Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Μεταπτυχιακή Διατριβή
Στην Ασφάλεια Υπολογιστών και Δικτύων

Ανάλυση Ασφάλειας Πρωτοκόλλων Δικτύου για Συστήματα Internet of Things (IoT)

Χρήστος Μπαλαφούτης

Επιβλέπων Καθηγητής
Δρ. Αρτέμιος Γ. Βογιατζής

Δεκέμβριος 2019

Ανοικτό Πανεπιστήμιο Κύπρου

Σχολή Θετικών και Εφαρμοσμένων Επιστημών

Ανάλυση Ασφάλειας Πρωτοκόλλων Δικτύου για Συστήματα Internet of Things (IoT)

Χρήστος Μπαλαφούτης

Επιβλέπων Καθηγητής
Δρ. Αρτέμιος Γ. Βογιατζής

Η παρούσα μεταπτυχιακή διατριβή υποβλήθηκε
προς μερική εκπλήρωση των απαιτήσεων για απόκτηση

μεταπτυχιακού τίτλου σπουδών
στην Ασφάλεια Υπολογιστών και Δικτύων

από τη Σχολή Θετικών και Εφαρμοσμένων Επιστημών
του Ανοικτού Πανεπιστημίου Κύπρου

Δεκέμβριος 2019

Περίληψη

Σε ένα κόσμο όπου το Διαδίκτυο (Internet) ενσωματώνεται όλο και περισσότερο στην καθημερινότητά μας βλέπουμε την τάση όλα τα πράγματα, τα οποία μας βοηθούν σε καθημερινές δραστηριότητες και ασχολίες να συνδεθούν και αυτά με το Διαδίκτυο. Σε αυτή την κατεύθυνση παρατηρούμε μια πληθώρα από συστήματα που απαρτίζουν το Διαδίκτυο των Αντικειμένων (Internet of Things, IoT) να αναπτύσσονται και παράλληλα πρωτόκολλα δικτύου, τα οποία προσπαθούν να καλύψουν τις ανάγκες που δημιουργούνται από την ανάπτυξη του κόσμου του IoT. Παράλληλα όμως με την ανάπτυξη των συστημάτων IoT έχουμε και την εμφάνιση νέων επιθέσεων και απειλών έναντι των συστημάτων IoT, είτε σε επίπεδο λογισμικού είτε σε φυσικό επίπεδο. Με γνώμονα τα προβλήματα σε επίπεδο λογισμικού, η παρούσα μεταπτυχιακή διατριβή μελετά την ασφάλεια υλοποιήσεων πρωτοκόλλων δικτύου που χρησιμοποιούνται ήδη σε συστήματα IoT, αναζητώντας προβλήματα στις υλοποιήσεις αυτές. Παράλληλα εξετάζει και επισημαίνει αδυναμίες δημοφιλών πρωτοκόλλων IoT. Χρησιμοποιώντας εργαλεία ελεύθερου λογισμικού και λογισμικό ανοικτού κώδικα, η διατριβή επιδεικνύει ένα τρόπο ελέγχου συστημάτων IoT που μπορεί ο καθένας να χρησιμοποιήσει.

Summary

In a world where the Internet is more and more part of our everyday life, we can see the trend that every device or "thing" that helps us with various activities becomes also connected to the Internet. With that in mind, various systems are developed and simultaneously many new protocols as well, so that the demand created from the growth of the Internet of Things (IoT) world is met. Unfortunately, in parallel with the IoT world expansion, we also witness the emergence of new security threats and attacks against IoT systems, either targeting software or hardware. In reference to the software problems, this Master Thesis contributes a security study on IoT network protocols that already are in use, seeking for potential vulnerabilities in their implementations. Moreover, the thesis identifies common mistakes and weaknesses of popular IoT protocols. By using free software and open source tools, the thesis proposes a security testing approach for IoT systems that any user can perform.

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα καθηγητή Δρ. Βογιατζή για την καθοδήγηση και το χρόνο του και τους δικούς μου που με υπέμειναν όλο αυτό το διάστημα.

Περιεχόμενα

Κεφάλαιο 1
Εισαγωγή

Ο όρος «Διαδίκτυο των Αντικειμένων (IoT)» χρησιμοποιείται για να περιγράψει τη διασύνδεση συσκευών («πραγμάτων») με το Διαδίκτυο (Internet). Η ιδέα είναι αρκετά απλή. Καθημερινές συσκευές μπορούν να ανταλλάσσουν δεδομένα μεταξύ τους ή να ελέγχονται από άλλους κόμβους σε ένα δίκτυο. Παραδείγματα τέτοιων συσκευών είναι το πλυντήριο, το ψυγείο, το αυτοκίνητο ή ένα ολόκληρο κτίριο με ένα εγκατεστημένο δίκτυο αισθητήρων. Ακόμα και μικροτσίπ, τα οποία ελέγχουν και παρακολουθούν ζωτικές λειτουργίες ενός ανθρώπου (για παράδειγμα, βηματοδότες και μετρητές ζαχάρου) ή εμφυτευμένα σε ζώα. Οποιαδήποτε ηλεκτρονική συσκευή μπορεί να συνδεθεί με το Διαδίκτυο αποτελεί κομμάτι του Διαδικτύου των Αντικειμένων. Υπάρχουν προβλέψεις ότι το 2020 θα υπάρχουν 30,7 δισεκατομμύρια συσκευές IoT και ο τζίρος των πωλήσεων θα αγγίζει τα 470 δισεκατομμύρια δολάρια ΗΠΑ το χρόνο.

Η ασφάλεια τέτοιων συσκευών αποκτά όλο και μεγαλύτερη σοβαρότητα και αξία, καθώς η κρισιμότητα των δεδομένων τα οποία δημιουργούνται και επεξεργάζονται από τα οικοσυστήματα IoT ολοένα και αυξάνεται. Οι συσκευές IoT ενσωματώνονται στην καθημερινότητά μας και επηρεάζουν όλο και περισσότερο τις ζωές μας. Μαζί όμως με την αύξηση των συσκευών αυτών και των δεδομένων που διακινούνται αυξάνεται και η προσοχή από αυτούς που προσπαθούν να εκμεταλλευτούν αυτές τις πληροφορίες για οικονομικά ή άλλα οφέλη (για

παράδειγμα τρομοκρατία και κατασκοπία). Οι πληροφορίες μπορούν να αποτελούν στοιχεία οικονομικών συναλλαγών. Λίστες με διευθύνσεις email χρηστών και άλλων προσωπικών δεδομένων πωλούνται στη μαύρη αγορά του Διαδικτύου. Βιομηχανικές εφαρμογές, ιατρικές εφαρμογές, εφαρμογές έξυπνων πόλεων μεταδίδουν και μοιράζονται όλο και μεγαλύτερο όγκο δεδομένων. Είτε αυτά είναι προσωπικού χαρακτήρα(για παράδειγμα ιατρικά δεδομένα ασθενών) είτε δεδομένα που επηρεάζουν άμεσα τη λειτουργία των βιομηχανιών (για παράδειγμα μετρήσεις πίεσης αγωγών μεταφοράς καυσίμων).

1.1 Ερευνητικά Ερωτήματα

Το κύριο ερευνητικό ερώτημα, το οποίο μελετά η παρούσα μεταπτυχιακή διατριβή, είναι ο βαθμός και το επίπεδο ασφάλειας των συστημάτων IoT ως προς τη λειτουργία τους σε ένα περιβάλλον όπου βρίσκονται διαρκώς συνδεδεμένα στο Διαδίκτυο (always-on) και, κατά συνέπεια, συνεχώς εκτεθειμένα σε επιθέσεις για τη μακρόθεν απόκτηση ελέγχου αυτών με σκοπό την υλοποίηση περαιτέρω κακόβουλων ενεργειών έναντι άλλων συστημάτων του Διαδικτύου αλλά και των ίδιων των κατόχων τους.

Η διατριβή εστιάζει στην ασφάλεια σε επίπεδο πρωτοκόλλων επικοινωνίας και εφαρμογής σε συστήματα IoT. Μελετώνται προτυποποιημένα (standardized) δικτυακά πρωτόκολλα και οι μηχανισμοί ασφάλειας που παρέχουν από τη σχεδίασή τους (Security-by-Design). Δίνεται περαιτέρω έμφαση σε συγκεκριμένες υλοποιήσεις επιλεγμένων πρωτοκόλλων και διερευνάται η δυνατότητα ανίχνευσης τρωτών σημείων με χρήση απλών τεχνικών ελέγχου ασφάλειας λογισμικού. Μέσω των ευρημάτων, γίνεται μία προσπάθεια να ανιχνευθούν οι λόγοι που οδηγούν στην ανάπτυξη ευάλωτων και ευπαθών υλοποιήσεων και να προταθούν μέθοδοι βελτίωσης των διαδικασιών ανάπτυξης λογισμικού.

1.2 Αναγκαιότητα και Σπουδαιότητα της Έρευνας

Η παρούσα μεταπτυχιακή διατριβή προσθέτει στη μέχρι τώρα έρευνα πάνω στην ασφάλεια στον κόσμο του IoT μέσω της έρευνας πάνω στα πρωτόκολλα δικτύου που χρησιμοποιούνται. Παράλληλα με την ανάπτυξη των εφαρμογών IoT και συσκευών υπάρχει η ανάγκη για έλεγχο των υλοποιήσεων των πρωτοκόλλων που συνοδεύουν αυτές τις συσκευές. Αναγνωρίζοντας τα προβλήματα που δημιουργούνται και για ποιους λόγους έχουμε αυτά τα κενά μπορούμε να προστατέψουμε καλύτερα τις επικοινωνίες και τη μετάδοση των πληροφοριών στον ψηφιακό

κόσμο, κομμάτι του οποίου είναι και το IoT. Καθώς βλέπουμε με πόσο γρήγορους ρυθμούς ενσωματώνονται οι συσκευές και εφαρμογές IoT στην καθημερινότητά μας, είναι αναγκαίο να εξασφαλίσουμε στο μέγιστο δυνατό την ασφάλεια σε αυτόν τον τομέα της τεχνολογίας. Εστιάζουμε στα πρωτόκολλα δικτύου IoT γιατί είναι ένα σημείο το οποίο μπορεί να ελέγξει και ο τελικός χρήστης μιας IoT συσκευής και παράλληλα του συστήματος που χρησιμοποιεί μετά από την κυκλοφορία στην αγορά.

1.3 Δομή Διατριβής

Τα επόμενα κεφάλαια της διατριβής δομούνται ως εξής. Το Κεφάλαιο 2 παρέχει μία βιβλιογραφική ανασκόπηση σε θέματα πρωτοκόλλων και ασφάλειας σε περιβάλλοντα IoT καθώς και σε τεχνικές δοκιμής λογισμικού και ελέγχου ασφάλειας. Το Κεφάλαιο 3 αφορά στην παρουσίαση προτυποποιημένων πρωτοκόλλων IoT. Το Κεφάλαιο 3 εμπεριέχει επιπλέον μία ανάλυση της ασφάλειας που αυτά παρέχουν από τη σχεδίασή τους. Το Κεφάλαιο 4 επικεντρώνεται στον πειραματικό έλεγχο των υλοποιήσεων του πρωτοκόλλου MQTT (Message Queuing Telemetry Transport), το οποίο χρησιμοποιείται ευρέως σε συστήματα IoT, καθώς και στην παράθεση των ευρημάτων από αυτούς τους ελέγχους. Τέλος το Κεφάλαιο 5 αφορά στην παρουσίαση των συμπερασμάτων και των αιτίων που γεννούν τα εντοπισμένα προβλήματα καθώς και οι μελλοντικές κατευθύνσεις περαιτέρω έρευνας στο συγκεκριμένο πεδίο.

Κεφάλαιο 2
Βιβλιογραφική Ανασκόπηση

2.1 Το Διαδίκτυο των Αντικειμένων

Έχουν γίνει αρκετές προσπάθειες για να δοθεί ένας ορισμός για το τί είναι το Διαδίκτυο των Αντικειμένων (Internet of Things, IoT). Ο όρος «IoT» αναφέρεται σε συσκευές που μέσω διάφορων αισθητήρων συλλέγουν πληροφορίες και μέσω της σύνδεσής τους σε δίκτυα όπως το Internet τις κάνουν διαθέσιμες στους χρήστες διάφορων εφαρμογών. Ή, όπως πιο συνοπτικά ορίζει ο οργανισμός OASIS, «*Το Internet συνδέεται με τον φυσικό κόσμο μέσω αισθητήρων*» [77].

Στο πλαίσιο της παρούσας διατριβής υιοθετούμε έναν ευρύτερο ορισμό, ο οποίος προτάθηκε από το Internet Engineering Task Force (IETF):«*στον κόσμο του IoT τα αντικείμενα (ηλεκτρονικά, ηλεκτρικά και μη) συνδέονται μεταξύ τους και παρέχουν υπηρεσίες στους χρήστες. Οι αισθητήρες και οι φορητές συσκευές, όπως τα κινητά, απαρτίζουν τον IoT κόσμο και προσφέρουν τη δυνατότητα να υπάρχει πρόσβαση σε υπηρεσίες κάθε στιγμή και από οπουδήποτε*»[77].

Ένα σύστημα IoT είναι μία ομάδα από συσκευές, ή και μία αυτόνομη συσκευή(υλικό, αισθητήρες, υλισμικό, λειτουργικό σύστημα και λογισμικό), που επικοινωνούν μεταξύ τους και με το Διαδίκτυο. Μπορούμε να το διακρίνουμε σε τέσσερα συστατικά μέρη:

1) **Συσκευές – Αισθητήρες**: Οι συσκευές και οι αισθητήρες μέσα στο σύστημα είναι υπεύθυνες για τη συγκέντρωση δεδομένων και την αλληλεπίδραση με το φυσικό περιβάλλον. Για παράδειγμα μία κάμερα καταγράφει εικόνα σε ένα δωμάτιο και ένας έξυπνος διακόπτης κλείνει το φωτισμό.

2) **Σύνδεση με το Διαδίκτυο**: Τα δεδομένα που συλλέγουν οι συσκευές και οι αισθητήρες μεταδίδονται σε κάποιο χώρο αποθήκευσης και περαιτέρω επεξεργασίας (για παράδειγμα cloud storage). Αυτό μπορεί να γίνει είτε μέσω ενσύρματων συνδέσεων (για παράδειγμα EthernetLAN) είτε με ασύρματες τεχνολογίες(για παράδειγμα IEEE 802.11 WiFi ή LoRa).

3) **Επεξεργασία δεδομένων**: Τα δεδομένα που συλλέγονται από τους αισθητήρες υπόκεινται σε μία πρωτογενή επεξεργασία για άμεση αντίδραση. Για παράδειγμα ένας αισθητήρας στάθμης υγρών καταγράφει το βαθμό πλήρωσης μίας δεξαμενής και το σύστημα συγκρίνει τις μετρήσεις με μία ανώτερη ή κατώτερη τιμή για να βγάλει ένα συμπέρασμα και να αποστείλει άμεσα ένα μήνυμα προειδοποίησης, αν είναι απαραίτητο.

4) **Παρουσίαση μετρήσεων**: Μετά την επεξεργασία των δεδομένων το σύστημα ανάλογα με την εφαρμογή δίνει κάποια αποτελέσματα. Είτε με τη μορφή ειδοποιήσεων στο χρήστη, όπως για παράδειγμα ηχητικοί συναγερμοί, αποστολή email και άνοιγμα διακοπτών φωτισμού. Ακόμα μπορεί να υπάρχει η πρόσβαση του χρήστη σε ένα γραφικό περιβάλλον, ώστε να βλέπει τα συλλεχθέντα δεδομένα και αναλόγως να έχει τη δυνατότητα να αποφασίσει τις επόμενες ενέργειες του συστήματος. Για παράδειγμα ένας γιατρός βλέπει τα δεδομένα από τις εξετάσεις ή τις μετρήσεις από συσκευές παρακολούθησης ενός ασθενή και μπορεί να αποστείλει οδηγίες για τη θεραπεία[2].

Ένα πιο σύνθετο παράδειγμα συστημάτων IoT είναι τα έξυπνα σπίτια [78, 79]. Σε ένα σπίτι μπορούμε να έχουμε αισθητήρες κίνησης, οι οποίοι ανάβουν αυτόματα τα φώτα. Επιπλέον, ο χρήστης της εφαρμογής μπορεί να αναβοσβήνει τα φώτα σε κάθε δωμάτιο με μία φωνητική εντολή. Επίσης μέρος του συστήματος αποτελούν και οι έξυπνες κλειδαριές στις πόρτες. Οι χρήστες μπορούν να στέλνουν προσωρινά κλειδιά σε επισκέπτες και αυτοί χρησιμοποιώντας τα κινητά τους να πλησιάζουν την κλειδαριά, η οποία διαβάζει τα δεδομένα από το κινητό (με τεχνολογία proximity sensor)και να επιτρέπει την πρόσβαση στο χώρο του σπιτιού.

Ένα παράδειγμα IoT στη βιομηχανία αποτελεί ένα σύστημα από αισθητήρες πίεσης σε αγωγούς μεταφοράς αερίων. Οι αισθητήρες ελέγχουν την πίεση σε όλο το μήκος του αγωγού αποστέλλοντας τις πληροφορίες σε ένα κεντρικό ελεγκτή, όπου και ελέγχονται οι τιμές. Σε περίπτωση έκτακτης ανάγκης, όπως για παράδειγμα ανίχνευσης μεγάλης πίεσης, το σύστημα μπορεί να διακόψει τη ροή του αερίου ή να την εκτρέψει ή να επιτρέψει ελεγχόμενη διαρροή, ώστε να επιτευχθεί η μείωση της πίεσης.

2.2 Ασφάλεια και Ιδιωτικότητα στο IoT

Τα συστήματα IoT μπορούν να παρέχουν πολλές λύσεις και ευκολίες στο πεδίο της ασφάλειας. Συσκευές IoT μπορούν να χρησιμοποιηθούν για να έχουμε αποδοτικότερα συστήματα ελέγχου της κυκλοφορίας και της πρόληψης ατυχημάτων στους δρόμους. Επίσης υπάρχουν αντικλεπτικά συστήματα που δίνουν την τοποθεσία συσκευών τηλεφώνων που κλάπηκαν. Συστήματα εντοπισμού οχημάτων ή ανθρώπων που βρίσκονται σε ανάγκη ειδοποιώντας σε βοήθεια όταν κάποιος ορειβάτης για παράδειγμα χαθεί ή χτυπήσει. Συστήματα πυρασφάλειας που μπορούν να ειδοποιήσουν παράλληλα με την κατάσβεση και τις αρχές μέσω Internetγια τον κίνδυνο.

Τα συστήματα IoT, αν δεν προσεχθεί η ασφάλειά τους, μπορεί να χρησιμοποιηθούν σε βάρος των χρηστών και ιδιοκτητών τους. Ένα παράδειγμα είναι οι έξυπνες κάμερες που υπάρχουν σε δωμάτια σπιτιών. Αν πέσουν στον έλεγχο τρίτων, μη εξουσιοδοτημένων προσώπων, παραβιάζεται η ιδιωτικότητα των ενοίκων καθώς ο επιτιθέμενος μπορεί να βλέπει εικόνες από το εσωτερικό του σπιτιού. Συσκευές οι οποίες μπορεί να συλλέγουν πληροφορίες για τις συνήθειες και ενέργειες των χρηστών στο Internetδίνουν πληροφορίες και δεδομένα στις εταιρείες χωρίς οι χρήστες να έχουν δώσει τη συγκατάθεσή τους, όπως ένα κινητό και διάφορες εφαρμογές λογισμικού ή ένα ρολόι με GPS, το οποίο καταγράφει και δημοσιοποιεί όλες τις διαδρομές του κατόχου του. Αν παραβιαστεί η ασφάλεια μίας συσκευής καταγραφής ιατροβιολογικών δεδομένων ασθενή, τότε ο επιτιθέμενος αποκτά πρόσβαση σε προσωπικές πληροφορίες ενός ασθενή, οι οποίες αφορούν την κατάσταση της υγείας του. Οι πληροφορίες αυτές θα μπορούσαν ενδεχομένως να χρησιμοποιηθούν εναντίον του, όπως για παράδειγμα την αλλαγή των ασφαλίστρων υγείας με βάση τα υποκλαπέντα δεδομένα. Πολλές εταιρείες υιοθετούν την πολιτική "bring your own device" (BYOD) στο χώρο εργασίας (για παράδειγμα, προσωπικός φορητός υπολογιστής ή προσωπικό έξυπνο τηλέφωνο). Κάτι όμως που μπορεί να έχει συνέπειες για την ασφάλεια του εταιρικού δικτύου και των εταιρικών δεδομένων, καθώς οποιαδήποτε από αυτές τις συσκευές μπορεί να έχει παραβιαστεί χωρίς να μπορούν οι διαχειριστές να την ελέγξουν, λειτουργώντας έτσι ως «Δούρειος Ίππος» (Trojan horse).

Ένας από τους μεγαλύτερους κινδύνους που αντιμετωπίζουν τα συστήματα IoT είναι η προσβολή τους και ο έλεγχός τους μέσω botnet σαν το Mirai [24, 73] και το Reaper [25]. Αυτά είναι λογισμικά ειδικά σχεδιασμένα για να αποκτούν τον έλεγχο των συστημάτων IoT και έπειτα να τα χρησιμοποιούν για προηγμένες επιθέσεις, όπως για παράδειγμα κατανεμημένες επιθέσεις άρνησης υπηρεσίας (Distributed Denial of Service attack, DDoS) εναντίων τρίτων ιστοσελίδων και

εφαρμογών. Αυτή είναι μία ολοένα και πιο δημοφιλής τακτική: το 2017 υπήρξε αύξηση[1] κατά 97% στις επιθέσεις DDoS, στις οποίες συμμετείχαν συσκευές IoT. Τα κακόβουλα λογισμικά (malicious software, malware) έχουν αυξηθεί, όπως και η πολυπλοκότητά τους [62].

Σε κάθε εφαρμογή όπου η επικοινωνία των συστημάτων γίνεται μέσω δημόσιων δικτύων και του Διαδικτύου είναι πάρα πολύ σημαντική η οργάνωση και ο σχεδιασμός της ασφάλειας. Όσο αυξάνονται οι συσκευές που συνδέονται στο Internet, με τον ίδιο ρυθμό υπάρχει και η αύξηση στα σημεία που μπορεί να γίνει επίθεση [8]. Σύμφωνα με έρευνα[2] της εταιρείας BitDefender, σε κάθε σπίτι στις ΗΠΑ εκτιμάται ότι υπάρχουν ήδη δύο ή περισσότερες «έξυπνες» συσκευές.

Διαφορετικά περιβάλλοντα και διαφορετικές εφαρμογές έχουν διαφορετικές απαιτήσεις λειτουργίας και κατά συνέπεια, διαφορετικό σχεδιασμό και υπολογιστικούς πόρους. Για αυτό υπάρχουν και διαφορετικές συσκευές IoT. Κάθε περιβάλλον έχει αντίστοιχα και διαφορετικές απαιτήσεις ασφαλείας. Σε μία εφαρμογή όπου έχουμε κάποιους αισθητήρες σε ένα χωράφι να αποστέλλουν μετρήσεις υγρασίας δεν υπάρχει η τόσο μεγάλη ανάγκη για κρυπτογράφηση της πληροφορίας που μεταδίδεται. Οπότε δε χρειάζονται λύσεις και χρήση πρωτοκόλλων κρυπτογράφησης. Σε τέτοιες περιπτώσεις η επικοινωνία μπορεί να απλουστευθεί και να εστιάσουμε στην αποτελεσματικότητα και την ορθή μέτρηση. Σε αντίθεση, η μετάδοση μηνυμάτων σε ένα κτίριο για έλεγχο του φωτισμού ή έλεγχο άλλων συστημάτων ασφαλείας, δεν μπορεί να επιτρέπεται οποιαδήποτε συσκευή (client) να στέλνει μηνύματα ελέγχου στο κεντρικό σύστημα (server), το οποίο να τα αναμεταδίδει ως «τυφλές» εντολές ελέγχου στους υπόλοιπους κόμβους του δικτύου. Σε τέτοια σενάρια χρήσης, η κρυπτογράφηση των μηνυμάτων, ο έλεγχος της ακεραιότητας των μηνυμάτων καθώς και ο έλεγχος ταυτότητας όλων των κόμβων που μεταδίδουν μηνύματα είναι απαραίτητα.

2.3Ασφάλεια συστημάτων IoT

Η μετεξέλιξη του Διαδικτύου σε ένα Διαδίκτυο των Αντικειμένων εντοπίζεται, μεταξύ άλλων, στο [14]. Κακόβουλα λογισμικά, επίδοξοι hacker, οργανωμένο έγκλημα και κυβερνο-τρομοκρατία (cyber-terrorism) είναι μερικές μόνο από τις απειλές στον κόσμο του IoT. Σύμφωνα με έρευνα[3] της εταιρείας HPτο 2014, το 70% των συσκευών είναι ευάλωτες σε επιθέσεις [37]. Το ποσοστό

[1] https://www.techrepublic.com/article/ddos-attacks-increased-91-in-2017-thanks-to-iot/
[2] https://www.bitdefender.com/files/News/CaseStudies/study/229/Bitdefender-Whitepaper-The-IoT-Threat-Landscape-and-Top-Smart-Home-Vulnerabilities-in-2018.pdf
[3] https://doi.org/10.1109/MC.2018.2141033

αυτό είναι ιδιαίτερα υψηλό. Μέσα στην ίδια έρευνα παρουσιάζονται προβλήματα που εντοπίζονται αντίστοιχα και σε άλλες εφαρμογές Διαδικτύου.

