί ότι είναι ένας νέος κόμβος στο κανάλι, ο οποίος καλύπτει όλα τα security requirements.

- Στις αρχικές ρυθμίσεις του Hyperledger Fabric δικτύου, το TLS client authentication τόσο στους κόμβους όσο και στους orderers και τους CA servers είναι απενεργοποιημένο, καθιστώντας το ευάλωτο σε αρκετές από τις επιθέσεις που αναφέρουμε παραπάνω.
- Κάθε νόμιμος χρήστης ενός καναλιού δύναται να πραγματοποιήσει επίθεση wormhole.

Επιπλέον, παρατηρήθηκαν συγκεκριμένες βελτιώσεις σε σύγκριση με τις προηγούμενες εκδόσεις του Hyperledger Fabric :

- Δεν εντοπίσαμε κάποιο σφάλμα σε επίπεδο chaincode, πραγματοποιώντας fuzzing σε chaincode γραμμένο σε Go και javascript, το οποίο εκτελεί βασικές CRUD ενέργειες.
- Το sandboxing των chaincode containers έχει ενισχυθεί, καθιστώντας δύσκολη μια επίθεση επαύξησης δικαιωμάτων.

Τέλος, αξίζει να αναφέρουμε ότι συναντήσαμε αρκετές ασυνέπειες στο documentation του project, οι οποίες ενδέχεται να προκαλέσουν σύγχυση στον χρήστη και να οδηγήσουν σε misconfigurations ή λανθασμένες υποθέσεις κατά τη συγγραφή chaincode.

12.2 Μελλοντικές Επεκτάσεις

Το Hyperledger Fabric είναι ένα project που εξελίσσεται συνεχώς, με ευρεία χρήση στην αγορά και ενεργή κοινότητα. Για το λόγο αυτό, η διαρκής επανεξέταση των παραμέτρων και του συνολικού επιπέδου ασφαλείας του είναι επιτακτική.

Δύο σημεία που χρήζουν ενδιαφέροντος και θα μπορούσαν να αξιοποιήσουν τα ευρήματά μας είναι η αλληλεπίδραση μεταξύ διαφορετικών καναλιών στο ίδιο δίκτυο, καθώς και η χρήση συλλογών ιδιωτικών δεδομένων (private data collections), καθώς και στις δύο περιπτώσεις είναι αναγκαία η συμμετοχή τρίτων χρηστών στη διαδικασία ως endorsing peers που ενδεχομένως δε τους αφορά άμεσα η διαχειριζόμενη πληροφορία και η οποία θεωρητικά αποκρύπτεται από αυτούς. Πρόκειται για ζητήματα που δεν έχουν εξερευνηθεί ακόμα εκτενώς και είναι μείζονος σημασίας για την ιδιωτικότητα των δεδομένων στα πλαίσια ενός παραγωγικού Hyperledger Fabric δικτύου.

