



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

**ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**Δημιουργία ενός API για την υποστήριξη κεντρικά
χρονοπρογραμματιζόμενων πολυνηματικών penetration tests
στο ZAP.**

Παύλος Χ. Τζιάνος

Επιβλέποντες:

**Κιαγιάς Άγγελος, Επίκουρος Καθηγητής
Σμαραγδάκης Γιάννης, Αναπληρωτής Καθηγητής**

ΑΘΗΝΑ

ΜΑΪΟΣ 2016

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

**Δημιουργία ενός API για την υποστήριξη κεντρικά
χρονοπρογραμματιζόμενων πολυνηματικών penetration tests.**

**Παύλος Χ. Τζιάνος
Α.Μ.: 1115200700162**

Επιβλέποντες:

**Κιαγιάς Άγγελος, Επίκουρος Καθηγητής
Σμαραγδάκης Γιάννης, Αναπληρωτής Καθηγητής**

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας εργασίας είναι να αναλύσει τους μεσοπρόθεσμους και μακροπρόθεσμους στόχους όπως έχουν αυτοί διατυπωθεί από την κοινότητα χρηστών και μηχανικών λογισμικού του Zed Attack Proxy(ZAP στο εξής για συντομία), και να εξερευνήσει το τοπίο όσον αφορά παρόμοια εργαλεία που είναι διαθέσιμα αυτή τη στιγμή, και βάσει αυτών να προτείνει και να υλοποιήσει επεκτάσεις στην υποδομή και στην λειτουργικότητα του εργαλείου, που κρίνονται ως αναγκαίες για τον εκσυγχρονισμό της εφαρμογής. Η μεθοδολογία που ακολουθήθηκε είναι η εξής: στο πρώτο στάδιο καταγράφηκαν με ακρίβεια τα σχέδια για την μελλοντική πορεία του ZAP, στην συνέχεια σχεδιάστηκε η δομή του API και, τέλος, υλοποιήθηκε το νέο API. Το κύριο αποτέλεσμα της εργασίας είναι η μεταφορά προϋπάρχουσας λειτουργικότητας πάνω στις νέες δομές προκειμένου να αποδειχθεί έμπρακτα η σωστή λειτουργία τους αλλά και να αποτελέσουν παράδειγμα του πώς θα μπορεί να χρησιμοποιηθεί το API για να δημιουργηθεί καινούργια ή να επεκταθεί η υπάρχουσα λειτουργικότητα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ανάπτυξη Λογισμικού, Ασφάλεια Υπολογιστικών Συστημάτων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Vulnerability assessemnt, penetration testing, πολυνηματικές εφαρμογές, ασφάλεια υπολογιστών

Ευχαριστώ όσους πίστεψαν και συνεχίζουν να πιστεύουν σε μένα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ.....	6
1. ΕΙΣΑΓΩΓΗ.....	7
1.1 Σημασία της Ασφάλειας Υπολογιστών στον Σύγχρονο Κόσμο.....	7
1.2 Ορισμός της Ασφάλειας Υπολογιστικών Συστημάτων.....	10
1.3 Ορισμός του Vulnerability Assessment και του Penetration Testing σε διαδικτυακές εφαρμογές.....	11
1.4 Ο ρόλος των αυτοματοποιημένων εργαλείων.....	13
1.5 Μία σύντομη περιγραφή του Zed Attack Proxy.....	13
1.6 Σύγκριση του ZAP με τα υπόλοιπα εργαλεία.....	17
1.7 Τι προσθέτει αυτή η εργασία.....	19
2. ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΤΟΥ API.....	20
2.1 Απαιτήσεις που πρέπει να καλύπτει το νέο API.....	20
2.2 Κεντρική διαχείριση και χρονοπρογραμματισμός των νημάτων με τον GeneralWorker.....	22
2.3 Εκτέλεση ενός νήματος GeneralWorker.....	24
2.4 Συγχρονισμός της λειτουργίας των νημάτων GeneralWorker.....	26
2.5 Μηχανισμός αίτησης ενός νήματος για χρονοπρογραμματισμό και εκτέλεση μίας εργασίας.....	30
2.6 Επικοινωνία των αποτελεσμάτων και της κατάστασης ενός νήματος GeneralWorker με το περιβάλλον.....	32
2.7 Εκτέλεση κάποιου είδους εργασίας πάνω σε ένα κόμβο της ιστοσελίδας.....	35
3. ΧΡΗΣΗ.....	37
3.1 Χρήση της κλάσης GeneralWorkerConcurrencyTester για την δημιουργία tests για το αντικείμενο ThreadState και των logs του.....	37
3.2 Δημιουργία του DownloadWorker και διεύρυνση του test suite.....	38
3.3 Χρήση του GeneralWorker για την ανακατασκευή του ActiveScan extension.....	39
3.4 Δημιουργία ενός CLI για την επίδειξη του νέου ActiveScanner.....	42
4. ΣΧΕΔΙΑΓΡΑΜΜΑ ΚΛΑΣΕΩΝ.....	46
ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ.....	47
ΣΥΝΤΗΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ.....	48
ΑΝΑΦΟΡΕΣ.....	49