Τα θέματα ασφάλειας των συστημάτων IoTμπορούμε να τα αναλύσουμε και κατηγοριοποιήσουμε σύμφωνα με το τρίπτυχο «CIA» της ασφάλειας της πληροφορίας [76]. Προβλήματα δηλαδή με την εμπιστευτικότητα (confidentiality), την ακεραιότητα (integrity) και τη διαθεσιμότητα (availability).

2.3.1 Προβλήματα με την εμπιστευτικότητα

Όταν αναφερόμαστε στην εμπιστευτικότητα εννοούμε την προστασία της πληροφορίας από τρίτους. Η μη εξουσιοδοτημένη πρόσβαση (unauthorized access) σε πληροφορίες μπορεί να προκύψει από πλήρη έλλειψη μηχανισμών εξουσιοδότησης (authorization). Ο οποιοσδήποτε καταφέρει και συνδεθεί σε μία εφαρμογή ή ένα δίκτυο θα μπορεί να έχει πρόσβαση σε όλους τους πόρους και τα αρχεία χωρίς καμία διάκριση. Κάποιοι πόροι και πληροφορίες είναι πιο σημαντικοί ή αποτελούν ευαίσθητα δεδομένα. Η έλλειψη μηχανισμών ελέγχου εξουσιοδότησης μπορούν να οδηγήσουν σε μη εγκεκριμένη πρόσβαση σε πληροφορίες. Συσκευές-κλώνοι (cloning device) είναι «ξένες» συσκευές όπου έχουν καταφέρει να συνδεθούν στο δίκτυο και να ενεργούν ως πιστοποιημένες πραγματοποιώντας υποκλοπή και έκθεση δεδομένων (data exposure),τα οποία μεταφέρονται μέσω αυτών των συσκευών σε τρίτους [34].

Η ύπαρξη αδύναμων μηχανισμών πιστοποίησης με προκαθορισμένα στοιχεία πρόσβασης (defaultή fixed password για παράδειγμα) ή και αδύναμα συνθηματικά πρόσβασης (password) είναι ένας από τους λόγους που επιτιθέμενοι χρήστες αποκτούν πρόσβαση σε πληροφορία που δεν τους ανήκει [33]. Ένα κομμάτι στο οποίο φέρουν ευθύνη και οι τελικοί χρήστες των συσκευών, καθώς πολλοί από αυτούς δεν αλλάζουν τα στοιχεία πρόσβασης είτε χρησιμοποιούν πολύ εύκολους συνδυασμούς χαρακτήρων και αριθμών ως συνθηματικά πρόσβασης [12].

Τα συστήματα IoT προσπαθούν να απλοποιήσουν και να δώσουν λύσεις σε ένα εύρος εφαρμογών. Και ένα από τα προτερήματά τους είναι η απλότητα στη χρήση τους. Για αυτό και οι κατασκευαστές προσπαθούν να τις κάνουν όσο το δυνατό φιλικότερες στο χρήστη. Οι έξυπνες συσκευές (όπως για παράδειγμα λαμπτήρες, κάμερες κλειστού κυκλώματος παρακολούθησης, συσκευές επέκτασης δικτύων WiFi) συνδέονται μέσω ενός πατήματος κουμπιού σε οικιακά και επαγγελματικά ασύρματα δίκτυα. Ένας σημαντικός λόγος που παρατηρούνται προβλήματα με την ασφάλεια σε συσκευές IoT είναι η πίεση χρόνου για την ταχύτερη διάθεση στην αγορά (time-

to-market) των συστημάτων IoT από τους κατασκευαστές, με αποτέλεσμα να μη δίνεται η απαραίτητη προτεραιότητα και προσοχή στα θέματα ασφάλειας των συσκευών [20, 33].

Οι έξυπνες συσκευές παρέχουν μεν τη δυνατότητα και την ευκολία να ρυθμίζονται σε μεγάλο βαθμό αυτόματα αλλά αυτό ενέχει πάντα κινδύνους. Υπάρχουν πολλές περιπτώσεις όπου οι ρυθμίσεις που είναι προγραμματισμένες από το εργοστάσιο μπορούν εύκολα να χρησιμοποιηθούν εναντίον των χρηστών και να γίνουν σημεία πρόσβασης για κακόβουλους χρήστες, όπως για παράδειγμα, προκαθορισμένα (default) διαπιστευτήρια σύνδεσης [8, 15, 20].

Οι κατασκευαστές κοιτώντας την λειτουργικότητα πρώτα και το οικονομικό κέρδος δε δίνουν την ίδια βαρύτητα στην ασφάλεια. Κάτι το οποίο αποδεικνύεται και από το πόσες συσκευές καταφέρνουν διάφορα botnet όπως το Mirai [73,84] και το Hajime [74,84] να προσβάλουν και να αποκτήσουν τον έλεγχό τους. Επίσης παρατηρούνται προβλήματα στις διαδικασίες αναβάθμισης του λογισμικού αυτών των συσκευών. Οι διαδικασίες δεν είναι πάντα αυτόματες αλλά απαιτούν την ενεργοποίηση και συμμετοχή του χρήστη-καταναλωτή. Το μεγαλύτερο ποσοστό των χρηστών είτε τις αμελεί είτε δεν έχει επαρκείς τεχνικές γνώσεις και χρόνο ώστε να τις συντηρεί διαρκώς [12].

Ο τελικός χρήστης-καταναλωτής θα πρέπει να έχει κάποιες βασικές γνώσεις για την προσεκτική χρήση συσκευών IoT και εφαρμογών με τέτοιο τρόπο, ώστε να προστατεύει την ιδιωτικότητά του. Για παράδειγμα, είναι ευθύνη του κατασκευαστή να παρέχει ένα ασφαλές αρχικό συνθηματικό και κατανοητές οδηγίες για την επιλογή ενός ασφαλούς νέου συνθηματικού κατά την πρώτη χρήση, ενώ ο καταναλωτής θα πρέπει να φροντίζει να ακολουθεί αυτές τις οδηγίες και να εφαρμόζει τους στοιχειώδεις κανόνες «ψηφιακής υγιεινής» (digital hygiene).

Η ετερογένεια των συσκευών και συστημάτων σε περιβάλλοντα IoT είναι ιδιαίτερα μεγάλη. Πολλές διαφορετικές συσκευές, με διαφορετικά πρωτόκολλα επικοινωνίας και με διαφορετικά δίκτυα σύνδεσης αποτελούν παράγοντες που δυσκολεύουν τον σχεδιασμό στην ασφάλεια και αφήνουν κενά προς εκμετάλλευση [10]. Σε αυτόν τομέα εντοπίζεται και η έλλειψη διεθνών κανονισμών για τη διασύνδεση όλων αυτών των διαφορετικών τεχνολογιών, το οποίο και φέρνει μεγάλες δυσκολίες στην επίβλεψη και την ενσωμάτωση σε ένα σύστημα [12].

Τέλος, η έλλειψη κρυπτογράφησης ή η χρήση αδύναμης κρυπτογράφησης της πληροφορίας οδηγεί σε προβλήματα εμπιστευτικότητας [10, 11]. Οι συσκευές IoT έχουν σε πολλά σενάρια χρήσης χαμηλούς επεξεργαστικούς πόρους και είτε δεν υλοποιείται κάποια τεχνική

κρυπτογράφησης είτε αυτή είναι αδύναμη (αδύναμοι και ξεπερασμένοι αλγόριθμοι κρυπτογράφησης). Άλλος ένας λόγος έλλειψης κρυπτογράφησης μπορεί να είναι και οι ενεργειακές ανάγκες συσκευών IoT [33]. Πολλές από αυτές εξαρτώνται από μπαταρίες και η χρήση αλγορίθμων κρυπτογραφίας είναι μια ενεργειακά απαιτητική διαδικασία. Δυστυχώς, πολλοί κατασκευαστές επιλέγουν να την αφαιρέσουν, θυσιάζοντας την ασφάλεια των δεδομένων και την προστασία των χρηστών προς όφελος της μεγαλύτερης ενεργειακής αυτονομίας.

2.3.2 Προβλήματα με την Ακεραιότητα

Η ακεραιότητα της πληροφορίας είναι η ιδιότητα η οποία περιγράφει ότι τα δεδομένα στα οποία έχουμε πρόσβαση ή ζητάμε πρόσβαση δεν έχουν αλλοιωθεί από κάποιον επιτιθέμενο στο σύστημα. Σε περίπτωση που η αλλοίωση έχει συμβεί, η ιδιότητα της ακεραιότητας εγγυάται ότι η αλλοίωση είναι ανιχνεύσιμη [84].

Ο κόσμος του IoT δε μένει ανεπηρέαστος από επιθέσεις στην ακεραιότητα των δεδομένων. Σε αυτές τις περιπτώσεις ο επιτιθέμενος παρεμβάλλεται στο σύστημα(για παράδειγμα, επιθέσεις middle man attack και replay attack) και αλλοιώνει την πληροφορία προς δικό του όφελος [32]. Για παράδειγμα θα μπορούσε να αλλάξει και να προσθέσει κάποιος κακόβουλο κώδικα σε μια διαδικασία αναβάθμισης λογισμικού με σκοπό να αποκτήσει τον έλεγχο μίας συσκευής. Επίσης θα μπορούσε να αλλοιώσει τις μετρήσεις ενός αισθητήρα κατά τη μετάδοση μέσω του Διαδικτύου και συνεπώς θα οδηγήσει το λήπτη της πληροφορίας σε εσφαλμένους συμπερασμούς και ενέργειες αποκατάστασης [75].

2.3.3 Προβλήματα με τη Διαθεσιμότητα

Η τρίτη ιδιότητα στην ασφάλεια της πληροφορίας είναι η διαθεσιμότητα. Δηλαδή το να είναι τα δεδομένα και οι υπηρεσίες διαθέσιμα οποτεδήποτε αυτές ζητηθούν από τους χρήστες ή άλλα εξουσιοδοτημένα συστήματα.

Όσον αφορά τα συστήματα IoT, υπάρχουν τρόποι να επιτεθεί κάποιος έμμεσα ή απευθείας σε αυτά και να τα καταστήσουν αδύναμα να επικοινωνήσουν με το υπόλοιπο δίκτυο. Για παράδειγμα έμμεσες επιθέσεις DDoS στους εξυπηρετητές με τους οποίους τα συστήματα επικοινωνούν είτε απευθείας στα ίδια τα συστήματα.

Ένα άλλο παράδειγμα είναι οι επιθέσεις σε συστήματα που χρησιμοποιούν μπαταρίες ως πηγή ενέργειας. Σε αυτές τις περιπτώσεις, μία επίθεση δημιουργεί άσκοπα αιτήματα προς τα συστήματα IoT, ώστε τα τελευταία να λειτουργούν συνεχώς τους με στόχο να εξαντληθεί πολύ γρήγορα η διαθέσιμη ενέργεια των μπαταριών τους. Έτσι τα συστήματα βγαίνουν λειτουργίας και φυσικά εκτός δικτύου, επηρεάζοντας έμμεσα και τη λειτουργία του ίδιου του δικτύου και των υπόλοιπων κόμβων του.

Ένας άλλος τρόπος επίθεσης στη διαθεσιμότητα ενός συστήματος IoTείναι η μη εξουσιοδοτημένη πρόσβαση σε αυτό. Ο επιτιθέμενος αλλάζει τα διαπιστευτήρια σύνδεσης, «κλειδώνοντας» τον τελικό χρήστη-ιδιοκτήτη έξω από το σύστημά του. Σε ένα περιβάλλον IoTδεν πρέπει να ξεχνάμε ότι, λόγω της φύσης των συστημάτων, ένας επιτιθέμενος θα μπορούσε να έχει φυσική πρόσβαση στις συσκευές που είναι εγκατεστημένες σε εξωτερικούς χώρους και να τις αφαιρέσει ή να τις θέσει εκτός λειτουργίας.

2.4 Έλεγχος Λογισμικού και Δοκιμές Ασφάλειας σε IoT

2.4.1 Κύκλος ανάπτυξης λογισμικού

Ο έλεγχος λογισμικού (software testing)αναφέρεται στη διαδικασία ελέγχου της συμπεριφοράς ενός προγράμματος σε σύγκριση με την αναμενόμενη σύμφωνα με τις προδιαγραφές του. Δηλαδή, όταν σε μία εφαρμογή εισάγουμε κάποια δεδομένα ή χρησιμοποιούμε κάποια εργαλεία της, αν το αποτέλεσμα είναι αυτό το οποίο έχει προβλεφθεί και σχεδιασθεί να είναι. Με γνώμονα αυτή τη λειτουργία οι εταιρείες και οι προγραμματιστές ελέγχουν τα προϊόντα τους προσπαθώντας να ανακαλύψουν αποκλίνουσες συμπεριφορές και αποτελέσματα. Ο έλεγχος λογισμικού (software testing)είναι μία από τις πιο σημαντικές διαδικασίες του κύκλου ανάπτυξης λογισμικού [37] (Software Development Lifecycle, SDLC).

Οι φάσεις στις οποίες χωρίζεται το software testing είναι οι ακόλουθες [70]:

- **Requirements Analysis**. Σε αυτή τη φάση γίνεται η ανάλυση και συγκέντρωση των σημείων στα οποία θέλει η ομάδα του ελέγχου να ελέγξει το πρόγραμμα. Ποιά χαρακτηριστικά ή ποιά λειτουργία θέλουν να επαληθεύσουν;

- **Test Planning**. Εδώ γίνεται όλος ο σχεδιασμός του ελέγχου (test). Ποιά εργαλεία θα χρησιμοποιηθούν, πόσο καιρό θα πάρει το test και πόσους πόρους θα διαθέσει η ομάδα;

- **Test design**. Σε αυτή τη φάση γίνεται ο σχεδιασμός του πώς θα γίνει το test και κάτω από ποιές συνθήκες και σενάρια λειτουργίας. Τί είσοδοι θα χρησιμοποιηθούν, ποιές θα είναι οι αυτοματοποιημένοι μέθοδοι;

- **Test Implementation**. Εδώ γράφονται τα σενάρια (script), τα οποία θα χρησιμοποιηθούν για τα test και ο κώδικας, ο οποίος είναι απαραίτητος για τα σενάρια ελέγχου.

- **Test Execution**. Το στάδιο στο οποίο τρέχουν όλα τα testκαι συλλέγονται τα δεδομένα από τις εξόδους.

- **Test Closure**. Συλλογή και ανάλυση των εξόδων του test. Επίσης γίνεται παρουσίαση των αποτελεσμάτων στην ομάδα ανάπτυξης για διορθώσεις.

Ο έλεγχος λογισμικού διακρίνεται σε τρεις βασικές μεθόδου [69]: λειτουργικός έλεγχος (functional testing), έλεγχος απόδοσης (performance testing) και ασφάλειας (security testing). Η διαδικασία ελέγχου μπορεί να είναι είτε αυτοματοποιημένη (παραγωγή σεναρίων ελέγχου και αυτόματη εκτέλεση ελέγχων από προγράμματα) είτε χειροκίνητη (από software tester).

Το functional testing είναι ο έλεγχος της λειτουργικότητας του λογισμικού. Δίνει το λογισμικό στην έξοδό του το αναμενόμενο αποτέλεσμα; Έχουμε την αναμενόμενη συμπεριφορά για συγκεκριμένες εισόδους;

Στο performance testing γίνεται έλεγχος σε παραμέτρους όπως χρόνοι απόκρισης, χρόνοι φόρτωσης του λογισμικού, χρόνοι συνεχούς ομαλής λειτουργίας. Σε αυτό τον έλεγχο εξετάζεται γενικά η αξιοπιστία (reliability) του λογισμικού. Το ζήτημα της αξιοπιστίας είναι ιδιαίτερα σημαντικό στον κόσμο του IoT, όπου ένα σύστημα θα πρέπει να λειτουργεί και να αποκρίνεται σύμφωνα με τις προδιαγραφές του σε όλο το διάστημα λειτουργίας του [68]. Η αξιοπιστία του λογισμικού μπορεί να επηρεαστεί από πολλούς παράγοντες, όπως οι ανανεώσεις και οι αναβαθμίσεις, η σύνδεση κάποιου νέας περιφερειακής συσκευής αλλά και η αλληλεπίδραση μέσω Διαδικτύου με άλλα συστήματα.

Στο security testing γίνεται έλεγχος του λογισμικού για τυχόν λογικά κενά και ατέλειες, τα οποία μπορούν να οδηγήσουν σε παραβίαση των προδιαγραφών ασφάλειας ενός συστήματος. Συνήθη σημεία ελέγχου αποτελούν η δυναμική διαχείριση μνήμης, η διαχείριση εισόδου από τους χρήστες (user input) και η διαχείριση σφαλμάτων κατά τη διάρκεια εκτέλεσης. Κρίσιμα σημεία από πλευράς ασφάλειας αποτελούν επίσης η διαχείριση των κρυπτογραφικών κλειδιών και ψηφιακών πιστοποιητικών, η υλοποίηση των κρυπτογραφικών αλγορίθμων καθώς και των μηχανισμών πρόσκτησης και αναίρεσης ειδικών δικαιωμάτων πρόσβασης και εξουσιοδότησης (access και authorization rights).

2.4.2 Το μεταβαλλόμενο περιβάλλον ελέγχου και δοκιμών ασφάλειας

Ο κύκλος ανάπτυξης λογισμικού (SDLC) συμπεριλαμβάνει και τον κύκλο ελέγχου του λογισμικού (Software Testing Lifecycle), ο οποίος εμπεριέχει και την ασφάλεια. Η ανάπτυξη του λογισμικού δεν οριοθετείται πλέον από πολύμηνες ή και πολύχρονες περιόδους εξαντλητικών δοκιμών ως την τελική διάθεση στην αγορά. Αντίθετα, η ανάπτυξη συνεχίζεται και μετά την παρουσίαση ενός προϊόντος στην αγορά, ενσωματώνοντας τα σχόλια των χρηστών και προσαρμόζοντάς το στις διαρκώς μεταβαλλόμενες συνθήκες λειτουργίας. Για παράδειγμα, γνωστά και δημοφιλή λειτουργικά συστήματα, όπως το Microsoft Windowsκαι το Ubuntu Linux, κυκλοφορούν τακτικά, σε επίπεδο μήνα, ενημερώσεις και βελτιώσεις, οι οποίες ενσωματώνονται στην υπάρχουσα βάση κώδικα (codebase).

Η ανάγκη διαρκούς ελέγχου λογισμικού και τακτικών δοκιμών ασφάλειας είναι πλέον μόνιμη και εκτείνεται πολύ παραπέρα από τη χρήση «παραδοσιακών» εργαλείων ελέγχου κατά τη φάση ανάπτυξης. Απαιτείται πλέον ιδιαίτερη ευρηματικότητα και βαθιά γνώση, τόσο των συστημάτων λογισμικού όσο και διαφορετικών γλωσσών προγραμματισμού και περιβαλλόντων λειτουργίας, ώστε να εντοπιστούν σφάλματα και κενά ασφάλειας, τα οποία μπορούν να θέσουν σε κίνδυνο τη λειτουργία εταιρειών, παρόχων υπηρεσιών Διαδικτύου και νεφο-υπολογιστικής (cloud computing)αλλά και ολόκληρων κρατών που συμμετέχουν στον αναδυόμενο κόσμο του ψηφιακού μετασχηματισμού.

Σε ένα τέτοιο περιβάλλον, τεχνικές ελέγχου που δε χαρακτηρίζονται από ακαδημαϊκή πληρότητα αλλά βασίζονται στη φαντασία και εφευρετικότητα μπορούν και επιτυγχάνουν σημαντικά αποτελέσματα και παρουσιάζουν ευρήματα ταχύτερα και οικονομικότερα. Μία συνήθης

πρακτική είναι η χρήση ελέγχων διεισδυτικότητας (penetration test)[4], όπου μία εταιρεία προσλαμβάνει τρίτους (μεμονωμένα άτομα ή εταιρείες) να δοκιμάσουν να παραβιάσουν την ασφάλεια του συστήματος και να εισέλθουν σε αυτό. Το σκεπτικό είναι ότι αυτοί οι τρίτοι, μη έχοντας ακριβή γνώση του λογισμικού αλλά και της πορείας ανάπτυξής του, θα μπορέσουν να εντοπίσουν λογικές διαδρομές που δεν έχουν καλυφθεί από τις εσωτερικές διαδικασίες ελέγχου και δεν έχει προβλεφθεί ο χειρισμός των καταστάσεων που προκύπτουν με έναν ασφαλή τρόπο.

Οι έλεγχοι διεισδυτικότητας κοστίζουν σημαντικά ποσά και σε πολλές περιπτώσεις είναι «μηχανοποιημένοι», με αποτέλεσμα πολλές φορές να μην έχουν κρίσιμα ευρήματα. Μία νεότερη προσέγγιση, η οποία κερδίζει διαρκώς έδαφος, είναι τα προγράμματα τύπου «bug bounty» [35]. Εδώ οι εταιρείες προκαλούν και προσκαλούν δημόσια όλους τους ενδιαφερόμενους να ανακαλύψουν και να αναφέρουν προβλήματα ασφάλειας, τα οποία, αφού αξιολογηθούν, οδηγούν σε χρηματικές αμοιβές αντίστοιχες του ευρήματος. Μόνο το 2018 δόθηκαν[5] από το GitHub περισσότερα από 250.000 δολάρια ΗΠΑ ως bug bounty. Αντίστοιχα προγράμματα[6] έχουν εταιρείες όπως για παράδειγμα η Apple, η Microsoft και η Facebook. Το πλέον πρόσφατο παράδειγμα αφορά στην εταιρεία Tesla, όπου δόθηκε[7] ως βραβείο ένα αυτοκίνητο TeslaModel 3. Το κέρδος των εταιρειών, τόσο χρηματικά όσο και σε εσωτερικό χρόνο εκτιμάται ότι είναι ιδιαίτερα μεγάλο [63].

Τα bug bounty προσφέρουν υψηλά βραβεία και ανταγωνίζονται τις αντίστοιχες αμοιβές που προσφέρονται στις «μαύρες» αγορές και στο «σκοτεινό Ιστό» (Dark Web) από «ενδιαφερόμενους» με εχθρικά κίνητρα. Δεν μπορεί ωστόσο κάποιος να είναι βέβαιος ότι ένα ανακαλυφθέν κενό ασφάλειας θα δηλωθεί στον κατασκευαστή του λογισμικού με στόχο τη δημιουργία μίας νέας, διορθωμένης έκδοσης. Θα μπορούσε εξίσου να καταλήξει στα χέρια κακόβουλων εγκληματιών ή ακόμη και αντίπαλων κρατών, οι οποίοι θα εκμεταλλεύονται την άγνοια ύπαρξης του κενού προς όφελός τους.

Τα honeypot και ιδιαίτερα τα high-interaction honeypot μπορούν να αποτελέσουν τη γραμμή άμυνας έναντι τέτοιων απειλών [17, 55, 57]. Η κεντρική ιδέα είναι ότι οι επιτιθέμενοι «παγιδεύονται» σε απομονωμένα εικονικά συστήματα, η λειτουργία των οποίων ομοιάζει με τα

[4]https://en.wikipedia.org/wiki/Penetration_test
[5]https://github.blog/2019-02-19-five-years-of-the-github-bug-bounty-program/
[6]https://www.bugcrowd.com/bug-bounty-list/
[7]https://techcrunch.com/2019/03/23/hackers-conquer-tesla-and-win-a-model-3/

πραγματικά, ώστε να καταγραφούν με κάθε λεπτομέρεια οι ενέργειες και η συμπεριφορά των επιτιθέμενων, οδηγώντας στην ανακάλυψη τρωτών σημείων που δεν ήταν έως τώρα γνωστά.

Η τεχνική fuzz testing (fuzzing) με τις διάφορες παραλλαγές της (για παράδειγμα random, heuristic, και guided) έχει χρησιμοποιηθεί ήδη από τις πρώτες μέρες των ηλεκτρονικών υπολογιστών [8] για την ανακάλυψη σφαλμάτων λογισμικού [76, 82] Η κεντρική ιδέα είναι λογισμικό προς έλεγχο (software under test, SUT) τροφοδοτείται με τυχαίες, μη έγκυρες και μη αναμενόμενες εισόδους, ώστε να ελεγχθεί η συμπεριφορά του και να εντοπιστούν σημεία εκτέλεσης εκτός προδιαγραφών. Η τεχνική μπορεί να εφαρμοστεί σε μία πληθώρα σεναρίων και ελέγχων, συμπεριλαμβανομένων λογισμικών με διεπαφή χρήστη (user interface), επεξεργασίας μορφότυπων αρχείων (file format), εφαρμογών Διαδικτύου (Internet/Web applications) και πρωτοκόλλων δικτύου [40, 41, 71].

Η χρήση της τεχνικής fuzzing, αν και είχε υποχωρήσει σημαντικά, προς χάρη ακαδημαϊκά πληρέστερων και πιο επιτηδευμένων (sophisticated) προσεγγίσεων, γνωρίζει τα τελευταία χρόνια μεγάλη άνθιση. Αυτό οφείλεται στην άπλετα διαθέσιμη επεξεργαστική ισχύ που προσφέρει η νεφο-υπολογιστική (cloud computing), στην υποστήριξη από εταιρείες όπως η Google και η Microsoft αλλά και μία σειρά από σημαντικά ευρήματα που έχει να επιδείξει το fuzzing. Τέτοια είναι το Shellshock[9], μία συλλογή από κρίσιμα προβλήματα στο λογισμικό UNIX Bourne Shell (bash) και το GODMODE[10]για επεξεργαστές x86. Η αποδοχή είναι τέτοια ώστε η Google ανακοίνωσε το 2016 το πρόγραμμα OSS-Fuzz, για το συνεχές fuzzing λογισμικού ανοικτού κώδικα ενώ ο κώδικας Chromium του φυλλομετρητή Google Chrome βρίσκεται υπό διαρκή fuzzing αξιοποιώντας 15.000 πυρήνες[11]. Αντίστοιχα η Microsoft έχει δαπανήσει[12] περισσότερα από 650 υπολογιστικά έτη για το fuzzing του φυλλομετρητή Microsoft Edge.