13. Βιβλιογραφικές Πηγές

1. https://www.sciencedirect.com/topics/computer-science/bitcoin
2. https://www.sciencedirect.com/topics/computer-science/blockchain
3. Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system. https://scholar.google.com/scholar?q=Nakamoto,%20S.,%202008.%20Bitcoin:%20A%20peer-to-peer%20electronic%20cash%20system.
4. https://www.sciencedirect.com/topics/computer-science/chronological-order
5. M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman, Blockchain technology: beyond bitcoin, Appl. Innovation, 2 (2016), pp. 6-10
6. Greenspan, G., 2015a. Ending the bitcoin vs blockchain debate, http://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate.
7. K. Christidis, M. Devetsikiotis, Blockchains and smart contracts for the internet of things, IEEE Access, 4 (2016), pp. 2292-2303 https://ieeexplore.ieee.org/document/7467408
8. https://www.sciencedirect.com/topics/computer-science/distributed-databases
9. https://www.sciencedirect.com/topics/computer-science/cryptocurrency
10. CoinMarketCap, 2017. Cryptocurrency Market Capitalizations, https://coinmarketcap.com/.
11. https://www.sciencedirect.com/topics/computer-science/interoperability
12. https://www.sciencedirect.com/topics/social-sciences/heterogeneity
13. F. Tschorsch, B. Scheuermann, Bitcoin and beyond: a technical survey on decentralized digital currencies, IEEE Commun. Surveys Tutorials, 18 (3) (2016), pp. 2084-2123 https://doi.org/10.1109/COMST.2016.2535718
14. M. Haferkorn, J.M. Quintana Diaz, Seasonality and Interconnectivity Within Cryptocurrencies – An Analysis on the Basis of Bitcoin, Litecoin and Namecoin, Springer International Publishing, Cham (2015), (pp. 106–120) https://scholar.google.com/scholar?q=Seasonality%20and%20Interconnectivity%20Within%20Cryptocurrencies%20%20An%20Analysis%20on%20the%20Basis%20of%20Bitcoin,%20Litecoin%20and%20Namecoin
15. Szabo, N., 1994. Smart contracts. https://scholar.google.com/scholar?q=Szabo,%20N.,%201994.%20Smart%20contracts.
16. Szabo, N., 1997. The idea of smart contracts. https://scholar.google.com/scholar?q=Szabo,%20N.,%201997.%20The%20idea%20of%20smart%20contracts.
17. J.L. Zhao, S. Fan, J. Yan, Overview of business innovations and research opportunities in blockchain and introduction to the special issue, Financial Innovation, 2 (1) (2016), p. 28

https://scholar.google.com/scholar_lookup?title=Overview%20of%20business%20innovations%20and%20research%20opportunities%20in%20blockchain%20and%20introduction%20to%20the%20special%20issue&publication_year=2016&author=J.L.%20Zhao&author=S.%20Fan&author=J.%20Yan

18. IBM, 2017. Three ways blockchain Explorers chart a new direction, https://www-935.ibm.com/services/studies/csuite/pdf/GBE03835USEN-00.pdf.
19. Hyperledger. http://www.hyperledger.org .
20. Hyperledger Fabric. http://github.com/hyperledger/fabric .
21. https://hyperledger-fabric.readthedocs.io/en/release-2.2/test_network.html
22. https://github.com/hyperledger/fabric-samples/tree/master/test-network
23. https://raft.github.io/raft.pdf
24. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf
25. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5-draft.pdf
26. https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014
27. https://github.com/hyperledger/fabric-sdk-go/blob/master/pkg/fab/txn/env.go , line 102
28. https://github.com/hyperledger/fabric-sdk-node/blob/master/fabric-common/lib/IdentityContext.js , line 54
29. https://hyperledger.github.io/fabric-chaincode-node/release-2.2/api/fabric-shim.ChaincodeStub.html#getBinding__anchor
30. https://github.com/hyperledger/fabric-chaincode-go/blob/master/shim/stub.go , line 110
31. https://www.researchgate.net/publication/334405589_Vulnerabilities_on_Hyperledger_Fabric
32. https://hyperledger.github.io/caliper/v0.3.2/getting-started/
33. https://github.com/hyperledger/caliper-benchmarks/tree/v0.3.2
34. https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html
35. https://github.com/hyperledger/fabric-ca
36. https://www.csa.iisc.ac.in/~vg/teaching/E0-256/projects/Team2.pdf
37. Google Go-fuzz, http://github.com/google/go-fuzz
38. https://www.wireshark.org/#download

39. https://www.epfl.ch/labs/dedis/wp-content/uploads/2020/01/report-2018_2-marie-jeanne-security-assessment.pdf

40. https://www.hyperledger.org/blog/2018/02/07/hyperledger-fabric-1-0-release-process

41. https://hyperledger-fabric.readthedocs.io/en/release-2.2/policies/policies.html

42. Casino, Fran, Thomas K. Dasaklis, and Constantinos Patsakis. "A systematic literature review of blockchain-based applications: current status, classification and open issues." Telematics and Informatics 36 (2019): 55-81. https://www.sciencedirect.com/science/article/pii/S0736585318306324