ΠΡΟΛΟΓΟΣ

Η παρουσίαση της μελέτης που ακολουθεί εντάσσεται στα πλαίσια της πτυχιακής εργασίας του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών (ΕΚΠΑ).

Η ιδέα της μελέτης και της υλοποίησης του API προέκυψε κατόπιν ενασχόλησης μου με τα μέλη της ανοικτής κοινότητας των μηχανικών λογισμικού του Zed Attack Proxy (ZAP) κατά την διάρκεια της εργασίας μου στο CERN που με βοήθησαν να αντιληφθώ τις δυσκολίες αλλά και την αναγκαιότητα της

Η υλοποίηση που προέκυψε ελπίζω ότι θα οδηγήσει στην επέκταση των δυνατοτήτων και της εργαλειοθήκης του ZAP που με τη σειρά του θα βοηθήσει τους ήδη υπάρχοντες χρήστες να διεξάγουν πιο ολοκληρωμένους ελέγχους ασφαλείας και θα προσελκύσει ακόμα περισσότερους νέους χρήστες στην πλατφόρμα και, ίσως, και στο χώρο γενικά της ασφάλειας υπολογιστικών συστημάτων.

1. ΕΙΣΑΓΩΓΗ

1.1 Σημασία της Ασφάλειας Υπολογιστών στον Σύγχρονο Κόσμο.

Είναι αναμφισβήτητο γεγονός ότι στον σύγχρονο κόσμο έχουν μείνει ελάχιστες πλευρές της καθημερινότητας ενός ανθρώπου που να μην βασίζονται έστω και σε κάποιο μικρό βαθμό στην χρήση ηλεκτρονικών υπηρεσιών και εφαρμογών. Από τον τρόπο που επικοινωνούν οι πολίτες, την διακυβέρνηση των κρατών μέχρι και τις τραπεζικές συναλλαγές και το εμπόριο, οι εφαρμογές είναι πολλές στον αριθμό και όλες οι μελέτες μας οδηγούν στο συμπέρασμα ότι πρόκειται να αυξηθούν δραστικά τα επόμενα χρόνια. Οι λόγοι για την εξάπλωση και την διείσδυση των ηλεκτρονικών εφαρμογών είναι σχεδόν εξίσου πολλοί όσο και οι ίδιες οι εφαρμογές και η ανάλυση όλων των παραγόντων, ξεφεύγει από τα πλαίσια της παρούσας εργασίας, αλλά μπορούμε να αναφέρουμε επιγραμματικά τους κυριότερους.