2.4.3 Δοκιμές ασφάλειας σε περιβάλλοντα IoT

Η ανάγκη δοκιμών μετά την εισαγωγή στην αγορά είναι ακόμη μεγαλύτερη για τα συστήματα IoT, τα οποία έμμεσα ή άμεσα απευθύνονται σε τελικούς καταναλωτές (consumer-facing IoT systems). Εδώ τα ζητήματα διαχείρισης προσωπικών δεδομένων και παραβίασης της ιδιωτικότητας είναι

[8]http://secretsofconsulting.blogspot.com/2017/02/fuzz-testing-and-fuzz-history.html
[9]https://www.nytimes.com/2014/09/26/technology/security-experts-expect-shellshock-software-bug-to-be-significant.html
[10]https://latesthackingnews.com/tag/god-mode-x86/
[11]https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf
[12]https://docs.microsoft.com/en-us/microsoft-edge/deploy/group-policies/security-privacy-management-gp

πρώτιστης σημασίας, ειδικά εντός της Ευρωπαϊκής Ένωσης (πβ. νέος Κανονισμός Προστασίας Προσωπικών Δεδομένων, General Data Protection Regulation, GDPR) αλλά και το σκάνδαλο[13] Facebook-Cambridge Analytica το 2018). Ταυτόχρονα, ο εκ των προτέρων έλεγχος καλής λειτουργίας σε όλα τα πιθανά σενάρια χρήσης και τις παραμέτρους του φυσικού (αναλογικού) κόσμου με τον οποίο αλληλοεπιδρούν τα συστήματα IoT είναι πρακτικά αδύνατος. Καθίσταται λοιπόν επιτακτική η ανάγκη για νέα εργαλεία ελέγχου της ασφάλειας των συστημάτων IoT [36-38].

Τα συστήματα IoT είναι συστήματα τα οποία είναι συνδεδεμένα στο Διαδίκτυο (online) σχεδόν αδιάλειπτα. Σε αυτό παίζει σημαντικό ρόλο και το λογισμικό το οποίο θα πρέπει να εγγυάται την αδιάκοπη και χωρίς σφάλματα, αναμενόμενη λειτουργία του συστήματος. Για παράδειγμα, ένας εξυπηρετητής σε ένα οικιακό σύστημα αυτοματισμών IoT, θα πρέπει να είναι πάντα διαθέσιμος και να ενημερώνεται από τις συσκευές αυτοματισμού. Ταυτόχρονα, θα πρέπει να είναι διαθέσιμος (availability) προς τον τελικό χρήστη-καταναλωτή, ο οποίος ελέγχει τη λειτουργία του σπιτιού, είτε είναι μέσα σε αυτό είτε βρίσκεται σε διακοπές σε μία άλλη χώρα και να εγγυάται ότι ενημερώνει το χρήστη με τις σωστές μετρήσεις του οικιακού περιβάλλοντος (integrity), χωρίς να τις εκθέτει σε τρίτους (confidentiality).

Οι υλοποιήσεις των *πρωτοκόλλων δικτύου* των συστημάτων IoT είναι κρίσιμα σημεία για τη συνολική ασφάλεια ενός συστήματος IoT, καθώς δέχονται και επεξεργάζονται ανεξέλεγκτα δεδομένα (εισόδους) από τρίτες, ενδεχομένως μη έμπιστες, πηγές. Ταυτόχρονα, οι υλοποιήσεις βρίσκονται και εκτελούνται στο πλέον εξουσιοδοτημένο τμήμα ενός λειτουργικού συστήματος, με άμεση πρόσβαση στις πιο κρίσιμες παραμέτρους λειτουργίας του. Τυχόν αστοχία κατά την επεξεργασία των δεδομένων που λαμβάνονται από το δίκτυο ή εκτροπή της εκτέλεσης του σχετικού λογισμικού μπορεί να έχει καταστροφικές συνέπειες για την ασφάλεια ενός συστήματος IoT.

Ο εξαντλητικός έλεγχος όλων των πιθανών εισόδων και όλων των πιθανών καταστάσεων (state machine) ενός πρωτοκόλλου δικτύου δεν είναι ρεαλιστικά εφικτός. Επιπλέον, η λειτουργία των συστημάτων IoT μπορεί να επηρεάζεται από το φυσικό περιβάλλον στο οποίο λειτουργούν (εξωτερικά συμβάντα που συλλαμβάνονται από τους αισθητήρες του) και συνεπώς να μεταβάλλεται η συμπεριφορά και η αντίδρασή τους.

[13] https://en.wikipedia.org/wiki/Cambridge_Analytica

Από τη διαθέσιμη ακαδημαϊκή βιβλιογραφία αλλά την ειδησεογραφία που παρουσιάστηκαν στα προηγούμενα, γίνεται σαφές ότι οι παραδοσιακές τεχνικές ελέγχου λογισμικού και δοκιμών ασφάλειας δεν επαρκούν για την περίπτωση των συστημάτων IoT. Οι παραβιάσεις ασφάλειας τέτοιων συστημάτων είναι διαρκείς και συνεχείς. Εξετάζουμε στα επόμενα κεφάλαια τους εγγενείς μηχανισμούς ασφάλειας δημοφιλών πρωτοκόλλων IoT καθώς και τις δυνατότητες εφαρμογής τεχνικών fuzzing για την αποδοτική αποκάλυψη σφαλμάτων υλοποίησης σε επιλεγμένα πρωτόκολλα IoT.

Κεφάλαιο 3
Ανάλυση Σχεδίασης Πρωτοκόλλων Δικτύου για IoT

Σημαντικό ρόλο στην ασφάλεια των συστημάτων IoT παίζει το λογισμικό και τα πρωτόκολλα δικτύου. Τα πρωτόκολλα δικτύου που χρησιμοποιούνται σε εφαρμογές IoT στοχεύουν στην απλότητα και τη χαμηλή κατανάλωση ενέργειας λόγω των χαρακτηριστικών των συσκευών IoT. Πολλές φορές βασίζονται σε υπάρχοντα πρωτόκολλα Διαδικτύου και προσαρμόζονται από τους εκάστοτε δημιουργούς στις ιδιαίτερες ανάγκες των συστημάτων IoT.

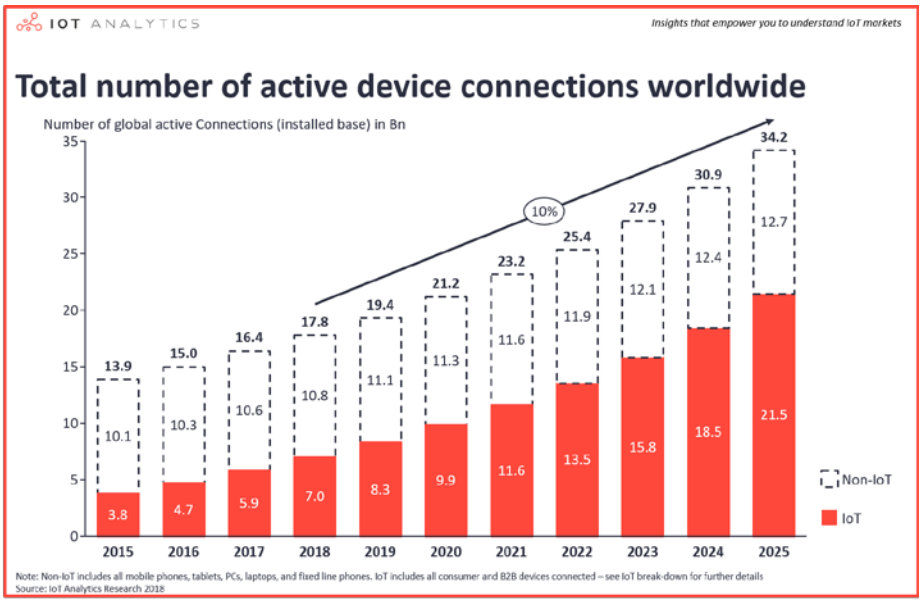
Τα πρωτόκολλα μπορούν να χωριστούν σε διάφορες κατηγορίες, ανάλογα με το είδος ταξινόμησης που επιλέγεται, όπως για παράδειγμα με βάση το σκοπό που εξυπηρετούν ή το επίπεδο (layer) του μοντέλου αναφοράς (Reference Model) OSI [3]. Παρουσιάζουμε στις επόμενες ενότητες τέσσερα προτυποιημένα και ιδιαίτερα δημοφιλή[14] πρωτόκολλα δικτύου για συστήματα IoT: το IPV6 Low Power Wireless Personal Networks (6LoWPAN), το Datagram Transport Layer Security (DTLS), το Constrained Application Protocol (CoAP) και το Message Queuing Telemetry Transport (MQTT). Εξετάζουμε τα χαρακτηριστικά ασφάλειας που ενσωματώνουν εγγενώς στη σχεδίασή τους, αναλύοντας τα όποια ποιοτικά χαρακτηριστικά ασφάλειας υπάρχουν.

3.1 Το πρωτόκολλο 6LoWPAN

Το πρωτόκολλο IPv6 over LowPower Wireless Personal Area Networks (6LoWPAN) ακολουθεί τη λογική ότι πρέπει να μπορεί να τρέχει ακόμα και στις «μικρότερες» συσκευές με λίγους πόρους, οπότε και το ίδιο πρέπει να είναι με μικρές απαιτήσεις σε πόρους. Θεωρείται ιδανικό για εφαρμογές τύπου home automation και smart metering [86].

[14] https://blog.benjamin-cabe.com/2018/04/17/key-trends-iot-developer-survey-2018

Το πρωτόκολλο 6LoWPAN επιτρέπει σε πακέτα πρωτοκόλλου IPv6 να μεταφέρονται αποδοτικά μέσω πλαισίων (frame) μικρού μήκους, όπως αυτά ορίζονται στο πρότυπο IEEE 802.15.4 [15]. Επιπρόσθετα, η χρήση διευθύνσεων IPv6 επιλύει το πρόβλημα του περιορισμένου αριθμού συσκευών με μοναδική διεύθυνση που μπορούμε να έχουμε σε ένα δίκτυο τεχνολογίας IPv4. Το χαρακτηριστικό αυτό είναι ιδιαίτερα σημαντικό, αναλογιζόμενοι ότι μέχρι το 2025 ο αριθμός συνδεδεμένων συσκευών στο Διαδίκτυο θα φτάσει τα 22 δισεκατομμύρια συσκευές, οι οποίες και θα χρειάζονται διευθύνσεις IP για την απευθείας προσπέλαση του Διαδικτύου. Αξιοποιώντας το 6LoWPAN και το IPv6, δε χρειάζεται να υλοποιεί κάποιος μεθόδους όπως περίπλοκα subnetwork ή τεχνικές NAT (Network Address Translation).



Εικόνα 1: Εξέλιξη αριθμού συνδέσεων από συμβατικές συσκευές και συσκευές IoT[15] (εκτίμηση)

Το πρόβλημα που επιλύει το πρωτόκολλο 6LoWPAN και το κάνει ιδιαίτερα δημοφιλές είναι ότι εξαλείφει την ανάγκη για ένα επίπεδο μετάφρασης (για παράδειγμα ένα gateway network system ή ένα application layer λογισμικό), το οποίο ήταν απαραίτητο για τη διασύνδεση τοπικής εμβέλειας δικτύων (για παράδειγμα Bluetoothκαι Zigbee) με κόμβους του Διαδικτύου. Το πρόβλημα δηλαδή που υπήρχε ήταν ότι οι τεχνολογίες μικρής ασύρματης εμβέλειας δεν είχαν τη δυνατότητα από μόνες τους να συνδεθούν με το Διαδίκτυο και είτε να μεταδώσουν δεδομένα σε απομακρυσμένους προορισμούς. Για αυτό το λόγο χρειάζονταν και τα απαραίτητα πρόσθετα στοιχεία και τεχνολογίες ώστε να επιτευχθεί αυτός ο στόχος.Το 6LoWPAN στην ουσία

[15]https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

δημιουργήθηκε ώστε ακόμα και οι μικρότερες συσκευές να μπορούν να είναι συνδεδεμένες απευθείας με το υπόλοιπο Διαδίκτυο.

Μέσα σε ένα δίκτυο 6LoWPAN μπορούμε να βρούμε δύο τύπους συσκευών. Τελικές συσκευές (end devices) και δρομολογητές (router), οι οποίοι είναι υπεύθυνοι να προωθήσουν τα δεδομένα στις τελικές συσκευές. Η σύνδεση των διάφορων συσκευών γίνεται μέσω μίας συσκευής πρόσβασης (access point device, AP) ή ενός δρομολογητή (edge router). Αυτή η συσκευή εκτελεί τρεις λειτουργίες: α) τη μετάδοση πληροφορίας μεταξύ των συσκευών του τοπικού δικτύου 6LoWPAN και του Διαδικτύου (Internet), β) αναμετάδοση της πληροφορίας τοπικά μεταξύ των συσκευών του δικτύου 6LoWPAN και γ) δημιουργία και διατήρηση του τοπικού δικτύου 6LoWPAN.

Το 6LoWPAN προδιαγράφει ένα μηχανισμό «stateless auto configuration» για την αποδοτική διαχείριση του δικτύου. Κάθε κόμβος μπορεί να επιλέξει τη δική του διεύθυνση IPv6 και, στην στατιστικά απίθανη, ακούσια περίπτωση διένεξης διευθύνσεων (address conflict), προδιαγραφεί το μηχανισμό επίλυσης, ώστε να μην υπάρχουν περισσότεροι του ενός κόμβου με την ίδια διεύθυνση IPv6 στο δίκτυο.

Το 6LoWPAN χρησιμοποιεί το μηχανισμό κρυπτογράφησης AES-128 για την ασφάλεια στο επίπεδο συνδέσμου (link layer security). Ο μηχανισμός προδιαγράφεται στο πρότυπο IEEE 802.15.4. Το επίπεδο συνδέσμου προσφέρει ταυτοποίηση συνδέσμου (link authentication) και κρυπτογράφηση. Επιπρόσθετα, μπορεί κάποιος να χρησιμοποιήσει μηχανισμούς προστασίας στο επίπεδο μεταφοράς (transport layer), όπως για παράδειγμα το πρωτόκολλο Transport Layer Security (TLS) για συνδέσεις TCP και το πρωτόκολλο Datagram Transport Layer Security (DTLS) για συνδέσεις UDP, το οποίο εξετάζουμε στην επόμενη ενότητα.

3.2 Το πρωτόκολλο DTLS

Υπάρχουν πολλές εφαρμογές που χρειάζονται την επικοινωνία μέσω UDP και που απαιτούν ασφάλεια στο επίπεδο μεταφοράς (transport layer). Το πρωτόκολλο SSL (παλαιότερα) και το πρωτόκολλο TLS (πλέον) λειτουργούν μέσω TCP και δεν μπορούν να καλύψουν τέτοιες

απαιτήσεις. Το πρωτόκολλο Datagram Transport Layer Security (DTLS) δημιουργήθηκε για να καλύψει αυτό το κενό. Το DTLS περιγράφεται[16] στο IETF RFC 6347.

Το DTLS είναι παρόμοιο με το TLS αλλά προσαρμοσμένο στους περιορισμούς που εισάγουν οι μεταδόσεις μέσω UDP. Όπως και στο TLS έτσι και το DTLS ξεκινά με την εγκαθίδρυση μίας ασφαλούς επικοινωνίας όπου ο client και ο server ανταλλάσσουν τις απαραίτητες πληροφορίες για την πιστοποίηση του καθενός. Αυτό γίνεται με τη διαδικασία χειραψίας (handshake).

Για τις περιπτώσεις όπου κάποιο πακέτο χαθεί, χρησιμοποιούνται μετρητές χρόνου (timer), οι οποίοι καταγράφουν το χρόνο που πέρασε ώσπου να ληφθεί απάντηση από τον παραλήπτη. Εάν ξεπεραστεί ένα χρονικό όριο (timeout), τότε επαναλαμβάνεται η αποστολή του πακέτου DTLS, θεωρώντας ότι το προηγούμενο πακέτο χάθηκε κατά τη μετάδοση από τον αποστολέα στον παραλήπτη.

Το DTLS κρυπτογραφεί ανεξάρτητα κάθε ένα πακέτο, σε αντίθεση με το TLS. Έτσι η δυνατότητα αποκρυπτογράφησης ενός νέου πακέτου δεν επηρεάζεται από την ολική απώλεια κάποιου προηγούμενου, όπως συμβαίνει στο TLS.

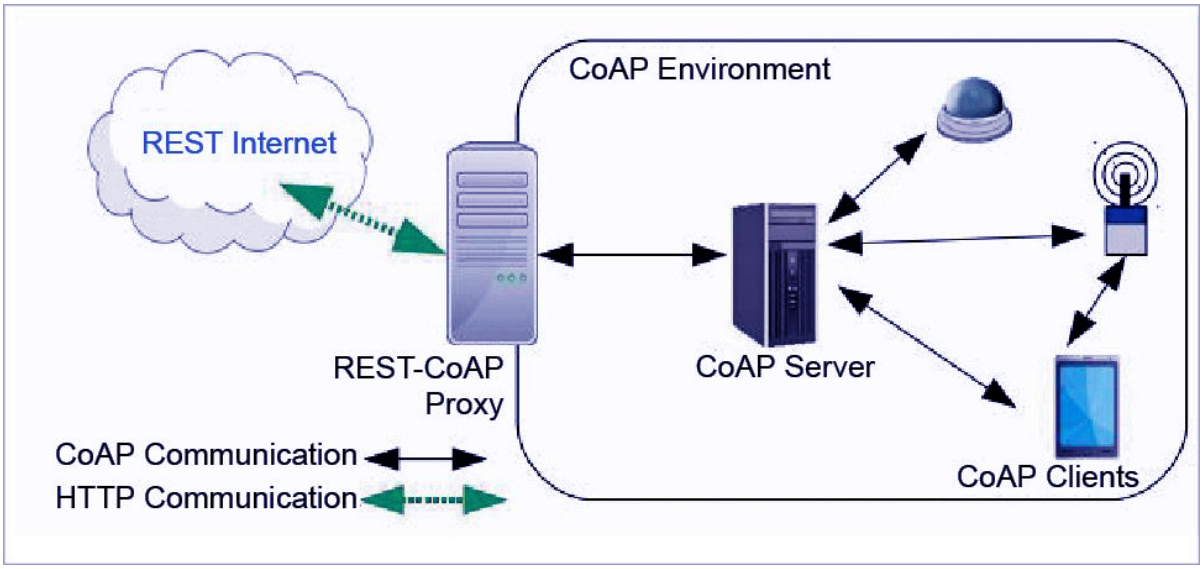
3.3 Το πρωτόκολλο CoAP

Το CoAP (Constrained Application Protocol) είναι ειδικό πρωτόκολλο μεταφοράς δεδομένων[17] μέσω Διαδικτύου για συσκευές με μικρές υπολογιστικές δυνατότητες [3, 7]. Σχεδιάστηκε για επικοινωνία M2M (Machine-to-Machine) και περιγράφεται στο IET FRFC 7252. Βρίσκει πολλές εφαρμογές σε περιβάλλοντα διαχείρισης ενέργειας και οικιακών αυτοματισμών.

Το πρωτόκολλο CoAP βασίζεται στο μοντέλο Representational State Transfer (REST). Δηλαδή οι εξυπηρετητές (server) αναρτούν και κάνουν διαθέσιμες πηγές και δεδομένα σε συγκεκριμένα URL (Uniform Resource Locator)και οι πελάτες (client) έχουν πρόσβαση σε αυτά κάνοντας κλήση των μεθόδων POST, GET, PUT και DELETE, αντίστοιχα με το πρωτόκολλο HTTP που χρησιμοποιείται στον Παγκόσμιο Ιστό (World Wide Web, WWW) [9].

17 https://tools.ietf.org/html/rfc6347
18 https://internetofhomethings.com/homethings/?tag=esp8266-coap

Με τη μέθοδο GET ο client ζητά και παίρνει πληροφορίες. Με τη μέθοδο POST αποστέλλει δεδομένα και με την PUT τα διαθέτει στο δίκτυο, είτε δημιουργώντας νέες καταχωρήσεις είτε ενημερώνοντας υπάρχουσες. Τέλος με την DELETE διαγράφει δεδομένα.



Εικόνα 2: Μοντέλο επικοινωνίας μέσω COAP

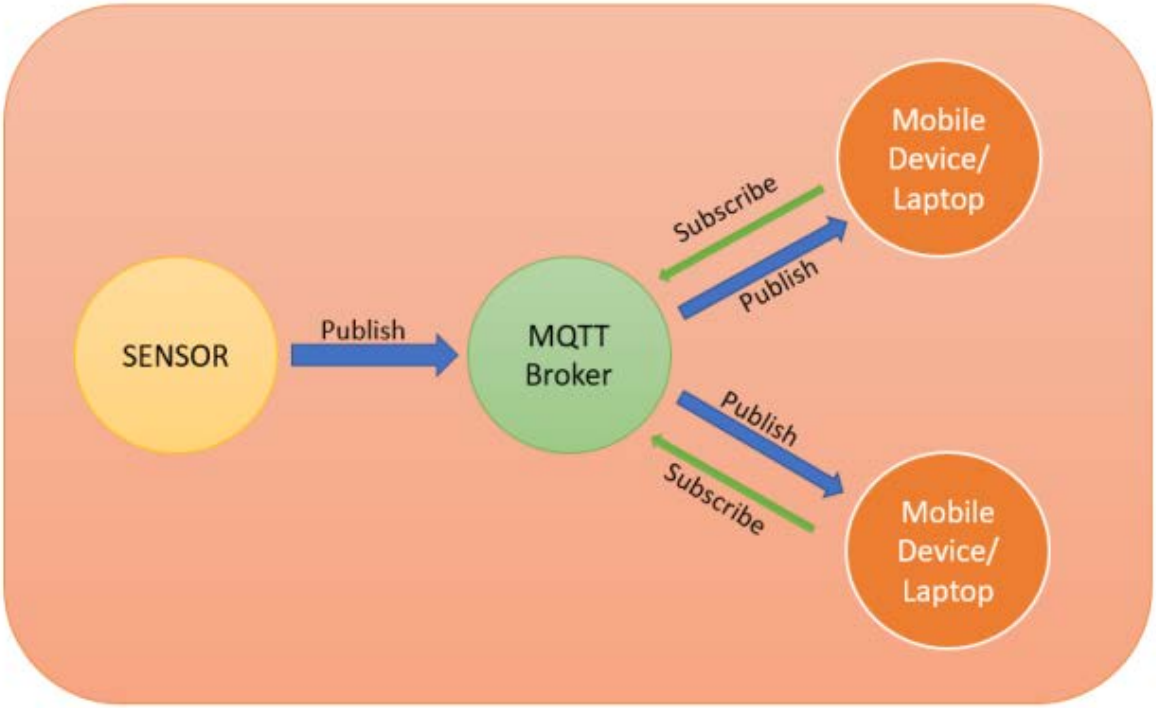
(Πηγή: https://data-flair.training/blogs/iot-protocols/)

3.4 Το πρωτόκολλο MQTT

Ένα από τα πιο διαδεδομένα πρωτόκολλα δικτύου σε εφαρμογές IoT είναι το MQTT [5]. Δημιουργήθηκε αρχικά από τους Andy Stanford-Clark και Arlen Nipper για την παρακολούθηση ενός σωλήνα μεταφοράς πετρελαίου. Οι δημιουργοί του στόχευαν σε ένα πρωτόκολλο, το οποίο θα μπορούσε να έχει χαμηλές απαιτήσεις σε εύρος ζώνης (bandwidth) και να καταναλώνει μικρό ποσό ενέργειας καθώς οι αισθητήρες τροφοδοτούνταν από μπαταρίες.

Το MQTTείναι ιδιαίτερα χρήσιμο γιατί διευκολύνει την επικοινωνία συσκευών, οι οποίες μπορεί να βρίσκονται σε απομακρυσμένες περιοχές, χωρίς αυτό να σημαίνει ότι περιορίζεται μόνο σε τέτοιου είδους εφαρμογές. Το πρωτόκολλο MQTT παρέχει αξιόπιστη μετάδοση και χαμηλές ενεργειακές και επεξεργαστικές απαιτήσεις, καλύπτοντας ένα ευρύ φάσμα σεναρίων λειτουργίας, όπως αισθητήρες που επικοινωνούν μέσω δορυφόρου αλλά και οικιακές συσκευές εντός ενός οικιακού δικτύου.

Το MQTT χρησιμοποιεί μία αρχιτεκτονική publish/subscribe για την αποστολή και λήψη μηνυμάτων. Κάθε αποστολέας όταν δημιουργεί ένα μήνυμα (publish), προσθέτει το θέμα (topic) που αφορά. Υπάρχει ένας κεντρικός κόμβος-διαμεσολαβητής (broker), από όπου περνάνε όλα τα μηνύματα από τους αποστολείς για να καταλήξουν στον προορισμό τους. Κάθε κόμβος που θέλει να λάβει μηνύματα «εγγράφεται» (subscribe) ως συνδρομητής στο θέμα (topic) και λαμβάνει μέσω του broker όλα τα μηνύματα που δημιουργούνται για το συγκεκριμένο θέμα. Η διαδικασία του subscribe και η κατηγοριοποίηση των μηνυμάτων σε θέματα αποτρέπει την άσκοπη λήψη και επεξεργασία από όλους τους κόμβους όλων των νέων μηνυμάτων, τα οποία κοινοποιεί ο broker [6]. Σημειώνεται ότι όλοι οι κόμβοι έχουν μία δικτυακή θύρα (network socket) μόνιμα ανοιχτή με τον broker.



Εικόνα 3: Μοντέλο μετάδοσης μηνυμάτων μέσω MQTT[18]

Ο broker είναι υπεύθυνος κάθε φορά που υπάρχει νέο μήνυμα σε κάποιο topic να το μεταδίδει στους αντίστοιχους συνδρομητές, οι οποίοι είναι εγγεγραμμένοι στο topic. Όπως φαίνεται και στο σχήμα παραπάνω, στο μοντέλο επικοινωνίας του MQTT ο broker είναι το κεντρικό σημείο του συστήματος.

[18] https://amalgjose.com/2018/06/29/what-is-mqtt-and-where-is-it-used-widely/

Όλοι οι client (publisher ή subscriber) χρειάζεται να γνωρίζουν μόνο τη διεύθυνση IP του broker για να επικοινωνήσουν με οποιοδήποτε άλλο client θέλουν. Η μετάδοση των μηνυμάτων αυτών ελέγχεται από το Quality of Service (QoS) , δηλαδή το πόσες φορές θα μεταδοθεί ένα μήνυμα μέσα στο σύστημα.

Οι συνθήκες με τις οποίες γίνεται η μετάδοση είναι:

α) "At most once" (QoS 0), δηλαδή τα μηνύματα μεταδίδονται το πολύ μία φορά. Μπορεί να υπάρξει απώλεια μηνυμάτων αλλά αυτό δεν επηρεάζει το σύστημα καθώς θα ακολουθήσει σε σύντομο χρονικό διάστημα μία νέα μέτρηση και αποστολή μηνύματος.

β) "At least once" (QoS 1), εδώ τα μηνύματα αποστέλλονται παραπάνω από μία φορά ώστε να εξασφαλιστεί η παράδοσή τους στον τελικό προορισμό. Το μήνυμα μεταδίδεται έως ότου παραληφθεί από τον αποστολέα μία επιβεβαίωση λήψης. Υπάρχει όμως το πρόβλημα των διπλών αναρτήσεων-αποστολών καθώς ο αποστολέας ξαναστέλνει το μήνυμα και αν μέσα σε ένα ορισμένο χρονικό διάστημα δεν παραλάβει επιβεβαίωση λήψης του σταλθέντος μηνύματος.

γ) "Exactly once" (QoS 2), χρησιμοποιείται για εφαρμογές, στις οποίες πρέπει να υπάρχει ακριβώς μία αποστολή και λήψη του μηνύματος, όπως για παράδειγμα σε σενάρια με εγχρήματες συναλλαγές, όπου η απώλεια μηνύματος ή οι διπλές αποστολές μπορούν να επηρεάσουν την εφαρμογή. Δηλαδή να οδηγήσουν σε λανθασμένες χρεώσεις.

Ο broker είναι επίσης υπεύθυνος για την προώθηση όλων των μηνυμάτων που ήταν εγγεγραμμένος ο clientσε περίπτωση που ο τελευταίος αποσυνδεθεί για κάποιο χρονικό διάστημα. Τα Qos 1 και 2 παρέχουν αυτή τη δυνατότητα στο σύστημα καθώς αποθηκεύουν σε μια ουρά τα μηνύματα έως ότου αυτά παραδοθούν. Σε περίπτωση που ένας client πρόκειται να αποσυνδεθεί, στέλνει ένα μήνυμα «*lastwill and testament*», το οποίο προωθείται από τον broker στους υπόλοιπους κόμβους, ώστε να γίνει γνωστή η αποχώρηση του συγκεκριμένου κόμβου.

Η εξέλιξη του πρωτοκόλλου παρακολουθείται και ανανεώνεται από τον οργανισμό OASIS,του οποίου και αποτελεί επίσημο πρότυπο[19]. Η τρέχουσα έκδοση του πρωτοκόλλου είναι η 5.0. Προηγούμενες επίσημες και σταθερές εκδόσεις αποτελούν οι MQTT 3.1 και MQTT 3.1.1.

[19] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

3.5 Αξιολόγηση ασφάλειας συστημάτων και πρωτοκόλλων IoT

3.5.1 Ασφάλεια συστημάτων IoT

Τα συστήματα IoTείναι συχνά απευθείας συνδεδεμένα στο ανοικτό Διαδίκτυο αλλά και άμεσα προσβάσιμα από αυτό. Αρκεί μία απλή αναζήτηση σε μηχανές αναζήτησης γενικού σκοπού, όπως για παράδειγμα το Google Searchή το Microsoft Bing, ή πιο εξειδικευμένες, όπως το Shodan[20], ώστε να αποκαλυφθεί η διεύθυνση IP τους [20]. Διαπιστώνεται ότι πολλά προβλήματα ξεκινούν από σφάλματα ή παραλείψεις στην αρχική εγκατάσταση και παραμετροποίηση των συστημάτων, τα οποία καταλήγουν να είναι προσπελάσιμα από το ανοικτό Διαδίκτυο, χωρίς να υπάρχει τέτοια βούληση.

Συστήματα και εφαρμογές IoTπου είναι αναγκαίο να μεταφέρουν δεδομένα σε πραγματικό χρόνο περιγράφονται στο [18]. Η σχετική μελέτη εξετάζει την ασφάλεια καθώς και σενάρια επιθέσεων. Σχετικά αντίμετρα θα μπορούσαν να αναπτυχθούν και σε επίπεδο πρωτοκόλλων IoT. Οι λύσεις που προτείνονται μπορούν να κατηγοριοποιηθούν σε δύο ομάδες. Λύσεις υλισμικού (hardware) και λύσεις λογισμικού (software).Η προσέγγιση που προτείνεται αφορά σε ένα σύστημα «simplex», όπου μέσω ενός κεντρικού ελεγκτή παρακολουθούνται όλες οι συσκευές και συστήματα για μία μη φυσιολογική μέτρηση ή λειτουργία κάποιου κόμβου. Σε περιπτώσεις όπου δεν είναι δυνατή η χρήση λύσεων υλισμικού, προτείνεται η χρήση λογισμικού για καταπολέμηση των επιθέσεων πλάγιου καναλιού (side channel attacks), την μη αιτιοκρατική (non-deterministic) εναλλαγή των διεργασιών σε ένα σύστημα και την χρήση συστημάτων ανίχνευσης εισβολών (intrusion detection system, IDS) σε συστήματα IoT. Κάτι τέτοιο όμως είναι ιδιαίτερα δαπανηρό σε πόρους και δεν είναι εφικτά υλοποιήσιμο σε όλα τα συστήματα.

Προβλήματα λόγω της ετερογένειας και του συνεχόμενα αυξανόμενου αριθμού συσκευών και συστημάτων εντοπίζονται σε «έξυπνες πόλεις» και σε συστήματα μεταφοράς ενέργειας [94]. Και πάντα υπάρχει η ανάγκη σε τομείς όπως η υγεία και ο κατασκευαστικός τομέας τα δεδομένα να παραμένουν ακέραια και να υπάρχει πλήρης προφύλαξη από μη εξουσιοδοτημένες προσβάσεις Παραδοσιακές λύσεις, όπως η κρυπτογράφηση, μπορεί να μην είναι αποδοτικές ή και εφαρμόσιμες λόγω της φύσης των συστημάτων IoT. Οι λύσεις που προτείνονται στο [94]

[20]https://www.shodan.io/

αφορούν συστήματα Software Defined Networking (SDN) και blockchain/distributed ledger technologies, τα οποία μπορούν να βοηθήσουν στον τομέα της ασφαλούς επικοινωνίας και κρυπτογράφησης των δεδομένων.

Στο επίπεδο εφαρμογής παρατηρούνται σοβαρά προβλήματα όπως η αναγνώριση ενός κακόβουλου ή παραβιασμένου κόμβου [95]. Επίσης υπάρχουν προβλήματα με τους μηχανισμούς που θα χρησιμοποιηθούν για έλεγχο στην πρόσβαση και πιστοποίηση των χρηστών-κόμβων. Προκύπτουν προβλήματα με την ενσωμάτωση τέτοιων μηχανισμών στις εφαρμογές και παρατηρούνται σημαντικά κενά ασφαλείας[96]. Χρειάζεται προσοχή στο σχεδιασμό και την επιλογή του κατάλληλου πρωτοκόλλου.

Ένα μεγάλο μέρος της ασφάλειας στα περιβάλλοντα IoT εξαρτάται από το λογισμικό υλοποίησης των πρωτοκόλλων δικτύου που χρησιμοποιούνται για τη μεταφορά της πληροφορίας από κόμβο σε κόμβο. Όπως σε κάθε λογισμικό υπάρχουν σφάλματα, παραλείψεις και αδυναμίες έτσι και στη σχεδίαση και στις υλοποιήσεις των πρωτοκόλλων δικτύου παρατηρούνται τέτοια κενά [65].

3.5.2 Ασφάλεια πρωτοκόλλου 6LoWPAN

Προβλήματα ασφάλειας έχουν εντοπιστεί στο πρωτόκολλο 6LoWPANκαι στα συστήματα με συσκευές που ακολουθούν υλοποιήσεις στοίβας 6LoWPAN [26]. Ένα από τα βασικά ζητήματα που το πρωτόκολλο δεν αντιμετωπίζει [21] εγγενώς είναι η από-άκρο-σε-άκρο (end-to-end) διασφάλιση της ασφάλειας στην επικοινωνία.

Το πρωτόκολλο 6LoWPAN είναι ευάλωτο σε επιθέσεις τύπου Sinkhole [83], Blackhole [22] και Helloflood [83]. Οι επιθέσεις Sinkholeέχουν σαν στόχο την συγκέντρωση όλης της κίνησης μέσω ενός προσβεβλημένου κόμβου σε ένα δίκτυο και την υποκλοπή των δεδομένων που διακινούνται. Στην επίθεση Blackhole ο προσβεβλημένος κόμβος καταστρέφει αντί να αναμεταδώσει τα λαμβανόμενα πακέτα. Στόχος της επίθεσης HelloFlood είναι η εξάντληση της ενέργειας και λοιπών πόρων των συστημάτων IoT, ώστε να μην μπορούν να αποκριθούν είτε να προκληθούν παρεμβάσεις στη δρομολόγηση της δικτυακής τους κίνησης [40].

Οι επιθέσεις έναντι του πρωτοκόλλου 6LoWPAN εστιάζουν και στις λειτουργίες κατακερματισμού και συναρμολόγησης (fragmentation and reassembly) των πακέτων. Δύο

[21] https://www.ietf.org/proceedings/64/slides/6lowpan-1.pdf
[22] https://en.wikipedia.org/wiki/Packet_drop_attack

μέθοδοι για αποφυγή των επιθέσεων fragmentation παρουσιάζονται στο [38]. Στην πρώτη μέθοδο, οι κόμβοι διακρίνουν τα έγκυρα τμήματα (fragment) έναντι των κακόβουλων μέσω μίας κρυπτογράφησης του περιεχομένου των πακέτων με την ταυτότητά τους. Στη δεύτερη μέθοδο, τα πακέτα, τα οποία θεωρούνται ύποπτα λόγω φόρτου του δικτύου, απορρίπτονται από το σύστημα κατά την επεξεργασία και τη λήψη των διάφορων τμημάτων τους.

3.5.3 Ασφάλεια πρωτοκόλλου DTLS

Το πρωτόκολλο DTLS χρησιμοποιείται σε υλοποιήσεις και εφαρμογές για να επιλύσει προβλήματα ασφαλούς μεταφοράς δεδομένων [9]. Χρειάζεται όμως προσοχή διότι αν δεν υλοποιηθεί προσεκτικά κατά τη διάρκεια εισαγωγής και χρήσης σε μία εφαρμογή μπορεί να αφήσει την συσκευή – εφαρμογή ευάλωτη. Επιθέσεις όπως Oracle padding[23], διαρροή του κλειδιού RSA σε τρίτους [5, 31] και επιθέσεις στις διαδικασίες διαπραγμάτευσης (renegotiation attacks)[24] είναι προβλήματα τα οποία χρήζουν προσοχής και αντιμετώπισης [4, 31].

Επιθέσεις σε αδυναμίες που έχουν «κληρονομηθεί» από το TLS στο DTLS επηρεάζουν τις αντίστοιχες υλοποιήσεις [31]. Επιθέσεις στον αλγόριθμο RC4 και η επίθεση triple hand shake παραμένουν [25] επικίνδυνες για παλαιότερες εκδόσεις του DTLS. Η έκδοση στην οποία και βρίσκεται το πρωτόκολλο αυτή τη στιγμή είναι η DTLS 1.2, η οποία και βασίζεται στην έκδοση του πρωτοκόλλου TLS 1.2. Το πρότυπο διατίθεται από τον οργανισμό IETF[26].

3.5.4 Ασφάλεια πρωτοκόλλου CoAP

Το πρωτόκολλο CoAP (Constrained Application Protocol) χρησιμοποιείται ευρέως, ωστόσο δεν παρέχει εγγενώς αξιόπιστους μηχανισμούς ασφάλειας [8]. Για αυτό και προτείνεται η συνδυαστική χρήση με το πρωτόκολλο DTLS για τη μεταφορά δεδομένων μέσω του Διαδικτύου, ώστε να διασφαλίζεται η εμπιστευτικότητα και η ακεραιότητα της πληροφορίας [8]. Η δυνατότητα λειτουργίας έχει επιδειχθεί για συσκευές Google Androidσε συνδυασμό με συσκευές χαμηλών επεξεργαστικών πόρων [9]. Άλλες μελέτες εστιάζουν περαιτέρω στην επίδειξη

[23] https://en.wikipedia.org/wiki/Padding_oracle_attack

[24] http://www.educatedguesswork.org/2009/11/understanding_the_tls_renegoti.html
[25] https://tools.ietf.org/html/rfc7457#section-2.14
[26] https://tools.ietf.org/html/rfc6347

λειτουργίας σε σενάρια αυτοματισμών σπιτιού (home automation) χρησιμοποιώντας συσκευές Arduinoκαι συστήματα Raspberry Pi [16].

Επειδή το CoAP χρησιμοποιεί το πρωτόκολλο UDP για τη μετάδοση των πακέτων του, είναι κατανοητό ότι οι ευπάθειες οι οποίες επηρεάζουν το UDP μπορούν να προσβάλλουν και το CoAP. Παράδειγμα τέτοιας ευπάθειας εξάγεται και από την επίθεση IP spoofing όπου η διεύθυνση IP του αποστολέα αντιγράφεται και ο επιτιθέμενος αναγκάζει συσκευές να αποστέλλουν πακέτα με πλαστή ταυτότητα σε ένα δίκτυο.

Άλλη μία επίθεση είναι το packet amplification όπου, ένα πακέτο το οποίο στάλθηκε από το μηχάνημα θύμα , αντιγράφεται από πολλά άλλα μηχανήματα-κόμβους μέσα σε ένα δίκτυο και τα μηχανήματα που επιτίθενται προσθέτουν δεδομένα μέσα σε αυτό το πακέτο. Σε συνδυασμό με το IP spoofing, όπου η αρχική IP αντιγράφεται από τα μηχανήματα επίθεσης, έχουμε σαν αποτέλεσμα να στέλνεται ένας σημαντικά αυξημένος αριθμός αιτημάτων στον server ο οποίος και απαντά σε όλα αυτά τα αιτήματα προωθώντας τις απαντήσεις στο μηχάνημα θύμα. Έτσι έχουμε και μία μεγάλη αύξηση στην κίνηση του δικτύου αλλά και μια επιτυχημένη επίθεση DDoS στο μηχάνημα θύμα από όλες αυτές τις απαντήσεις [87].Έχει παρατηρηθεί[27] μία αύξηση σε τέτοιου είδους επιθέσεις μέσω του πρωτοκόλλου CoAP σε παγκόσμια κλίμακα .

3.5.5 Ασφάλεια πρωτοκόλλου MQTT

Το πρωτόκολλο MQTT δεν απαιτεί επαλήθευση ταυτότητας (authentication) για τους χρήστες. Η επαλήθευση μέσω μηχανισμών όπως τα password είναι θέμα των υλοποιήσεων των broker. Αν οι broker δεν υλοποιούν τέτοιους μηχανισμούς, τότε ο οποιοσδήποτε μπορεί να συνδεθεί με τον server. Αυτό είναι κάτι που συμβαίνει σε μεγάλο βαθμό στο ανοικτό Διαδίκτυο.

Μία απλή αναζήτηση μέσω της μηχανής αναζήτησης Shodan αποκαλύπτει ότι ένας μεγάλος αριθμός από server παρέχει στοιχεία που δε θα έπρεπε και αφήνει τον οποιοδήποτε client να έχει πρόσβαση σε όλα τα διαθέσιμα topic [64].

Το πρωτόκολλο MQTT δε διαθέτει μηχανισμούς εξουσιοδότησης (authorization). Αυτό επαφίεται στις υλοποιήσεις των broker και στην ουσία τους προγραμματιστές της εφαρμογής-υλοποίησης

[27] https://www.a10networks.com/wp-content/uploads/A10-EB-14115-EN.pdf

του πρωτοκόλλου. Ο οποιοσδήποτε client συνδεθεί σε ένα broker έχει αυτόματα πρόσβαση σε όλα τα topic και μπορεί να λάβει και να αναρτήσει μηνύματα.

Το πρωτόκολλο MQTT δεν προδιαγράφει κάποιο μηχανισμό κρυπτογράφησης για την προστασία της εμπιστευτικότητας και της διαθεσιμότητας κατά τη μετάδοση των πληροφοριών. Και αυτό αποτελεί προαιρετικό θέμα υλοποίησης κάθε broker, εκτός της προδιαγραφής του MQTT, με ό,τι αυτό μπορεί να συνεπάγεται για την ποιότητα της ασφάλειας.

Συνοψίζοντας, αρκεί η γνώση της διεύθυνσης IP ενός MQTT broker, ώστε κάποιος να αποκτήσει πλήρη πρόσβαση σε ένα δίκτυο IoT.

Οι αιτίες και οι λόγοι για τις προβληματικές υλοποιήσεις του πρωτοκόλλου MQTT και ιδιαίτερα των χαρακτηριστικών ασφάλειας αναλύονται στο [42]. Διαπιστώνεται ότι οι επιλογές των προγραμματιστών είναι συνειδητές και αποτέλεσμα άγνοιας, προτιμώντας την αύξηση της λειτουργικότητας και της ευχρηστίας έναντι των μηχανισμών προστασίας και ασφάλειας. Η μειωμένη διαθεσιμότητα επεξεργαστικής ισχύος στα συστήματα IoT επιτείνει αυτή την επιλογή. Επίσης, σε μεγάλες εγκαταστάσεις, όπως σε εταιρικά περιβάλλοντα, όπου εγκαθίστανται μεγάλος αριθμός συσκευών, δυσχεραίνεται το έργο των τμημάτων πληροφορικής τεχνολογίας (IT department) σε ό,τι αφορά στη διαχείριση των συσκευών και την τακτική συντήρηση των διαπιστευτηρίων σύνδεσης (credential).

Ζητήματα υλοποιήσεων του πρωτοκόλλου MQTTαναλύονται και στο [21], όπου διαπιστώνεται η ύπαρξη μίας πληθώρας επιλογών σε διαφορετικές γλώσσες προγραμματισμού. Οι συγγραφείς προτείνουν τον έλεγχο ασφάλειας των υλοποιήσεων αξιοποιώντας τεχνικές fuzz testing με μία προσέγγιση προσαρμοσμένη στη λειτουργία του πρωτοκόλλου MQTT. Η προσέγγιση βασίζεται στη μετάλλαξη πακέτων που έχουν συλληφθεί (capture) από υπάρχοντα, σε λειτουργία δίκτυα MQTT. Σημειώνουμε ότι πέρα από τη θεωρητική μελέτη, οι συγγραφείς δεν παρουσιάζουν κάποια εφαρμογή της προτεινόμενης προσέγγισης αλλά και ούτε πειραματικά αποτελέσματα και ευρήματα για συγκεκριμένες υλοποιήσεις του πρωτοκόλλου MQTT.

3.6 Υλοποιήσεις πρωτοκόλλων IoT

3.6.1 Υλοποιήσεις 6LoWPAN

Contiki 2.6: Το ContikiOS πρόκειται για ένα λειτουργικό σύστημα, το οποίο προορίζεται για συσκευές χωρίς πλεόνασμα υπολογιστικών πόρων[28]. Το λειτουργικό παρέχει και την υλοποίηση του 6LoWPAN για την ασύρματη διασύνδεση όλων αυτών των μικρών «πραγμάτων» του Διαδικτύου. Είναι γραμμένο σε γλώσσα C και ο κώδικάς του είναι ανοικτός για όλους τους προγραμματιστές. Η υλοποίηση λοιπόν του 6LoWPAN μέσα στο Contiki καλείται αυτόματα όταν ένα πακέτο 6LoWPAN λαμβάνεται είτε υπάρχει η ανάγκη για αποστολή ενός πακέτου IPv6.

RiotOS: Το RiotOS είναι ένα λειτουργικό σύστημα στοχευμένο και εξελιγμένο πάνω στον κόσμο του IoT [29] . Υποστηρίζει τις περισσότερες συσκευές IoT και τις αρχιτεκτονικές των μικροεπεξεργαστών. Γίνεται προσπάθεια να συμπεριληφθούν όλα τα πρότυπα για τον κόσμο του IoT ώστε να μπορεί να στηρίξει όλα τα «πράγματα» του Διαδικτύου. Μέσα σε αυτά τα πρότυπα και η υλοποίηση για το 6LoWPAN.

TinyOS: Όπως αναφέρεται και στην επίσημη ιστοσελίδα[30] του λειτουργικού «*πρόκειται για ένα λειτουργικό ανοικτού κώδικα, προορισμένο για ασύρματες συσκευές χαμηλών ενεργειακών αναγκών όπως τα δίκτυα αισθητήρων, τα έξυπνα κτήρια και οι έξυπνοι μετρητές*». Μέσα στις λειτουργίες που προσφέρει συγκαταλέγεται και μία πλήρης υποστήριξη 6LoWPAN IPv6 stack. Το TinyOS επίσης παρέχει δυνατότητες για προγραμματισμό εκτέλεσης διαδικασιών, έλεγχο ενέργειας μικροελεγκτών, σειριακή επικοινωνία, πρωτόκολλα πακέτων και πολλά άλλα.

Ακολουθεί ο συγκριτικός πίνακας για τις υλοποιήσεις στον οποίο και φαίνονται οι απαιτήσεις των υλοποιήσεων σε πόρους μνήμης καθώς και η ικανότητά τους να μπορούν να χρησιμοποιηθούν ολόκληρα ή κομμάτια αυτών σε άλλες εφαρμογές ως προσθήκες (modularity).

Θα παρατηρήσουμε ότι το RiotOS έχει ένα προβάδισμα σε σχέση με τα άλλα στο να χρησιμοποιηθεί και ενσωματωθεί πιο εύκολα σε διάφορα περιβάλλοντα με χαμηλούς πόρους και

[28] http://contiki.sourceforge.net
[29] https://riot-os.org/
[30] http://www.tinyos.net/

για τις πολύ μικρές απαιτήσεις σε μνήμη αλλά και για το γεγονός ότι μπορούμε να το διαχειριστούμε και τμηματικά πιο εύκολα από τις άλλες εκδόσεις.

OS	Min RAM	Min ROM	C Support	C++ Support	Modularity
Contiki	< 2kB	< 30kB	●	✗	●
Tiny OS	< 1kB	< 4KB	✗	✗	✗
Riot OS	~1.5KB	~5KB	✓	✓	✓

Πίνακας 1. ΠηγήRiot OS Website

Full support ✓ Partial support ● No support ✗

3.6.2 Υλοποιήσεις DTLS

Υπάρχουν αρκετές βιβλιοθήκες, οι οποίες υλοποιούν το πρωτόκολλο DTLS. Οι περισσότερες ακολουθούν ήδη τη δεύτερη έκδοση του DTLS 1.2 το οποίο και στηρίζεται και ακολουθεί την εξέλιξη του TLS 1.2.

Ανάμεσα στις βιβλιοθήκες που υποστηρίζουν το DTLS βρίσκουμε τις εξής:

Botan: Μία βιβλιοθήκη[31] γραμμένη σε γλώσσα C++. Παρέχει τη δυνατότητα κρυπτογράφησης δεδομένων σε διάφορα πρωτόκολλα όπως SSL, TLS και φυσικά DTLS.

OpenSSL: Η βιβλιοθήκη[32] είναι ανοικτού κώδικα και παρέχει υλοποιήσεις SSL και TLS. Μία πολύ βασική βιβλιοθήκη, στην οποία έχουν βασιστεί και έχουν αναπτυχθεί και άλλες βιβλιοθήκες και υλοποιήσεις.

PyDTLS: Μία υλοποίηση[33] του DTLS σε Python. Είναι μία υλοποίηση που ακολουθεί τη λογική του SSL module. Απλά περνάμε UDP socket στην συνάρτηση *wrap_socket* αντί να τα μεταδώσουμε μέσω TCP. Ο στόχος της βιβλιοθήκης είναι η προσαρμοστικότητα και η υποστήριξη μεγάλου εύρους συστημάτων. Για αυτό και πέρα από την Pythonn υλοποίηση έχει δημιουργηθεί από το OpenSSL.

TinyDTLS: Βιβλιοθήκη[34] η οποία είχε σαν γνώμονα την υποστήριξη για κρυπτογράφηση της επικοινωνίας σε συσκευές με μικρή μνήμη. Είναι μία ελαφριά υλοποίηση για συστήματα με 100 KB μνήμης τύπου flash και 10 KB RAM. Μπορεί να καλύψει τις ανάγκες συστημάτων τύπου server και client. Είναι γραμμένη σε γλώσσα C και υποστηρίζει τις υποχρεωτικές εκδόσεις πρωτοκόλλων κρυπτογράφησης που απαιτούνται ως οι ελάχιστες από το πρότυπο.

MbedTLS: Μία βιβλιοθήκη[35], η οποία παρέχει υλοποιήσεις SSL και TLS. Ο στόχος της ανάπτυξης ήταν η ενσωμάτωση σε περιβάλλοντα και δίκτυα με μικρές συσκευές.

Επίσης κάθε κομμάτι αυτής της βιβλιοθήκης μπορεί να χρησιμοποιηθεί ξεχωριστά από τα υπόλοιπα, κάτι που την κάνει πολύ ευέλικτη. Έτσι κάποιος μπορεί να χρησιμοποιήσει αποκλειστικά τις συναρτήσεις κρυπτογράφησης ή τα πρωτόκολλα μεταφοράς. Άλλο ένα πλεονέκτημα της βιβλιοθήκης είναι ότι επειδή είναι γραμμένη σε γλώσσα C μπορεί να τρέξει στα περισσότερα λειτουργικά συστήματα και αρχιτεκτονικές.

[31] https://botan.randombit.net/handbook/
[32] https://www.openssl.org/
[33] https://github.com/rbit/pydtls
[34] https://github.com/eclipse/tinydtls
[35] https://tls.mbed.org/

3.6.3 Υλοποιήσεις COAP

LibCoap: Μία βιβλιοθήκη[36] γραμμένη σε γλώσσα C, προσαρμοσμένη για συσκευές με χαμηλούς υπολογιστικούς πόρους. Οι συσκευές που προστίθενται μέσα στο δίκτυο ονομάζονται κόμβοι. Η βιβλιοθήκη παρέχει ένα απλό τρόπο προσθήκης νέων κόμβων στο δίκτυο και ο προγραμματιστής απλά πρέπει να επιλέξει ποιές λειτουργίες θέλει να έχει αυτός ο κόμβος οι οποίες θα υλοποιούνται από ένα χειριστή (handler).

MicroCoap: Μία βιβλιοθήκη [37] γραμμένη σε γλώσσα C, όπου σχεδιάστηκε με γνώμονα μικροελεγκτές τύπου Arduino. Η υλοποίηση είναι για server.Είναι μία υλοποίηση ανοικτού κώδικα και είναι διαθέσιμη στην πλατφόρμα GitHub.

CantCoap: Άλλη μία υλοποίηση[38] σε γλώσσα C με γνώμονα συσκευές με λίγους υπολογιστικούς πόρους. Δίνει έμφαση στην απλότητα της λειτουργίας. Η βιβλιοθήκη παρέχει μόνο τον κώδικα για δημιουργία και «αποσυναρμολόγηση» των πακέτων PDU's (Protocol Data Units). Επαφίεται στον προγραμματιστή να εξασφαλίσει τις ενέργειες για επαναποστολή πακέτων και παρακολούθηση ονομασίας πακέτων. Σε ένα σύστημα όπου υπάρχουν λίγοι αισθητήρες όπου στέλνουν κάθε 15 λεπτά μία μέτρηση δεν είναι δύσκολη η παρακολούθησή των μηνυμάτων. Έτσι δε χρειάζεται ένα πλήρες πολύπλοκο περιβάλλον του CoAP.

Lobaro Coap: Μία πλήρης υλοποίηση[39] που σχεδιάστηκε για ενσωματωμένα συστήματα, όπως για παράδειγμα το ARM Cortex ESP8266. Μπορεί να χρησιμοποιηθεί σε κάθε περιβάλλον όπου υποστηρίζεται η γλώσσα C. Υποστηρίζει server και client σε μία υλοποίηση και βρίσκεται και σε πειραματικό στάδιο η υποστήριξη για Arduino.

Node-Coap: Η βιβλιοθήκη [40] είναι σε γλώσσα Javascript. Η δημιουργία των κόμβων και των πηγών δε γίνεται αυτόματα από τον κώδικα της υλοποίησης αλλά αφήνεται στην εφαρμογή και τον προγραμματιστή. Δεν παρέχεται ένας εύκολος τρόπος δημιουργίας νέων κόμβων όπως συμβαίνει σε άλλες υλοποιήσεις.

[36] https://libcoap.net/
[37] https://github.com/1248/microcoap
[38] https://github.com/staropram/cantcoap
[39] https://github.com/lobaro/lobaro-coap
[40] https://github.com/mcollina/node-coap

NCoap: Υλοποίηση[41] του CoAP σε Java. Η υλοποίηση είναι κατάλληλη για περιβάλλοντα τα οποία μπορούν να τρέξουν κώδικα Java. Σε αυτά συγκαταλέγονται και πολλά συστήματα λειτουργικού Android. Η υλοποίηση προσφέρει την παρακολούθηση πηγών CoAP, blockwise μεταφορά[42] σύμφωνα με το πρότυπο IETF RFC 7959, χωρίς όμως να υποστηρίζει DTLS.

CoAPthon: Μία βιβλιοθήκη[43] που είναι γραμμένη σε γλώσσα Python. Προσφέρει τη δημιουργία serverκαι client. Μπορεί να υποστηρίξει Blockwise μεταφορά αλλά δεν υποστηρίζει ακόμα DTLS.

Για λόγους πληρότητας, αναφέρουμε στις επόμενες παραγράφους και ορισμένες εμπορικά διαθέσιμες υλοποιήσεις του πρωτοκόλλου CoAP.

OneMPOWER (INTERDIGITAL): Η εταιρεία μέσω της πλατφόρμας της oneMporwer έφτιαξε μια υλοποίηση CoAP η οποία μπορεί να συνδεθεί με οποιαδήποτε εφαρμογή IoT σε περιβάλλοντα CoAP. Η υλοποίηση παρέχεται μέσω της Machine-to-Machine (M2M) πλατφόρμας παροχής υπηρεσιών (Service Delivery Platform, SDP). Μέσα στις παροχές της πλατφόρμας είναι και η διαδικασία αναβάθμισης και ενημέρωσης του firmwaretων συσκευών των χρηστών. Επίσης δίνεται η δυνατότητα του ελέγχουν των πόρων και των συσκευών που είναι συνδεδεμένες στην πλατφόρμα καθώς και η παρακολούθηση ξεχωριστά κάθε συσκευής, όπως για παράδειγμα όνομα, ημερομηνία δημιουργίας και περιγραφή της συσκευής.

Thethings.IO: Πρόκειται για μία πλατφόρμα, η οποία παρέχει τη δυνατότητα της σύνδεσης backend συστημάτων ανεξάρτητα από το hardware, το οποίο χρησιμοποιείται στην εκάστοτε εφαρμογή. Η πλατφόρμα παρέχει υποστήριξη σε πολλά πρωτόκολλα επικοινωνίας και ανάμεσα σε αυτά είναι και το CoAP. Οι χρήστες της πλατφόρμας έχοντας τη συσκευή την οποία θέλουν να συνδέσουν φτιάχνουν έναν κόμβο μέσω της πλατφόρμας. Η σύνδεση σε αυτόν τον κόμβο γίνεται μέσω του API και πιστοποιείται με ένα κλειδί-token το οποίο και χρησιμοποιείται σε κάθε κλήση από τη συσκευή σε αυτόν τον κόμβο του cloud της υπηρεσίας.

Στον πίνακα που ακολουθεί συγκρίνονται διάφορες υλοποιήσεις του πρωτοκόλλου CoAP. Οι υλοποιήσεις βασίζονται σε διάφορες γλώσσες προγραμματισμού. Κάτι το οποίο αναδεικνύει και τις πολλές επιλογές που υπάρχουν ανάλογα με τις ανάγκες κάθε εφαρμογής. Στον κόσμο του IoT μπορεί ο καθένας να φέρει στα μέτρα του και να χρησιμοποιήσει οποιοδήποτε εργαλείο έχει στη διάθεσή του. Με λίγη αναζήτηση μπορούν να βρεθούν υλοποιήσεις είτε server είτε client για

[41] https://github.com/okleine/nCoAP
[42] https://datatracker.ietf.org/doc/rfc7959/
[43] https://github.com/Tanganelli/CoAPthon

σχεδόν όλα τα περιβάλλοντα-λειτουργικά συστήματα, τα οποία χρησιμοποιούνται σε διάφορες εφαρμογές.

Οι υλοποιήσεις που αναζητήσαμε και βρήκαμε είναι ανοικτού κώδικα και ελεύθερες για να παρέμβει ο καθένας στον κώδικα ώστε να μπορεί να κάνει την όποια αλλαγή επιθυμεί. Υπάρχουν και κλειστού κώδικα, εμπορικές υλοποιήσεις αλλά είναι πολύ εύκολο για μη εμπορικές εφαρμογές να εντοπιστούν υλοποιήσεις δωρεάν πράγμα πολύ χρήσιμο και βολικό για οικιακές εφαρμογές και ερασιτέχνες προγραμματιστές.

Υλοποίηση	**Γλώσσα Προγραμματισμού**	**Server / Client**	**Πλατφόρμες**	**Άδεια Χρήσης**
Libcoap	C	Server, Client	Contiki, ,TinyOS, POSIX	BSD/GPL
Microcoap	C	Server	Arduino	OpenSource (MIT license)
Cantcoap	C	Server , Client	Συσκευές που υποστηρίζουν Linux	BSD
LobaroCoap	C	Server, Client	Embedded Systems, C supporting systems	OpenSource (MIT license)
Node-Coap	Javascript	Server, Client	Node.js	OpenSource (MIT license)
NCoap	Java	Server, Client	Java συσκευές Android	BSD

CoAPthon	Python	Server, Client	Περιβάλλοντα που τρέχουν Python	OpenSource (MIT license)

Πίνακας 2. Σύγκριση υλοποιήσεων Coap

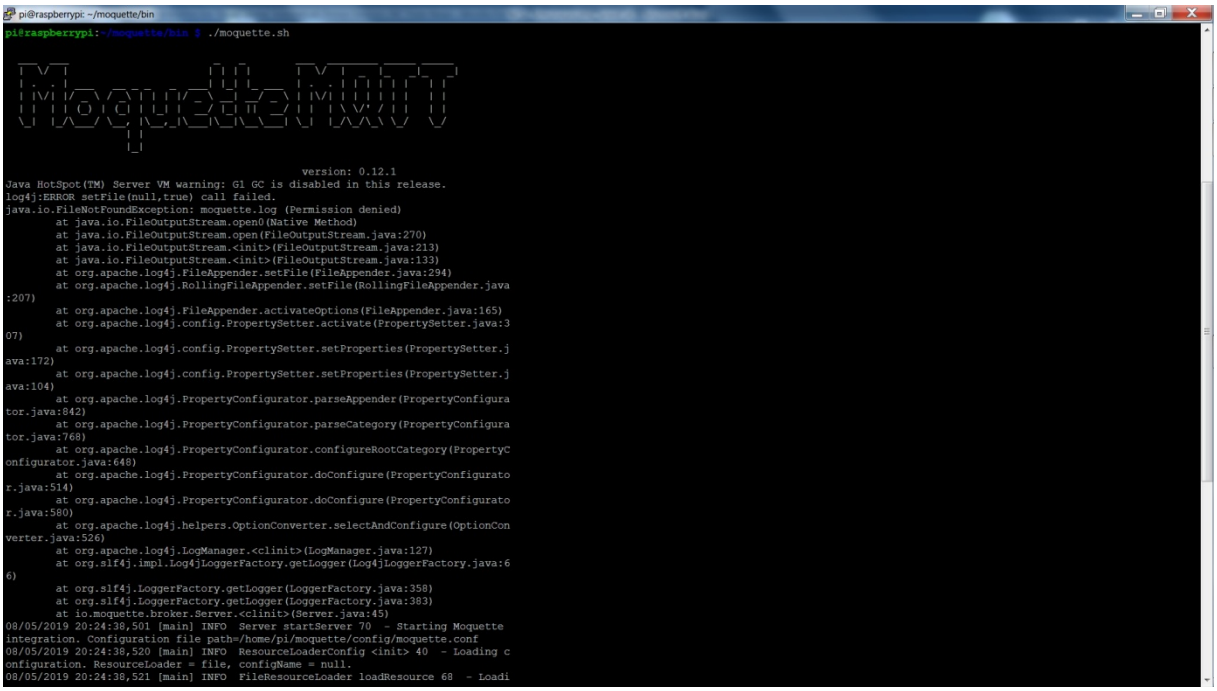
3.6.4 Υλοποιήσεις MQTT

Λόγω της μεγάλης διείσδυσης του IoT κόσμου στην καθημερινότητα και του μεγάλου ενδιαφέροντος από πολλούς κλάδους εμπορικούς και μη, μπορεί κανείς να βρει υλοποιήσεις σε διάφορες προγραμματιστικές γλώσσες. Η υποστήριξη υπάρχει από διάφορα λειτουργικά συστήματα (Windows , Linux, Android κ.λπ.).

Mosquito: Το mosquito είναι ένας open source broker βασισμένος στο πρωτόκολλο MQTT. Υποστηρίζει και τις τρεις μεθόδους μετάδοσης μηνυμάτων(Quality of Service). Έχει υλοποιηθεί σε αρκετές γλώσσες προγραμματισμού και μπορεί να τον χρησιμοποιήσει κανείς και σε περιβάλλοντα Linux, Windows και Android.

Bevywise MQTT Broker: Μία υλοποίηση γραμμένη σε C και Python, η οποία υποστηρίζει τις εκδόσεις 3.1 και 3.1.1 του πρωτοκόλλου MQTT. Είναι μία υλοποίηση που παρέχει πολλές λειτουργίες για τους χρήστες. Ο πυρήνας του broker είναι γραμμένος σε C με στόχο την ταχύτερη απόκριση του server σε αιτήσεις. Τα υπόλοιπα πρόσθετα που μπορούν να συνδεθούν και να χρησιμοποιηθούν μαζί με τον broker μπορούν να υλοποιούνται σε Python, όπως για παράδειγμα η σύνδεση με οποιαδήποτε μηχανή επεξεργασίας μεγάλων δεδομένων (big data processing engines), γραφικά περιβάλλοντα χρήσης και άλλα. Υποστηρίζει πλήρως το πρωτόκολλο MQTT σύμφωνα με τις προδιαγραφές MQTT 3.1 και 3.1.1.Επίσης υποστηρίζει όλα τα επίπεδα «Παροχής Υπηρεσίας» (QoS 0, 1 και 2) καθώς και τις λειτουργίες WILL, Last Retain και wild card subscription. Μπορεί να υποστηρίξει σύνδεση με οποιαδήποτε βάση, όπως MongoDB, για την αποθήκευση δεδομένων μέσω ειδικού προσθέτου. Τέλος παρέχει προστασία και κρυπτογράφηση δεδομένων μέσω TLS/SSL.

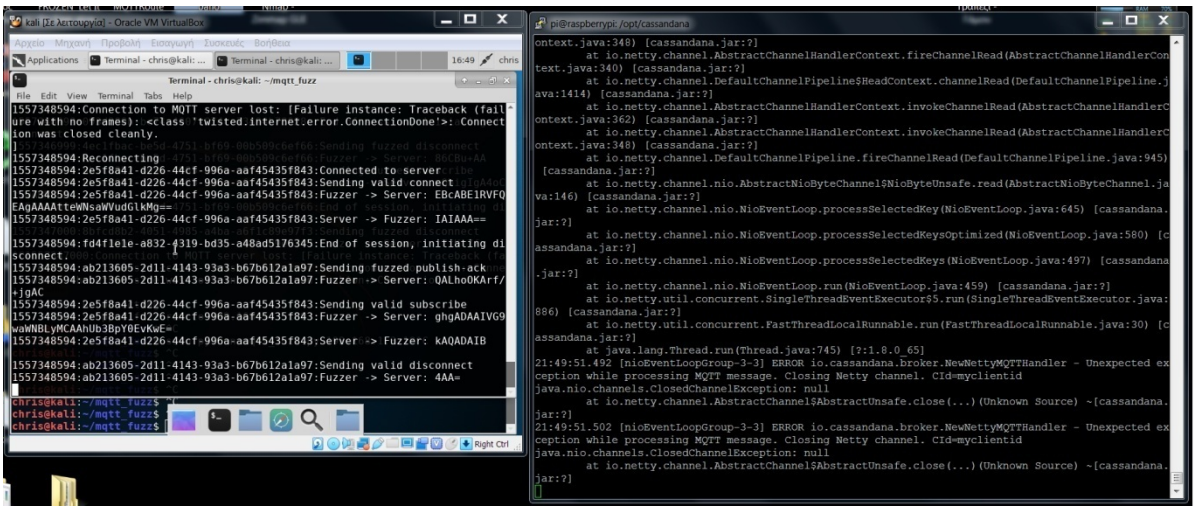
Moquette : Ο Moquette[44] είναι μια υλοποίηση MQTT Broker σε Java. Χρησιμοποιεί τη βιβλιοθήκη Netty[45] για την υλοποίηση του πρωτοκόλλου. Ένα μεγάλο πλεονέκτημα του Moquette broker είναι το γεγονός ότι μπορεί να χρησιμοποιηθεί σε πολλές εφαρμογές ως ενσωμάτωση. Δηλαδή να χρησιμοποιηθεί ο κώδικάς του μέσα σε οποιαδήποτε τρίτη εφαρμογή, η οποία απαιτεί την ύπαρξη ενός MQTT server, εξαλείφοντας την ανάγκη για ανάπτυξη ενός ξεχωριστού broker.



Εικόνα 4: Ο Broker Moquette σε λειτουργία

Cassandana: Η υλοποίηση Cassandana[46] του MQTT αναπτύχθηκε σε Java. Ο broker αυτός υποστηρίζει κρυπτογράφηση μέσω TLS/SSL και θα μπορούσε να χρησιμοποιηθεί σε εμπορικές εφαρμογές. Μπορεί να συνδεθεί με βάσεις όπως MongoDB, MySQLκαι Postgre SQL. Υποστηρίζει memory cache μέσα στην υλοποίηση, ώστε να μειώνονται οι απαιτήσεις για ενέργειες εισόδου/εξόδου (Input/Output, I/O). Επίσης υπάρχει η δυνατότητα αρχειοθέτησης των μηνυμάτων που μεταδίδονται με το εργαλείο Silo[47]. Τέλος υποστηρίζει Websocket. Τα WebSocket είναι ένα πρωτόκολλο επικοινωνίας που παρέχει τη δυνατότητα πλήρους αμφίδρομης επικοινωνίας (full-duplex) μέσω μίας ενεργής σύνδεσηςTCP.

[44] https://github.com/moquette-io/moquette
[45] https://netty.io/
[46] https://github.com/mtsoleimani/cassandana/
[47] https://github.com/mtsoleimani/silo



Εικόνα 5: Ο Broker Cassandana δεξιά και δίπλα ο fuzzer σε Kali Linux.

PAHOMQTT: Ένας client για το πρωτόκολλο MQTT που αναπτύχθηκε από την Eclipse[48]. Υπάρχει υλοποιημένος σε πολλές γλώσσες για χρήση σε πολλά περιβάλλοντα και εφαρμογές. Υλοποιήθηκε για να λειτουργεί σε συνεργασία με τον Mosquitto broker αλλά και με οποιονδήποτε άλλο MQTT Broker.

Στον ακόλουθο πίνακα βλέπουμε και τις υλοποιήσεις που εξετάσαμε. Οι γλώσσες προγραμματισμού ποικίλουν καθώς και τα περιβάλλοντα (λειτουργικά συστήματα) στα οποία μπορούν να τρέξουν.

Υλοποίηση	Γλώσσα Προγραμματισμού	Server / Client	Περιβάλλον / Λειτουργικό σύστημα	Άδεια Χρήσης
Mosquitto	C	Broker, Client	Linux, Windows, C90, Unix, MacOS, Raspberry Pi	BSD
Moquette	C	Broker	Linux , Windows Raspberry Pi ,	

[48] https://www.eclipse.org/paho/

Bewywise	C, Python	Broker, Client	Linux, Windows, Unix, MacOS, Raspberry Pi	Commercial License
Cassandana	Java	Broker	Linux, FreeBSD, MacO , Windows	
PahoMqtt	C/C++ , Python , Java, Javascript, .Net(C#)	Client	Linux , MacOS , Winodws	Open Source (BSD)

Πίνακας 3. Σύγκριση υλοποιήσεων πρωτοκόλλου MQTT

Κεφάλαιο 4
Πειραματικός Έλεγχος Ασφάλειας Υλοποιήσεων Πρωτοκόλλου MQTT

Η μελέτη της σύγχρονης ακαδημαϊκής βιβλιογραφίας και η θεωρητική ανάλυση των χαρακτηριστικών ασφάλειας που (δεν) ενσωματώνουν τα δημοφιλή πρωτόκολλα δικτύου για περιβάλλοντα IoT αναδεικνύει ένα σημαντικό πρόβλημα. Υπάρχουν πολλές υλοποιήσεις των πρωτοκόλλων διαθέσιμες και ευρέως χρησιμοποιούμενες. Διαφαίνεται ωστόσο ότι τόσο ενδογενείς όσο και εξωγενείς παράγοντες έχουν περιορίσει σημαντικά τον έλεγχο της ασφάλειας που παρέχουν, πριν αυτά διατεθούν ελεύθερα. Καθώς όμως όλες και περισσότεροι κατασκευαστές ενσωματώνουν αυτές τις υλοποιήσεις στα προϊόντα IoT που προσφέρουν, καθίσταται ιδιαίτερα κρίσιμο να ελεγχθεί η ποιότητα του λογισμικού και ειδικότερα η ανθεκτικότητά του σε διάφορα σενάρια κακόβουλων επιθέσεων.

Το παρόν κεφάλαιο προσεγγίζει το ζήτημα αυτό αξιοποιώντας τεχνικές fuzz testing για τον έλεγχο συστημάτων IoT που χρησιμοποιούν τεχνολογίες ελεύθερου λογισμικού και λογισμικού ανοικτού κώδικα (ΕΛ/ΛΑΚ) στην υλοποίηση του πρωτοκόλλου MQTT.

4.1 Εργαλεία Ελέγχου Ασφάλειας

Επειδή ο κόσμος του IoT δε διαφέρει πάρα πολύ από τις μέχρι τώρα εφαρμογές Διαδικτύου, πολλά από τα εργαλεία για τον έλεγχο της ασφάλειας εφαρμογών Διαδικτύου θα μπορούσαν να χρησιμοποιηθούν και για τον έλεγχο της ασφάλειας σε εφαρμογές IoT. Τα εργαλεία που είναι διαθέσιμα είναι πολλά και υπάρχουν λύσεις και από ιδιωτικές εταιρείες, οι οποίες προσφέρουν πλήρη υποστήριξη για τον έλεγχο εφαρμογών και συστημάτων IoT. Στον παρόν κεφάλαιο εστιάζουμε σε λύσεις ΕΛ/ΛΑΚ, στις οποίες μπορεί να έχει πρόσβαση ο καθένας. Ένα τέτοιο

παράδειγμα είναι η πλατφόρμα Metasploit[49], η οποία παρέχει πολλά εργαλεία στο εσωτερικό της για δοκιμές πάνω σε εφαρμογές Διαδικτύου. Ενσωματώνει επίσης εργαλεία για ανάλυση κίνησης δικτύου, εύρεση μυστικών κωδικών πρόσβασης (password) και έλεγχο σε πρωτόκολλα επικοινωνίας. Ένα μεγάλο πλεονέκτημα της συγκεκριμένης πλατφόρμας είναι και η βάση γνώσης, στην οποία έχει πρόσβαση και μπορεί να ελέγχει για γνωστές ευπάθειες του συστήματος ή εφαρμογής προς έλεγχο. Μέσω έτοιμων προγραμμάτων μπορεί κάποιος να ελέγξει το σύστημά του για ήδη γνωστά κενά ασφάλειας, με έτοιμα σενάρια επίθεσης εναντίων της εφαρμογής του.

Σημαντικό ρόλο παίζουν και τα προγράμματα που υλοποιούν επιθέσεις «ωμής βίας» (brute force attacks) για εύρεση μυστικών κωδικών, όπως το εργαλείο John The Ripper[50]. Αυτό είναι ένα εργαλείο με εκδόσεις για λειτουργικά συστήματα Linux και Microsoft Windows, το οποίο μπορεί να χρησιμοποιηθεί αποτελεσματικά για την εύρεση αδύναμων password. Η ύπαρξη τέτοιων password είναι ένα συχνό φαινόμενο στον κόσμο του IoT.

Το Rainbow Crack [51] είναι ένα hash password cracker. Με βάση πίνακες με γνωστά κρυπτογραφημένα password, το RainbowCrack προσπαθεί να εντοπίσει το password μίας εφαρμογής. Το Brutus είναι διαθέσιμο για συστήματα Microsoft Windows και μπορεί να ενεργεί επιθέσεις σε 60 στόχους ταυτόχρονα καθώς και να κάνει online δοκιμές password. Κομμάτια του κώδικα του botnet Mirai χρησιμοποιήθηκαν και για το σπάσιμο αρχικών ή αδύναμων password. Είναι πολύ σημαντικό το κατά πόσο μπορούν να αντισταθούν οι εφαρμογές IoT και τα πρωτόκολλα σε τέτοιου είδους επιθέσεις κατά των συστημάτων ταυτοποίησης των χρηστών.

4.2 Εργαλεία Fuzz Testing

Τα εργαλεία fuzz testing (fuzzing) είναι προγράμματα, τα οποία μας επιτρέπουν να αυτοματοποιήσουμε την διαδικασία ελέγχου και να εξετάσουμε όσο το δυνατόν παραπάνω σενάρια και κακόβουλες εισόδους στα σημεία ελέγχου. Fuzzers όπως τα AspFuzz [90], AutoFuzz [91] και SecFuzz [92] είναι παραδείγματα τέτοιων προγραμμάτων που έχουν χρησιμοποιηθεί σε δοκιμές για ανεύρεση σφαλμάτων σε πρωτόκολλα επικοινωνίας IoT.

Τα SecFuzz και AutoFuzz καταγράφουν τα μηνύματα τα οποία μεταδίδονταν στο δίκτυο και προσθέτουν ή μεταβάλλουν τα πακέτα δικτύου, στέλνοντας αλλοιωμένα μηνύματα στα σημεία

[49] https://www.rapid7.com/products/metasploit/download/
[50] https://www.openwall.com/john/
[51] http://project-rainbowcrack.com

εισόδου ενός συστήματος. Το AspFuzz είναι ένα fuzzer, το οποίο χρησιμοποιεί εκ των προτέρων γνωστές προδιαγραφές ορισμένων πρωτοκόλλων και δημιουργεί μηνύματα ειδικά προσαρμοσμένα στα πρότυπα των πακέτων μετάδοσης. Το «κακόβουλο» κομμάτι του μηνύματος ενσωματώνεται στο πακέτα και στη συνέχεις αποστέλλεται σαν ένα κανονικό πακέτο στο σημείο εισόδου.

Το Sulley fuzzer[52] είναι ένα δημοφιλές πρόγραμμα fuzzing, το οποίο εκτός από την παραγωγή και δοκιμή των test cases διαθέτει πρόσθετα για την παρακολούθηση του δικτύου και καταγραφή πακέτων και δεδομένων καθώς και δυνατότητα για παράλληλο έλεγχο σε πολλαπλά σημεία ελέγχου.

4.2 Δημιουργία Περιβάλλοντος Δοκιμών

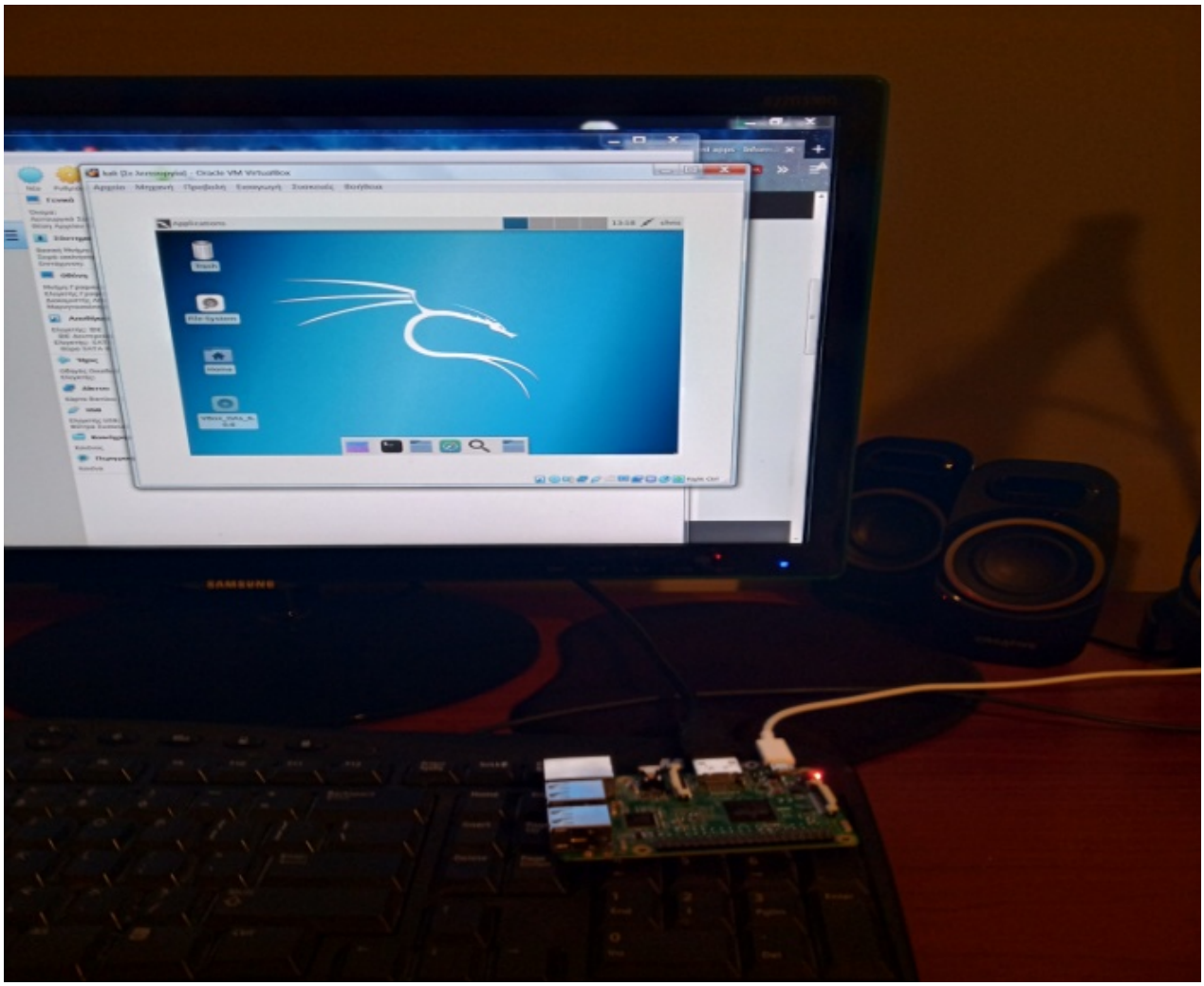
Είδαμε ότι ένα μεγάλο κομμάτι του IoT έχει εφαρμογή σε αυτοματισμούς και οικιακά περιβάλλοντα. Απευθύνεται δηλαδή σε τελικούς καταναλωτές (consumer facing). Για το λόγο αυτό επιλέξαμε να εστιάσουμε στη δημιουργία ενός περιβάλλοντος δοκιμών, το οποίο να είναι σε θέση να διαμορφώσει ένας χρήστης με βασικές γνώσεις διαχείρισης υπολογιστικών συστημάτων και οικιακών δικτύων. Προς αυτή την κατεύθυνση, επιλέξαμε να βασιστούμε αποκλειστικά σε ελεύθερο λογισμικό και λογισμικό ανοικτού κώδικα (ΕΛ/ΛΑΚ), ώστε να μην υπάρχει επιπλέον οικονομική επιβάρυνση για τον τελικό καταναλωτή.

Με γνώμονα αυτή τη λογική, εγκαταστήσαμε διάφορες υλοποιήσεις server του πρωτοκόλλου MQTT σε ένα ιδιωτικό δίκτυο τοπικής εμβέλειας (private LAN). Για την επιβεβαίωση της ορθής λειτουργίας του συστήματος εγκαταστήσαμε στο ίδιο περιβάλλον και υλοποιήσεις MQTT client, ώστε το περιβάλλον να είναι πλήρως λειτουργικό και ταυτόχρονα να παραμένει απομονωμένο από το οικιακό δίκτυο. Αυτή η επιλογή προστατεύει το οικιακό δίκτυο και κατ' επέκταση τον τελικό καταναλωτή, καθώς οι δοκιμές ασφάλειας μπορεί να οδηγήσουν σε κατάρρευση ή λειτουργία εκτός προδιαγραφών εγκατεστημένα συστήματα που εξυπηρετούν τις πραγματικές ανάγκες ενός σπιτιού.

Το περιβάλλον δοκιμών βασίζεται στο λειτουργικό σύστημα Microsoft Windows και σε εικονικές μηχανές (virtual machines) βασισμένες στο λειτουργικό σύστημα Kali Linux. Επιπλέον, χρησιμοποιείται ένας μικρο-υπολογιστής Raspberry Pi, ο οποίος εκτελεί το βασισμένο στο Linux

[52] https://github.com/OpenRCE/sulley

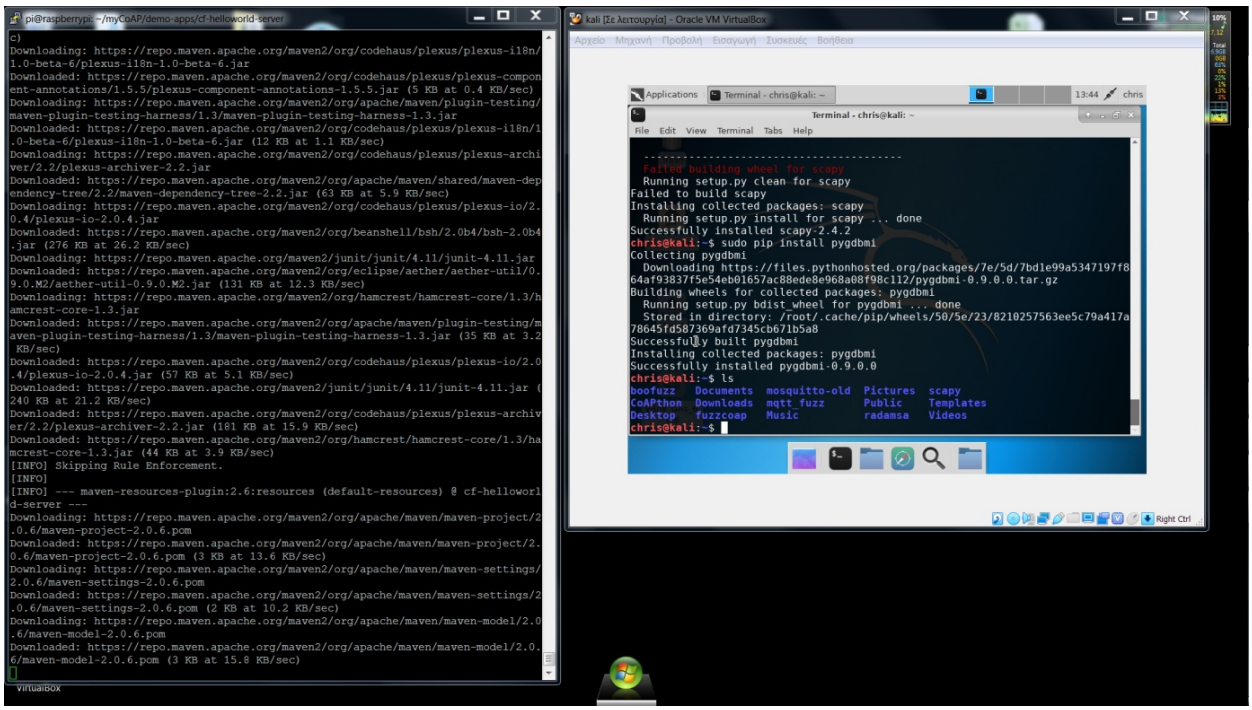
λειτουργικό σύστημα Raspbian Jessy καθώς και ένα έξυπνο κινητό τηλέφωνο (smartphone) με λειτουργικό σύστημα Google Android. Μέσα στις εικονικές μηχανές εκτελείται το λογισμικό MQTT fuzzer, το οποίο παρουσιάζουμε στην επόμενη ενότητα.



Εικόνα 6: Raspberry Pi και kali virtual machine σε Desktop

Εγκαταστήσαμε στο περιβάλλον δοκιμών τις εξής, συνολικά επτά, υλοποιήσεις MQTT broker:

- Mosquitto έκδοση 1.4.10 αλλά και 1.6, σε περιβάλλον Raspberry Pi καθώς και Microsoft Windows (host environment για τις εικονικές μηχανές)
- Cassandana έκδοση 0.1.1, σε περιβάλλον Raspberry Pi
- Moquette έκδοση 0.12.1, σε περιβάλλον Raspberry Pi
- Bevywise MQTT Route έκδοση 1, σε περιβάλλον Microsoft Windows



Εικόνα 7: Raspberry Pi shell και virtual machine με Kali Linux

Εγκαταστήσαμε επίσης στο περιβάλλον δοκιμών δύο υλοποιήσεις MQTT client:

- Paho, σε περιβάλλον Microsoft Windows (host environment για τις εικονικές μηχανές)
- MyMQTT, σε περιβάλλον Google Android (smartphone)

Οι υλοποιήσεις του MQTT client χρησιμοποιούνται επικουρικά, για τον έλεγχο καλής λειτουργίας των υλοποιήσεων MQTT broker κατά τη διάρκεια του fuzz testing. Οι MQTT client εγγράφονται (subscribe) στο θέμα (topic) MQTT «#», οπότε και λαμβάνουν οποιοδήποτε μήνυμα MQTT που διακινείται από το εκάστοτε MQTT broker.

4.3 MQTT Fuzzer και Radamsa

Αρχικά έγινε μία αναζήτηση για εργαλεία MQTT fuzzing, τα οποία να είναι δωρεάν διαθέσιμα και ανοικτού κώδικα., εφαρμόζοντας τη λογική που προαναφέρθηκε, δηλαδή της χρήσης εργαλείων στα οποία μπορεί να έχει πρόσβαση ο οποιοσδήποτε. Ο στόχος ήταν όσο το δυνατόν πιο γρήγορα και εύκολα, χωρίς ιδιαίτερες ρυθμίσεις να μπορούμε να έχουμε ένα αξιόπιστο και δοκιμασμένο περιβάλλον για ελέγχους, το οποίο να μπορεί να επεκταθεί κατά το δοκούν, αν χρειάζεται.

Το κριτήριο επιλογής ήταν ένα εργαλείο, το οποίο είναι έτοιμο να δουλέψει πάνω σε πρωτόκολλο MQTT, ώστε να αποφύγουμε την ανάπτυξη εκ του μηδενός. Για τον έλεγχο στις εκδόσεις των MQTT brokers επιλέχθηκε το MQTT fuzzer **mqtt_fuzz**. Το mqtt_fuzz[53] είναι ένα έργο ανοικτού κώδικα, το οποίο διατίθεται μέσω του GitHub. Το fuzzer δοκιμάζει το πρωτόκολλο MQTT αποστέλλοντας ένα σύνολο (ακολουθίες) από προηγουμένως καθορισμένα πακέτα MQTT κατά τη διάρκεια ενός session και ανά τακτά χρονικά διαστήματα στέλνει παραλλαγμένα πακέτα πίσω στον broker. Οι παραλλαγές στα πακέτα γίνονται μέσω ενός δεύτερου λογισμικού, του Radamsa[54], το οποίο είναι και η «καρδιά» του mqtt fuzz.

Το **Radamsa** είναι ένα γενικού σκοπού εργαλείο fuzzing, το οποίο μπορεί να ενσωματωθεί και να ελέγξει οποιαδήποτε εφαρμογή χωρίς να υπάρχει απαίτηση για γνώσεις των εισόδων ή της δομής τους. Είναι ένα καθαρά black box fuzzer, το οποίο παράγει τα πακέτα με τις τυχαίες εισόδους προς αποστολή στην υλοποίηση απλά δεχόμενος την οποιαδήποτε είσοδο..

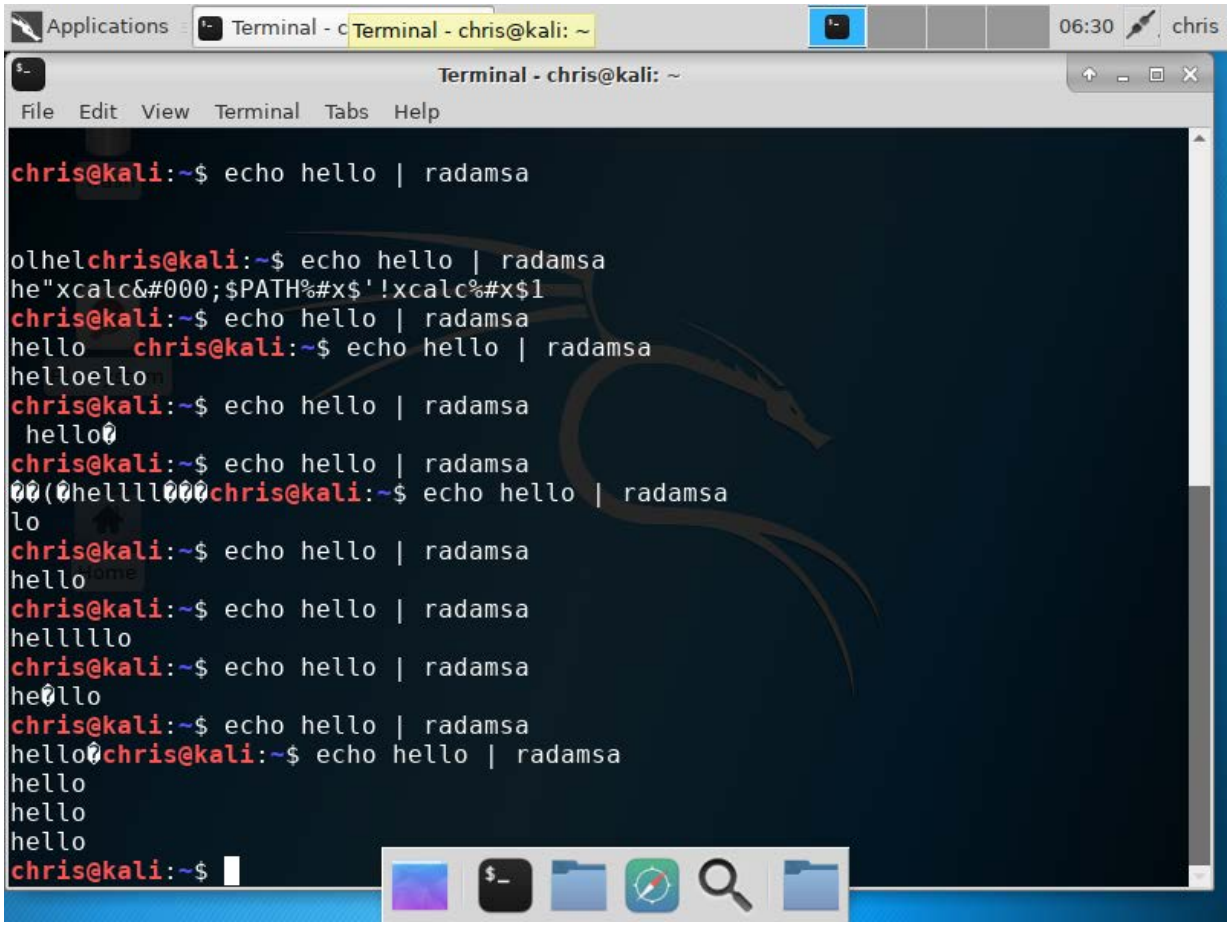
Το Radamsa έχει αποδειχτεί ένα πολύ καλό fuzzer. Έχει συνεισφέρει[55] στον εντοπισμό πάρα πολλών ευπαθειών λογισμικού [14]. Μερικά παραδείγματα είναι τα CVE-20073641 και CVE-2007-3644 (archive_read_support_format_tar.c library vulnerabilities), το CVE-2008-6536 (7-zip program vulnerability) και το CVE-2010-2482 (LibTIFF 3.9.4 vulnerability).

Το Radamsa στην περίπτωση του πρωτοκόλλου MQTT επεξεργάζεται ήδη υπάρχοντα έγκυρα πακέτα ελέγχου MQTT, στα οποία είτε προσαρτά νέα δεδομένα, είτε αλλοιώνει τα υπάρχοντα δεδομένα προς αποστολή. Δυνητικά και για όσο το mqtt fuzz βρίσκεται σε λειτουργία, το Radamsa θα παράγει νέα fuzz test πακέτα για ανεξάντλητο αριθμό περιπτώσεων ελέγχου (test cases). Η εγκατάσταση του Radamsa στο περιβάλλον δοκιμών έγινε σύμφωνα με τις οδηγίες που παρέχονται στην ιστοσελίδα του[56].

Η εκτέλεση του MQTT fuzzer γίνεται μέσα από μία εικονική μηχανή με λειτουργικό σύστημα Kali Linux. Το mqtt_fuzz δέχεται σαν είσοδο τη διεύθυνση IP και τη θύρα δικτύου (port number), στις οποίες πρέπει να συνδεθεί, ώστε να αποκτήσει επικοινωνία με το MQTT broker προς έλεγχο. Για παράδειγμα, αν το MQTT broker «τρέχει» στη διεύθυνση 192.168.168.1.42:883, τότε:

[53] https://github.com/F-Secure/mqtt_fuzz
[54] https://gitlab.com/akihe/radamsa
[55] http://primeurmagazine.com/weekly/AE-PR-09-13-6.html
[56] https://gitlab.com/akihe/radamsa

pythonmqtt_fuzz.py192.168.1.42 883



Εικόνα 8: Απλά παραδείγματα αλλοίωσης συμβολοσειρών εισόδου με το Radamsa

Κάθε συνεδρία που εκκινεί από το mqtt_fuzz λαμβάνει μία μοναδική ταυτότητα (UUID), ώστε σε περίπτωση κάποιου σφάλματος στο σύστημα να μπορούμε να το αναγνωρίσουμε. Ανατρέχοντας στο αρχείο καταγραφής εξόδου (output log) από το fuzzer μπορούμε να βρούμε την αντίστοιχη χρονοσφραγίδα (UNIX time stamp) που θα έχουμε πάρει από το σύστημα όταν έγινε και το crash του προγράμματος προς έλεγχο και να εντοπίσουμε την ακολουθία των πακέτων με το συγκεκριμένο UUID.

Η αναλογία των έγκυρων και fuzzed πακέτων διαφέρει σε κάθε νέα συνεδρία (session). Για παράδειγμα, δε γίνεται πάντα αποστολή πακέτων fuzzed Connect, καθώς αυτό μπορεί να κατέληγε ότι δε θα φτάναμε ποτέ να δούμε τι γίνεται σε πακέτα PUBLISH ή DISCONNECT. Το fuzzer λειτουργεί συνεχώς και στέλνει νέα πακέτα στο MQTT broker έως ότου το MQTT broker πάψει να ανταποκρίνεται σε αιτήματα για νέες συνεδρίες (sessions).

Το εύρος των πακέτων που εξετάζει το mqtt_fuzz από τα ήδη υπάρχοντα έγκυρα πακέτα είναι: CONNECT, CONNACK, PUBLISH, PUBACK, SUBSCRIBE, PUBCOMP, PUBREL, PUBREC και DISCONNECT.

Στο πλαίσιο της μεταπτυχιακής διατριβής, βελτιώσαμε και επεκτείναμε την υλοποίηση του mqtt_fuzz, ώστε να μπορεί να χειριστεί επιπλέον τα πακέτα: UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, SUBACK και AUTH, τα οποία συμπεριλαμβάνονται στη νέα έκδοση 5.0 του πρωτοκόλλου MQTT. Προκειμένου να επιτευχθεί αυτό, χρησιμοποιήσαμε το εργαλείο Wireshark [57] σε περιβάλλον Linux, ώστε να καταγράψουμε από υλοποιήσεις της νεότερης έκδοσης 3.1.1 του πρωτοκόλλου MQTT παραδείγματα τέτοιων πακέτων.

Τα συλληφθέντα πακέτα χρησιμοποιήθηκαν στη συνέχεια ώστε να τροφοδοτηθεί η μηχανή παραγωγής μεταλλαγμένων πακέτων του Radamsa. Για να δημιουργηθεί ένα νέο Radamsa test case από τα συλληφθέντα πακέτα, δημιουργούμε ένα νέο κατάλογο στον κατάλογο **valid-cases** της εγκατάστασης του Radamsa. Από τη σύλληψη του Wireshark παίρνουμε 1-15 πακέτα raw data και τα προσθέτουμε στον κατάλογο. Ο ακριβής χρόνος σύλληψης και οι αντίστοιχες χρονοσφραγίδες (timestamp) δεν καταγράφηκε, καθώς δεν απαιτείται στα σενάρια χρήσης για το fuzz testing που θα εφαρμόσουμε στη συνέχεια. Η επικοινωνία με το MQTT broker διατηρήθηκε τόσο όσο ήταν απαραίτητο ώστε να εντοπιστεί τουλάχιστον ένα παράδειγμα από κάθε ένα από τα νέα πακέτα του MQTT.

Στο πλαίσιο της μεταπτυχιακής διατριβής, δημιουργήσαμε δύο hard disk images, τα οποία είναι έτοιμα προς χρήση από οποιοδήποτε χρήστη. Το ένα περιέχει το βελτιωμένο mqtt_fuzzer σε λειτουργικό περιβάλλον Kali Linux, έτοιμο προς εκτέλεση δίνοντας ως παράμετρο τη διεύθυνση IP και θύρα (port) του προς έλεγχο MQTT broker. Το δεύτερο περιέχει το βελτιωμένο mqtt_fuzzer σε λειτουργικό περιβάλλον Raspbian. Ομοίως, είναι έτοιμο προς εκτέλεση δίνοντας ως παράμετρο τη διεύθυνση IP και θύρα (port) του προς έλεγχο MQTT broker.

4.4 Ανάλυση Ευρημάτων MQTT fuzzing

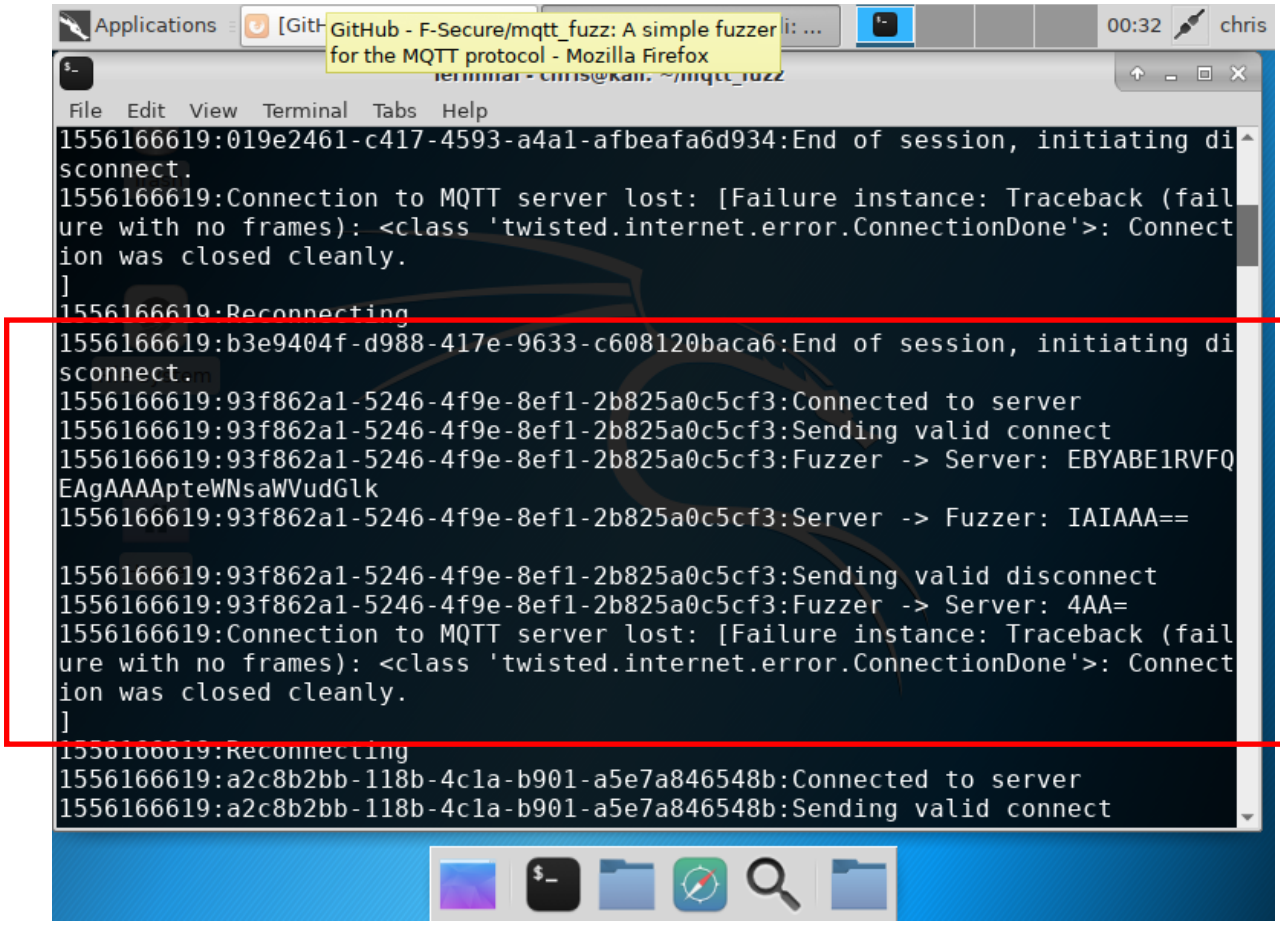
Το fuzz testing εφαρμόστηκε στις υλοποιήσεις των MQTT broker που εγκαταστήσαμε στο περιβάλλον δοκιμών (συνολικά επτά). Το κάθε σενάριο δοκιμών αφορούσε μία συγκεκριμένη υλοποίηση MQTT broker, στο οποίο συνδέαμε και τα δύο MQTT client και στη συνέχεια

[57] https://www.wireshark.org/

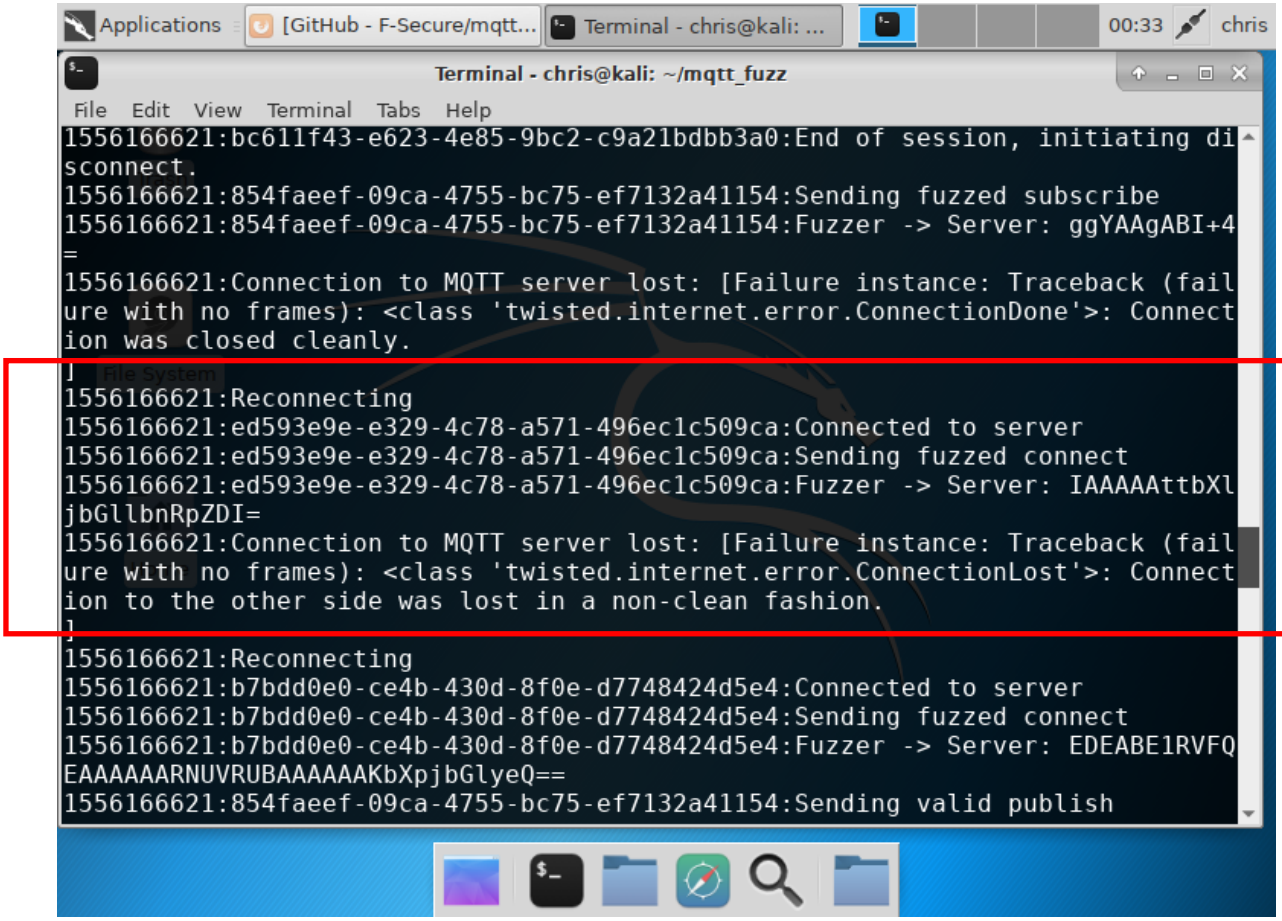
ενεργοποιούσαμε το βελτιωμένο mqtt_fuzzer. Κάθε σενάριο εκτελέστηκε για δώδεκα (12) ώρες συνεχούς fuzzing ή έως την κατάρρευση της σύνδεσης μεταξύ mqtt_fuzzer και MQTT broker.

Τρέχοντας το βελτιωμένο mqtt fuzzer παρατηρήσαμε ότι οι υλοποιήσεις MQTT broker σε αρκετές περιπτώσεις παρουσιάζουν τερματισμό της επικοινωνίας υπό μη κανονικές συνθήκες σε αντίθεση με τους αναμενόμενους τερματισμούς σε μια ακολουθία. Πιο συγκεκριμένα, πειραματιστήκαμε με τις εκδόσεις MQTT broker Mosquito (C-based, Linux και Windows), Cassandana (Java σε Linux), Moquette (C-based σε Linux) και Bevywise (Python σε Linux).

Σε ορισμένες περιπτώσεις και ιδιαίτερα σε περιβάλλον Microsoft Windows, το MQTT broker, ανάλογα με τα περιεχόμενα του πακέτου fuzzing, άλλοτε τερμάτιζε τη σύνδεση μετά την ολοκλήρωση της επικοινωνίας κανονικά (αναμενόμενη συμπεριφορά) και άλλοτε η σύνδεση διακοπτόταν βίαια και *εκτός της προδιαγραφής* του πρωτοκόλλου MQTT. Δύο χαρακτηριστικά παραδείγματα απεικονίζονται στις παρακάτω συλλήψεις οθόνης (screenshot) κατά την εκτέλεση του MQTT broker **mosquitto**.



Εικόνα 9: Αναμενόμενοι τερματισμοί συνεδριών client server από fuzzed πακέτα

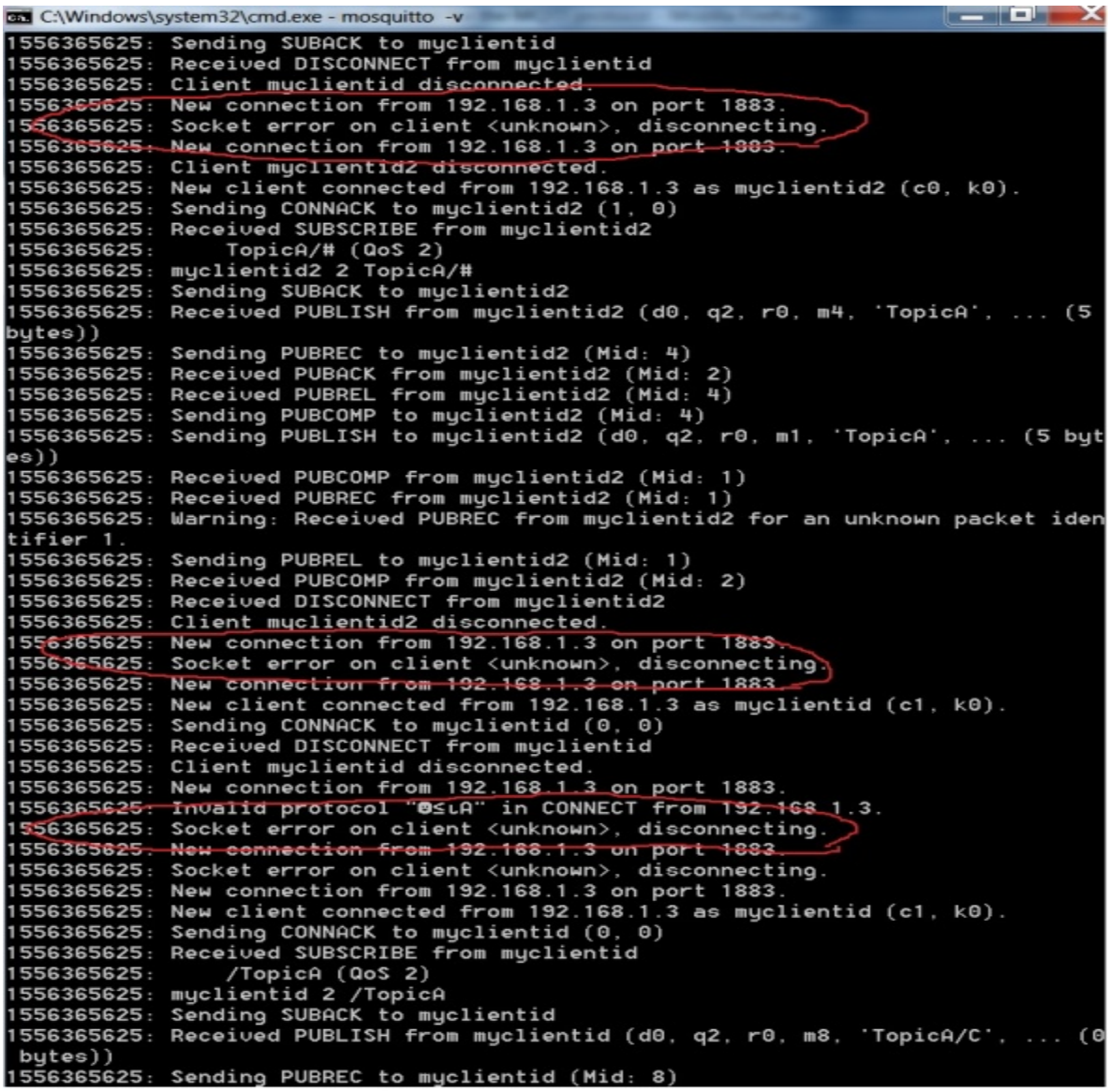


Εικόνα 10: Μη αναμενόμενοι τερματισμοί συνεδριών κατά τη διαδικασία fuzzing

Αυτοί οι μη αναμενόμενοι τερματισμοί εμφανίζονται μετά από αποστολή fuzzed πακέτων και των τριών κατηγοριών πακέτων που το συγκεκριμένο πρόγραμμα αλλοιώνει. Είχαμε μη αναμενόμενους τερματισμούς και μετά από fuzzed connection, publish , subscribe πακέτα.

Επίσης στην παραπάνω σύλληψη οθόνης παρουσιάζεται η απόκριση του server και τους μη αναμενόμενους τερματισμούς των session από τον broker. Τα αποτελέσματα ήταν παρόμοια και για τις δύο εκδόσεις 1.4 και 1.6 του mosquito.

Παρόμοια αποτελέσματα είχαμε και με την υλοποίηση του **Moquette**. Εγκαταστήσαμε την τελευταία έκδοση σε Raspberry Pi και στο τερματικό παράθυρο του προγράμματος παρατηρήθηκαν σφάλματα και τερματισμοί session λόγω αναπάντεχων σφαλμάτων από εισόδους. Ενδεικτικά:

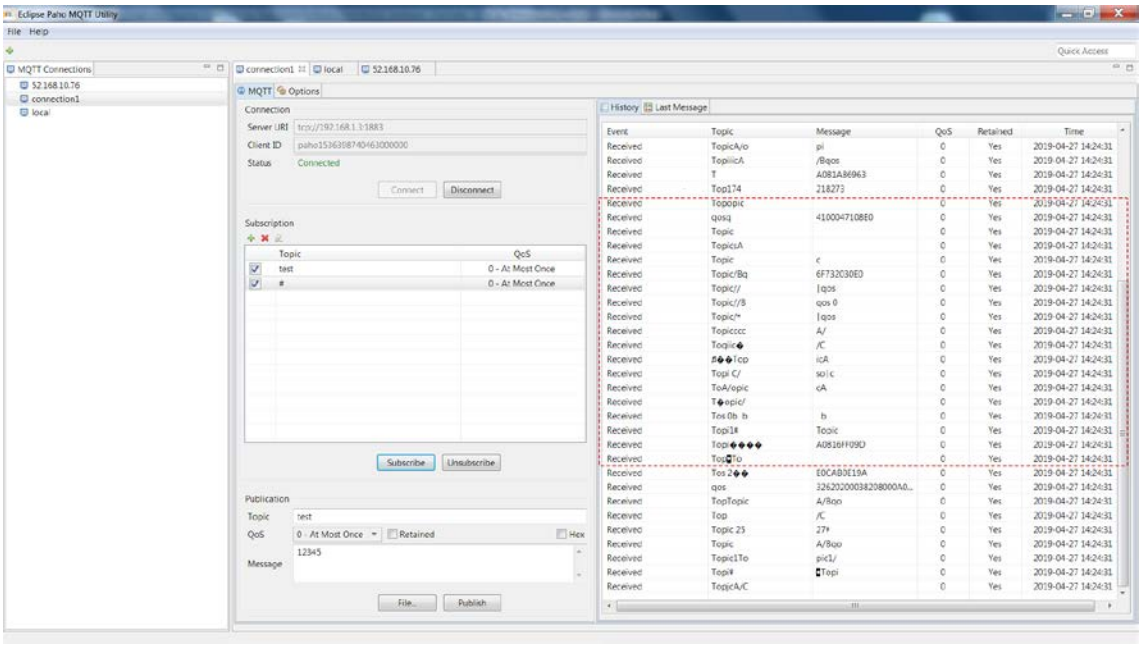


Εικόνα 11: Εμφάνιση disconnection στο τερματικό του mosquito server σε περιβάλλον Windows

Το **paho** client, σε αντίθεση, παρέμενε συνδεδεμένο σε όλη τη διάρκεια των δοκιμών. Το **MyMQTT** client, μετά από λίγη ώρα (2-3 λεπτά), κατά την οποία και παραλάμβανε fuzzed μηνύματα από τον broker και έστελνε ο fuzzer, έχανε απρόσμενα τη σύνδεση από το broker.

Επίσης παρατηρήθηκε ότι στην έκδοση του **mosquito**1.4.10 ο broker **σταματούσε να δέχεται νέες συνδέσεις** από το fuzzer μετά από περίπου 20 λεπτά και το πρόγραμμα fuzzer αυτόματα σταματούσε τη λειτουργία του. Αυτή η συμπεριφορά παρατηρήθηκε στην έκδοση του mosquito που έτρεχε και στη συσκευή Raspberry Pi 3 με λειτουργικό σύστημα Raspbian αλλά και στην έκδοση για Microsoft Windows. Στη νεότερη έκδοση του mosquito αυτή η συμπεριφορά δεν

παρατηρήθηκε ακόμα και μετά από 12 ώρες fuzzing. Συνεπώς, η παλαιότερη έκδοση είναι ευάλωτη σε μία επίθεση άρνησης παροχής υπηρεσίας (denial of service attack).



Εικόνα 12: Μηνύματα του συστήματος που στέλνονταν στον client Paho

Κατά τον έλεγχο του **moquette** παρατηρήθηκαν κατά τη διαδικασία του fuzzing μη αναμενόμενες συμπεριφορές και αναφορές για τερματισμό sessions με μη αναμενόμενο τρόπο. Ένα χαρακτηριστικό παράδειγμα από το αρχείο καταγραφής (log file) του moquette είναι το εξής:

27/04/2019 18:39:37,094 [nioEventLoopGroup-3-7] ERROR NewNettyMQTTHandlerexceptionCaught 87- Unexpected exception while processing MQTT message. Closing Netty channel. CId=myclientid

java.nio.channels.ClosedChannelException
at io.netty.channel.AbstractChannel$AbstractUnsafe.close(...)(Unknown Source)
27/04/2019 18:39:37,098 [nioEventLoopGroup-3-7] ERROR NewNettyMQTTHandlerexceptionCaught 87- ***Unexpected exception while processing MQTT message. Closing Netty channel. CId=myclientid***
java.nio.channels.ClosedChannelException
at io.netty.channel.AbstractChannel$AbstractUnsafe.close(...)(Unknown Source)
27/04/2019 18:39:37,559 [nioEventLoopGroup-3-8] ERROR NewNettyMQTTHandlerexceptionCaught 87- ***Unexpected exception while processing MQTT message. Closing Netty channel. CId=myclientid2***
java.nio.channels.ClosedChannelException
at io.netty.channel.AbstractChannel$AbstractUnsafe.close(...)(Unknown Source)

27/04/2019 18:39:37,566 [nioEventLoopGroup-3-8] INFOMQTTConnectionhandleConnectionLost 252- Notifying connection lost event. CId: myclientid2, channel: [id: 0x850016e1, L:0.0.0.0/0.0.0.0:1883 ! R:/192.168.1.100:35762]

27/04/2019 18:39:36,927 [nioEventLoopGroup-3-6] ERROR NewNettyMQTTHandlerexceptionCaught 87- Unexpected exception while processing MQTT message. Closing Netty channel. CId=null

java.io.IOException: invalid massage

Οι υλοποιήσεις MQTT Broker της Bevywise και Cassandana παρουσίασαν συμπεριφορά και αποτελέσματα παρόμοια με τα παραπάνω. Σε πολλές περιπτώσεις το fuzzing προκαλούσε μη αναμενόμενους τερματισμούς σε ανοικτές συνεδρίες. Ωστόσο, είναι ιδιαίτερα ενθαρρυντικό ότι ακόμη και μετά από 12 ώρες συνεχούς fuzzing, δεν παρατηρήθηκε κάποια ολική κατάρρευση του broker.

Τα παραπάνω ευρήματα δείχνουν ότι είχαμε **τερματισμούς σε συνεδρίες λόγω των fuzzed πακέτων** καθώς και **ολική άρνηση παροχής υπηρεσίας**. Τα ευρήματα αυτά είναι ιδιαίτερα σημαντικά, ειδικά σε ό,τι αφορά στα MQTT broker, τα οποία θα έπρεπε να είναι σε θέση να διαχειρίζονται με ασφάλεια οποιοδήποτε πακέτο και αν λάβουν. Επί του παρόντος, και δεδομένου το πόσο εύκολα προσπελάσιμα είναι τα MQTT broker από το ανοικτό Διαδίκτυο, μπορεί κάποιος κακόβουλος να κατασκευάσει και να αποστείλει τέτοια πακέτα (crafted packets), τα οποία μπορούν να διακόψουν πλήρως τη λειτουργία του MQTT broker.

Περαιτέρω ανάλυση είναι ιδιαίτερα χρήσιμη, ώστε να διαπιστωθεί αν πέρα από τη διακοπή της λειτουργίας που εντοπίσαμε, κάποια πακέτα μπορούν να χρησιμοποιηθούν με τέτοιο τρόπο ώστε να δημιουργήσουν τις συνθήκες για την εκτέλεση ενός remote exploit από κακόβουλους χρήστες, ώστε να αποκτήσουν τον πλήρη έλεγχο του συστήματος που εκτελεί το MQTT broker.

Κεφάλαιο 5
Συμπεράσματα και Μελλοντικές Κατευθύνσεις

5.1 Συμπεράσματα

Η χρήση των συστημάτων IoT βαίνει αυξανόμενη. Οι υλοποιήσεις των πρωτοκόλλων επικοινωνίας θα πρέπει να εξελιχθούν στις νέες ανάγκες τις αγοράς και στα μέτρα των συσκευών που θα παράγουν οι κατασκευαστές. Είναι αναγκαίο όμως να υπάρχει καλύτερη και μεθοδικότερη πρόληψη και έλεγχος των συστημάτων προτού γίνουν διαθέσιμα στους τελικούς χρήστες. Και μετά την κυκλοφορία θα πρέπει να υπάρχει ενημέρωση των λογισμικών ώστε να είναι προστατευμένα τα συστήματα από νέους κινδύνους και κενά ασφαλείας.

Διαπιστώσαμε ότι με εργαλεία ελεύθερου λογισμικού και λογισμικού ανοικτού κώδικα (ΕΛ/ΛΑΚ), μπορεί ο καθένας να ελέγξει μία υλοποίηση πρωτοκόλλων IoT σε ένα ελεγχόμενο περιβάλλον δοκιμών. Στους ελέγχους ασφάλειας που κάναμε για το πρωτόκολλο MQTT διαπιστώσαμε αποκλίνουσα των προδιαγραφών λειτουργία του MQTT broker αλλά και άρνηση υπηρεσίας (απόρριψη νέων συνδέσεων).

Προκύπτει από τα πειραματικά μας ευρήματα ότι υπάρχουν απλοί είσοδοι (πακέτα πρωτοκόλλων δικτύων IoT), οι οποίες προκαλούν μη αναμενόμενη και εκτός προδιαγραφών συμπεριφορά των συστημάτων IoT. Επιπλέον, συλλέχθηκαν δεδομένα και ενδείξεις, τα οποία χρήζουν περαιτέρω ελέγχου και ανάλυσης, ώστε να διαπιστωθεί αν και σε ποιό βαθμό μπορούν να αποτελέσουν σημεία εκμετάλλευσης από κακόβουλους χρήστες εφαρμογών για την απόκτηση απομακρυσμένης πρόσβασης χωρίς εξουσιοδότηση (remote exploit) στο σύστημα που εκτελεί το MQTT broker.

Οι προγραμματιστές υλοποιήσεων πρωτοκόλλων και οι κατασκευαστές και προμηθευτές συστημάτων IoT θα πρέπει να ενσωματώσουν και αυτοί στους ελέγχους και στις πρακτικές

software testing τουλάχιστον τις τεχνικές fuzzing, οι οποίες προσφέρουν ένα γρήγορο και αποδοτικό έλεγχο, ανεξάρτητα από την δομή του προγράμματος και των δυνατοτήτων των συσκευών. Κρίνουμε επίσης αναγκαία την ενσωμάτωση αυτόματης διαδικασίας αναβάθμισης των λειτουργικών συστημάτων και σταθερισμικών (firmware) σε συστήματα IoT και ιδιαίτερα σε συστήματα που απευθύνονται σε οικιακούς καταναλωτές. Η διαδικασία αυτή θα παρέχει τη δυνατότητα κάλυψης και διόρθωσης σημείων, τα οποία δίνουν τη δυνατότητα σε επιτιθέμενους να εκτελέσουν επιτυχώς κακόβουλο κώδικα ή να αποκτήσουν μη εξουσιοδοτημένη πρόσβαση σε μεταδιδόμενη πληροφορία σε συστήματα IoT.

5.2 Μελλοντικές κατευθύνσεις

Αποτελεί ενδιαφέρον θέμα για περεταίρω έρευνα η ενσωμάτωση προηγμένων τεχνικών fuzzing για την ολοκληρωμένη ανάλυση και επέκταση του ελέγχου σε όλο το εύρος των πρωτοκόλλων και των υλοποιήσεών τους σε συστήματα IoT. Μία άλλη κατεύθυνση έρευνας αφορά στην αυτοματοποίηση της διαδικασίας fuzzing, ώστε να γίνεται συνεχώς και αδιαλείπτως, σε υποδομές νεφο-υπολογιστικής (cloud computing). Δεδομένης της διαρκούς μείωσης του κόστους των σχετικών υποδομών, κάτι τέτοιο είναι πλέον εφικτό, τόσο κατά τη φάση ανάπτυξης και την ενσωμάτωση σε διαδικασίες συνεχούς ολοκλήρωσης και παράδοσης (continuous integration and continuous delivery, CI/CD).

Μία άλλη κατεύθυνση έρευνας είναι η ενσωμάτωση προηγμένων τεχνικών ελέγχου ασφάλειας, πέραν του fuzzing, στη διαδικασία ανάπτυξης του λογισμικού. Ιδιαίτερο ενδιαφέρον παρουσιάζει η ολοκλήρωση σε αυτές των ιδιαίτερων χαρακτηριστικών των συστημάτων IoT και ειδικότερα της μοντελοποίησης του φυσικού περιβάλλοντος, ώστε να διευρυνθεί το πεδίο δοκιμών και ελέγχου πέραν των στατικών υλοποιήσεων. Αποτελεί ενδιαφέρουσα κατεύθυνση επίσης η συγκριτική μελέτη των διαφόρων τεχνικών ελέγχου ασφάλειας, ώστε να επιλέγεται σε κάθε σενάριο χρήσης η πλέον αποδοτική και κατάλληλη ή ο πλέον αποδοτικός συνδυασμός διαφορετικών τεχνικών, ώστε να καλύπτεται όσο το δυνατό μεγαλύτερος χώρος αναζήτησης (search space) με όσο το δυνατό μικρότερο πλήθος δοκιμών (test cases).

Είναι σαφές ότι πλην εξαιρετικών περιπτώσεων, οι τεχνικές ελέγχου ασφάλειας δεν παρέχουν σαφή αποτελέσματα για το βαθμό ευπάθειας ενός συστήματος ή μίας υλοποίησης. Παρέχουν όμως σαφείς ενδείξεις για σημεία ενδιαφέροντος, τα οποία χρήζουν περαιτέρω ανάλυσης, ώστε να αποδειχθεί αν υπάρχει όντως ένα εκμεταλλεύσιμο κενό ασφάλειας ή πρόκειται για μία αστοχία

υλοποίησης, η οποία μπορεί στην καλύτερη (από πλευράς επιτιθέμενου) περίπτωση να οδηγήσει σε μία επίθεση στη διαθεσιμότητα του συστήματος. Αποτελεί ένα ενδιαφέρον πεδίο περαιτέρω έρευνας η δυνατότητα αυτόματης παραγωγής και ελέγχου σεναρίων επίθεσης βασισμένα στα ευρήματα των τεχνικών ελέγχου ασφάλειας.

Βιβλιογραφία

[01] K. Ashton, "That 'Internet of Things' Thing," *RFiD J.*, p. 4986, 2009.

[02] K. Ullah, M. A. Shah, and S. Zhang, "Effective ways to use Internet of Things in the field of medical and smart health care," in *2016 International Conference on Intelligent Systems Engineering, ICISE 2016*, 2016, pp. 372–379.

[03] Postscapes, "IoT Standards and Protocols | 2016 Comparison Guide on Network, Wireless Comms, Security, Industrial.," *Postscapes*, 2017. [Online]. Available: http://www.postscapes.com/internet-of-things-protocols/.

[04] C. Hennebert, , and J. Dos. Santos, (2014). Security protocols and privacy issues into 6LoWPAN stack: A synthesis. *IEEE Internet of Things Journal*, *1*(5), 384–398. https://doi.org/10.1109/JIOT.2014.2359538

[05] X. Perry, HiveMQ MQTT Broker. In *Designing Embedded Systems and the Internet of Things (IoT) with the Arm® Mbed™*. (2018) https://doi.org/10.1002/9781119364009.app2

[06] R. A Light, "Mosquitto: server and client implementation of the MQTT protocol," *J. Open Source Softw.*, vol. 2, no. 13, 2017.

[07] V. Gupta, M Wurm, Y.Zhu, M. Millard, S Fung, N. Gura, H. Eberle, and S.C. Shantz, "Sizzle: A standards – Based End to End Security Architecture for the Embeded Internet" pervasive moblie computing, vol. 1, PP 425-445 , Dec. 2005

[08] J. Vishwesh and M. B. Rajashekar, "Internet of Things (IoT): Security Analysis & Security Protocol CoAP," *Int. J. Recent Trends Eng. Res.*, vol. 3, no. 3, pp. 417–425, 2017.

[09] D. Trabalza, "Implementation and Evaluation of Datagram Transport Layer Security (DTLS) for the Android Operating System," , June, 2012

[10] Q. Jing, A.V. Vasilakos, J. Wan, J. Lu, D. Qui, "Security of the Internet of Things: perspectives and challenges", Springer, Wireless Networks, vol. 20, Iss.8, pp. 2481–2501.

[11] S. Kraijak and P. Tuwanut, "A survey on internet of things architecture, protocols, possible applications, security, privacy, real-world implementation and future trends", Proceedings of ICCT, 2015, pg.26-31.

[12] G.A. Fink, D.V. Zarhitsky, T.E. Carroll, and E.D. Farquhar, "Security and Privacy Grand Challenges for the Internet of Things", IEEE, 2015, pp.27-34.

[13] J. Du and S.W. Chao, "A Study of Information Security for M2M of lOT", IEEE International Conference on Advanced Computer Theory and Engineering (ICACTE), 2010, pp.576-579

[14] P. Fernandes, A. Monteiro, and S. A. Lasrado, "(IOT): Security Challenges and Future Scope," International Journal of Latest Trends in Engineering and Technology Special Issue SACAIM,2016, pp. 164–168

[15] A. Oracevic, S. Dilek, and S. Ozdemir, "Security in internet of things: A survey," *2017 Int. Symp. Networks, Comput. Commun.*, no. May, pp. 1–6, 2017.

[16] P. S. Rajebhosale, M. A. Sonawane, M. S. Pawar, M. V. Kadam, and M. A. Waghela, "'IOT based Home Automation and Security System,'" *Ijarcce*, vol. 6, no. 3, pp. 821–824, 2017.

[17] M. Wang, "Understanding security flaws of IoT protocols through honeypot technologies," 2017, University of Technology of Delft, Netherlands

[18] C.-Y. Chen, M. Hasan, and S. Mohan, "Securing Real-Time Internet-of-Things," pp. 1–10, 2017.

[19] D. M. Mendez, I. Papapanagiotou, and B. Yang, "Internet of Things: Survey on Security and Privacy," pp. 1–16, 2017.

[20] H. Lin and N. Bergmann, "IoT Privacy and Security Challenges for Smart Home Environments," *Information*, vol. 7, no. 3, p. 44, 2016.

[21] S.-H. Ramos, M.T. Villalba, and R. Lacuesta, "MQTT Security: A Novel Fuzzing Approach," Wireless Communications and Mobile Computing, vol. 2018, Article ID 8261746, 11 pages, 2018. https://doi.org/10.1155/2018/8261746.

[22] R. A. Mughal, "Internet of Things (IoT) Protocols : A Brief Exploration of MQTT and CoAP Internet of Things," International Journal of Computer Applications (0975 – 8887) Volume 179 – No.27, March 2018

[23] D. M. Rathod, "Security Analysis of Constrained Application Protocol (CoAP): IoT Protocol," no. August, 2017.

[24] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J.-A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai Botnet. In 26th USENIX Security Symposium, pages 1093–1110, Vancouver, BC, 2017.

[25] R. Hummen, J. Hiller, H. Wirtz, M. Henze, H. Shafagh, and K. Wehrle, "6LoWPAN Fragmentation Attacks and Mitigation Mechanisms", ACM WiSec'2013, Budapest, Hungary.

[26] S. Torabi, E. Bou-Harb, C. Assi, M. Galluscio, A. Boukhtouta, and M. Debbabi. Inferring, characterizing, and investigating Internet-Scale malicious IoT device activities: A network telescope perspective. *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018*, (April), 562–573. https://doi.org/10.1109/DSN.2018.00064

[27] A. Triantafyllou, P. Sarigiannidis, and T.D. Lagkas, "Network Protocols, Schemes, and Mechanisms for Internet of Things (IoT): Features, Open Challenges, and Trends," Wireless Communications and Mobile Computing, vol. 2018, Article ID 5349894, 24 pages, 2018. https://doi.org/10.1155/2018/5349894.

[28] B. Rossi, "*Gartner's Internet of Things Perdictions*", Information Age, Vitesse Media, January 2015

[29] S. Hernández Ramos, M. T. Villalba, and R. Lacuesta, "MQTT Security: A Novel Fuzzing Approach". *Wireless Communications and Mobile Computing*, *2018*. http://doi.org/10.1155/2018/8261746

[30] M. Panwar and A. Kumar, Security for IoT: An effective DTLS with public certificates. *Conference Proceeding - 2015 International Conference on Advances in Computer*

Engineering and Applications, ICACEA 2015, 163–166. https://doi.org/10.1109/ICACEA.2015.7164688

[31] M. Patton, E. Gross, R. Chinn, S. Forbis, L. Walker, and C. Hsinchun, Uninvited connections: A study of vulnerable devices on the Internet of Things (IoT). In Proceedings of the 2014 IEEE Joint Intelligence and Security Informatics Conference (JISIC), The Hague, The Netherlands, 24–26 September 2014; pp. 232–235.

[32] M. Burhan, R. A. Rehman, B. Khan, and B.S. Kim, "IoT elements, layered architectures and security issues: A comprehensive survey". *Sensors* (Switzerland), 18(9). https://doi.org/10.3390/s18092796

[33] M. Bures, T. Cerny, and B.S. Ahmed (2019). Internet of things: Current challenges in the quality assurance and testing methods. *LectureNotes in ElectricalEngineering*, *514*, 625–634. https://doi.org/10.1007/978-981-13-1056-0_61

[34] S. Naik, and V. Maral, Cyber security - IoT. RTEICT 2017 - 2nd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology, Proceedings, 2018–January, 764–767. https://doi.org/10.1109/RTEICT.2017.8256700

[35] D. Votipka, R. Stevens, E. Redmiles, J. Hu, and M. Mazurek. Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes. *Proceedings - IEEE Symposium on Security and Privacy*, *2018–May*, 374–391. https://doi.org/10.1109/SP.2018.00003

[36] R. Lekh and Pooja, "Exhaustive study of SDLC phases and their best practices to create CDP model for process improvement," *2015 International Conference on Advances in Computer Engineering and Applications*, Ghaziabad, 2015, pp. 997-1003. doi: 10.1109/ICACEA.2015.7164852

[37] R. Hummen, J. Hiller, H. Wirtz, M. Henze, and H. Shafagh. 6LoWPAN fragmentation attacks and mitigation mechanisms, https://doi.org/10.1145/2462096.2462107

[38] G. Murad, A. Badarneh, A. Quscf, and F. Almasalha. Software Testing Techniques in IoT. *2018 8th International Conference on Computer Science and Information Technology, CSIT 2018*, 17–21. https://doi.org/10.1109/CSIT.2018.8486149

[39] A. Rghioui, M. Bouhorma, and A. Benslimane Analytical study of security aspects in 6LoWPAN networks. *2013 5th International Conference on Information and Communication Technology for the Muslim World, ICT4M 2013*, (March). https://doi.org/10.1109/ICT4M.2013.6518912

[40] M. Li, L. He, Y.X. Teng, X. Wang, J. Zhang, and S. Qing, Research on network protocol vulnerability discovery based on fuzz testing. *Proceedings of the 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2017*, *2018–Janua*, 1354–1358. https://doi.org/10.1109/ITNEC.2017.8285016

[41] I. Andrianto, M.M.I. Liem, and Y.D.W. Asnar. Web application fuzz testing. *Proceedings of 2017 International Conference on Data and Software Engineering, ICoDSE 2017*, *2018–Janua*, 1–6. https://doi.org/10.1109/ICODSE.2017.8285893

[42] S. Andy, B. Rahardjo, and B. Hanindhito. Attack scenarios and security analysis of MQTT communication protocol in IoT system. *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, *4*(September), 600–604. https://doi.org/10.11591/eecsi.4.1064

[43] M. Calabretta, R. Pecori, M. Vecchio, and L. Veltri, MQTT-Auth: a Token-based Solution to Endow MQTT with Authentication and Authorization Capabilities. Journal of Communications Software and Systems. 14. 10.24138/jcomss.v14i4.604.

[44] L. Markowsky and G. Markowsky, "Scanning for Vulnerable Devices in the Internet of Things," The 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing System: Technology and Applications, September 2015.

[45] A. Prada. "Communication with resource-constrained devices through MQTT for control education," 11th IFAC Symposium on Advances in Control Education ACE, pp 150-155, Bratislava, Slovakia, 2016.

[46] M. Iqbal and M. Bayoumi, "Secure End-to-End key establishment protocol for resource-constrained healthcare sensors in the context of IoT," International Conference on High Performance Computing & Simulation (HPCS), pp. 523-530, 2016.

[47] J. M. Blythe and S. D. Johnson, "The consumer security index for IoT: A protocol for developing an index to improve consumer decision making and to incentivize greater security provision in IoT devices," *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, London, 2018, pp. 1-7. doi: 10.1049/cp.2018.0004

[48] S. S. Sezer, T1C: IoT Security: - Threats, Security Challenges and IoT *Security Research and Technology Trends*, 1–2. https://doi.org/10.1109/socc.2018.8618571

[49] C.T. Phong and W.Q. Yan, An Overview of Penetration Testing. *International Journal of Digital Crime and Forensics*. https://doi.org/10.4018/ijdcf.2014100104

[50] R. Baloch, Ethical Hacking and Penetration Testing Guide. Ethical Hacking and Penetration Testing Guide. New York, Auerbach Publications ,https://doi.org/10.4324/9781315145891

[51] J. Chen, W. Diao, Q. Zhao, C. Zuo, Z. Lin, X.F. Wang, W.C. Lau, M. Sun, Ro. Yang, and K. Zhang, IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing., https://doi.org/10.14722/ndss.2018.23159

[52] J.-Z. Luo, C. Shan, J. Cai, and Y. Liu, IoT Application-Layer Protocol Vulnerability Detection using Reverse Engineering. Symmetry. 10. 561. 10.3390/sym10110561.

[53] U.D. Gandhi, P.M. Kumar, R. Varatharajan, G. Manogaran, R. Sundarasekar, and S. Kadu, HIoTPOT: Surveillance on IoT Devices against Recent Threats. *Wireless Personal Communications*. https://doi.org/10.1007/s11277-018-5307-3

[54] M. Niedermaier, F. Fischer, and A. Von Bodisco, PropFuzz - An IT-security fuzzing framework for proprietary ICS protocols. *International Conference on Applied Electronics*. https://doi.org/10.23919/AE.2017.8053600

[55] M. Anirudh, S.A. Thileeban, and D.J. Nallathambi, Use of honeypots for mitigating DoS attacks targeted on IoT networks. *International Conference on Computer, Communication, and Signal Processing: Special Focus on IoT, ICCCSP 2017*, 1–4. https://doi.org/10.1109/ICCCSP.2017.7944057

[56] S.N.M. García, J.L. Hernández-Ramos, and A.F. Skarmeta, Test-based risk assessment and security certification proposal for the Internet of Things. *IEEE World Forum on Internet of Things, WF-IoT 2018 - Proceedings*, *2018–Janua*, 641–646. https://doi.org/10.1109/WF-IoT.2018.8355193

[57] Y. Minn, P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, IoTPOT: A Novel Honeypot for Revealing Current IoT Threats. *Journal of Information Processing*. https://doi.org/10.2197/ipsjjip.24.522

[58] V. Visoottiviseth, P. Akarasiriwong, S. Chaiyasart, and S. Chotivatunyu, PENTOS: Penetration testing tool for Internet of Thing devices. In *IEEE Region 10 Annual International Conference, Proceedings/TENCON*. https://doi.org/10.1109/TENCON.2017.8228241

[59] M. Schukat and P. Cortijo. Public key infrastructures and digital certificates for the Internet of things. In *2015 26th Irish Signals and Systems Conference, ISSC 2015*. https://doi.org/10.1109/ISSC.2015.7163785

[60] Y.S. Ko, I.K. Ra, and C.S. Kim. A study on IP exposure notification system for IoT devices using IP search engine Shodan. *International Journal of Multimedia and Ubiquitous Engineering*. https://doi.org/10.14257/ijmue.2015.10.12.07

[61] R. Williams, E. McMahon, S. Samtani, M. Patton, and H. Chen. Identifying vulnerabilities of consumer Internet of Things (IoT) devices: A scalable approach. In *2017 IEEE International Conference on Intelligence and Security Informatics: Security and Big Data, ISI 2017*. https://doi.org/10.1109/ISI.2017.8004904

[62] A. Spognardi, M. Donno, N.D. Dragoni, and A. Giaretta, Analysis of DDoS-Capable IoT Malwares. *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems*, *11*(June 2018), 807–816. https://doi.org/10.15439/2017f288

[63] H. Fryer and E, Simperl, Web Science Challenges in Researching Bug Bounties. In Proceedings of the 2017 ACM on Web Science Conference (WebSci '17). ACM, New York, NY, USA, 273-277. DOI: https://doi.org/10.1145/3091478.3091517

[64] M.S. Harsha, B.M. Bhavani, and K.R. Kundhavai. Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs.

2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018, 2244–2250. https://doi.org/10.1109/ICACCI.2018.8554472

[65] S.N. Swamy, D. Jadhav, and N. Kulkarni. Security threats in the application layer in IOT applications. *Proceedings of the International Conference on IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2017*, 477–480. https://doi.org/10.1109/I-SMAC.2017.8058395

[66] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, A Survey on Application Layer Protocols for the Internet of Things. *Transaction on IoT and Cloud Computing*, 1–10. https://doi.org/10.5281/ZENODO.51613

[67] C. Gündoğan, P. Kietzmann, T. Schmidt, M. Lenders, H. Petersen, and M. Wählisch. *NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT*.

[68] M. Ahmad, Reliability models for the internet of things: A paradigm shift. *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, 52–59. https://doi.org/10.1109/ISSREW.2014.107

[69] I. Hooda and C.R. Singh, Software Test Process, Testing Types and Techniques. *International Journal of Computer Applications*, *111*(13), 10–14. https://doi.org/10.5120/19597-1433

[70] I. Hooda and C.R. Singh, Software Test Process, Testing Types and Techniques. *International Journal of Computer Applications*, *111*(13), 10–14. https://doi.org/10.5120/19597-1433

[71] S. Bekrar, C.R. Groz, and L. Mounier. Finding software vulnerabilities by smart fuzzing. *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011*, 427–430. https://doi.org/10.1109/ICST.2011.48

[72] A. Ahmed. *Introduction to Software Testing Management. Software Testing as a Service*. Cambridge University Press. https://doi.org/10.1201/9781420099577.ch1

[73] O. Cetin, C. Gañán, L. Altena, T. Kasama, D. Inoue, K. Tamiya., anc M. van Eeten, Cleaning Up the Internet of Evil Things: Real-World Evidence on ISP and Consumer Efforts to Remove Mirai. *NDSS 2019*. https://doi.org/10.14722/ndss.2019.23438

[74] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet", Network and Distributed Systems Security (NDSS) Symposium 2019, San Diego, CA, USA, February 2019 https://doi.org/10.14722/ndss.2019.23488

[75] S. Bhattacharjee, M. Salimitari, M. Chatterjee, K. Kwiat, and C. Kamhoua, Preserving Data Integrity in IoT Networks Under Opportunistic Data Manipulation. *Proceedings - 2017 IEEE 15th International Conference on Dependable, Autonomic and Secure Computing, 2017 IEEE 15th International Conference on Pervasive Intelligence and Computing, 2017 IEEE 3rd International Conference on Big Data Intelligence and Computing and 2017 IEEE Cyber Science and Technology Congress, DASC-PICom-DataCom-CyberSciTec 2017*, *2018–January*, 446–453. https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2017.87

[76] W. Duran; Simeon C. Ntafos. A report on random testing. *ICSE '81. Proceedings of the ACM SIGSOFT International Conference on Software Engineering (ICSE'81). pp. 179–183.*

[77] M. Roberto, B. Abyi, and R. Domenico. Towards a definition of the Internet of Things (IoT). *IEEE Internet of Things*, 1–86. https://doi.org/10.5120/19787-1571

[78] P.V. Dudhe, N.V. Kadam, R.M. Hushangabade, M.S. Deshmukh, Internet of Things (IOT): An overview and its applications. *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, 2650–2653. https://doi.org/10.1109/ICECDS.2017.8389935

[79] G. Bedi, G.K. Venayagamoorthy, R. Singh, R.R. Brooks, K.C. Wang. Review of Internet of Things (IoT) in Electric Power and Energy Systems. *IEEE Internet of Things Journal*, *5*(2), 847–870. https://doi.org/10.1109/JIOT.2018.2802704

[80] K. Wrona, Securing the Internet of Things a military perspective. *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, 502–507. https://doi.org/10.1109/WF-IoT.2015.738910

[81] M. Kolisnyk and V. Kharchenko. A Markov Model of IoT System Availability Considering DDoS Attacks, Patching and Energy Modes, pp .185–207.

[82] Joe W. Duran; Simeon C. Ntafos. "An Evaluation of Random Testing". *IEEE Transactions on Software Engineering (TSE).*

[83] R. Singh, J. Singh, and R. Singh, "Hello Flood Attack Countermeasures in Wireless Sensor Networks," *Int. J. Comput. Sci. Mob. Appl.*, vol. 4, no. 5, pp. 1–9, 2016.

[84] N. Neshenko, E. Bou-harb, J. Crichigno, G. Kaddoum, and N. Ghani, "Demystifying IoT Security : An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-scale IoT Exploitations," no. July 2018, pp. 1–30, 2019.

[85] T. J. Oconnor, W. Enck, and B. Reaves, "Blinded and Confused : Uncovering Systemic Flaws in Device Telemetry for Smart-Home Internet of Things."

[86] F. Abate, M. Carratù, C. Liguori, M. Ferro and V. Paciello, "Smart meter for the IoT," *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, Houston, TX, 2018, pp. 1-6. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8409838&isnumber=8409512

[87] D.R. Thomas and A.R. Beresford. 1000 days of UDP amplification DDoS attacks.

[88] R.K..C Pooja, Review on Security Attacks and Countermeasures in Wireless Sensor Networks, International Journal of Advanced Research in Computer Science, Vol. 8(5), May-June 2017

[89] J. Luo, C. Shan, J. Cai, and Y. Liu, "SS symmetry IoT Application-Layer Protocol Vulnerability Detection using Reverse Engineering," 2018.

[90] T. Kitagawa, M. Hanaoka, K. Kono, AspFuzz: A state-aware protocol fuzzer based on application-layer protocols. In Proceedings of the IEEE Symposium on Computers and Communications, Riccione, Italy, 22–25 June 2010; pp. 202–208.

[91] S. Gorbunov and A. Rosenbloom, AutoFuzz: Automated Network Protocol Fuzzing Framework. Int. J. Comput. Sci. Netw. Secur. 2010, 10, 239–245.

[92] P. Tsankov, M.T. Dashti, and D. Basin, SecFuzz: Fuzz-testing security protocols. In Proceedings of the International Workshop on Automation of Software Test, Zurich, Switzerland, 2–3 June 2012; pp. 1–7

[93] M. Noor and W. H. Hassan, "Current research on Internet of Things (IoT) security : A survey Current research on Internet of Things (IoT) security : A survey," *Comput. Networks*, vol. 148, no. December, pp. 283–294, 2018.

[94] D.E. Kouicem , A. Bouabdallah , H. Lakhlef , Internet of things security: A top– down survey, *Comput. Networks 141 (2018) 199–221*

[95] H.Z. Yuchen Yang , Longfei Wu , Guisheng Yin , Lijie Li , A survey on security and privacy issues in internet-of-things*, 2015 10th Int. Conf. Internet Technol. Secur. Trans., 4, 2015, pp. 202–207 .*

[96] M.A .Ferrag , L.A . Maglaras , H. Janicke, and J. Jiang, Authentication Protocols for Internet of Things: A Comprehensive Survey*, Security and Communication Networks 2017 (2017) 1–41.*