Η πρόοδος της τεχνολογίας, των γλωσσών προγραμματισμού και των συστημάτων λογισμικού έχουν δώσει την δυνατότητα να δημιουργηθούν πολλές καινοτόμες υπηρεσίες και προϊόντα τα οποία θα θεωρούνταν προϊόν επιστημονικής φαντασίας μόλις πριν από λίγες δεκαετίες. Ταυτόχρονα, πολλές εταιρίες και οργανισμοί του δημόσιου και του ιδιωτικού τομέα χρησιμοποιούν ηλεκτρονικές υπηρεσίες και εφαρμογές προκειμένου να μπορούν να διευκολύνουν την επικοινωνία με τους πελάτες και τους πολίτες. Ενώ έχει αλλάξει σημαντικά και η παραγωγική διαδικασία, αφού πολλές εταιρίες έχουν δημιουργήσει πλέον ηλεκτρονικές υποδομές για την αυτοματοποίηση όσο το δυνατόν μεγαλύτερου κομματιού της, προκειμένου να επιτύχουν μείωση του κόστους παραγωγής και αύξηση της παραγωγικότητας. Από την πλευρά των πολιτών, η πρόοδος της τεχνολογίας τους και η ανάπτυξη των ενσύρματων και ασύρματων δικτύων επιτρέπει πλέον στους πολίτες να είναι συνεχώς συνδεδεμένοι στο διαδίκτυο. Με αυτό τον τρόπο, έχουν επίσης πρόσβαση και σε μία ευρεία γκάμα εφαρμογών και υπηρεσιών που έχουν σχέση με την επικοινωνία, το εμπόριο, τις διατραπεζικές συναλλαγές από όπου και αν βρίσκονται.

Οι εξελίξεις της τεχνολογίας έχουν αλλάξει δραστικά τον τρόπο που ζουν, εργάζονται και επικοινωνούν οι πολίτες σε όλο τον κόσμο. Εκτός όμως από τις ευεργετικές τους επιπτώσεις, έχουν δημιουργήσει και μία νέα πραγματικότητα στον τομέα της ασφάλειας. Διότι, οι περισσότερες υπηρεσίες με τις οποίες έρχονται σε επαφή οι πολίτες συλλέγουν δεδομένα, ηθελημένα ή άθελα, σχετικά με την δραστηριότητα και την ταυτότητα

του χρήστη. Για παράδειγμα, μία υπηρεσία web banking θα κρατάει, τουλάχιστον για όσο την χρησιμοποιεί ο χρήστης, τα στοιχεία τα οποία εισήγαγε προκειμένου να αποκτήσει πρόσβαση στα στοιχεία του λογαριασμού του. Στις περισσότερες εφαρμογές web banking, τα ίδια στοιχεία μπορούν να χρησιμοποιηθούν προκειμένου ένας χρήστης να δώσει εντολή για να γίνουν συναλλαγές και μεταφορές χρημάτων. Άλλες εφαρμογές, μέσα από τις οποίες μπορούν να γίνουν αγοραπωλησίες, ζητούν από τους χρήστες να δώσουν πληροφορίες όπως τον αριθμό της πιστωτικής τους κάρτας. Δεδομένα, δηλαδή, τα οποία είναι πολύ ευαίσθητα και ως εκ τούτου ένας πολύ δελεαστικός στόχος για κακόβουλους παράγοντες. Όμως, ακόμα και αν μία εφαρμογή δεν χρειάζεται να αποθηκεύσει ευαίσθητα δεδομένα, πολλές φορές παρακολουθεί και αποθηκεύει διάφορες πληροφορίες σχετικά με τον χρήστη προκειμένου να διεκπεραιώσει την λειτουργία της, όπως το ιστορικό της πλοήγησής του στο διαδίκτυο, προτιμήσεις σε προϊόντα και άλλες προσωπικές πληροφορίες όπως δεδομένα σχετικά με την υγεία του, οι οποίες αν συγκεντρωθούν, μπορούν να χρησιμοποιηθούν από αλγόριθμους για να δημιουργηθεί ένα προφίλ του χρήστη το οποίο μπορεί να βοηθήσει ένα κακόβουλο παράγοντα να μαντέψει κάποιους από τους κωδικούς και τα διαπιστευτήρια του.

Αντίστοιχα προβλήματα ασφαλείας αντιμετωπίζουν και δημόσιες και ιδιωτικές εταιρείες, οι οποίες βασίζονται σε βάσεις δεδομένων για να αποθηκεύσουν ένα μεγάλο όγκο πληροφοριών, πολλές από τις οποίες μπορεί να είναι ιδιαίτερες κρίσιμες για την επιβίωσή τους, όπως οικονομικά δεδομένα, σχέδια για το μέλλον και πληροφορίες σχετικά με τα προϊόντα και τις υπηρεσίες που παράγει ο οργανισμός. Μία άλλη κατηγορία δεδομένων που πολύ συχνά έχει στην κατοχή της μία εταιρεία είναι τα προσωπικά δεδομένα των εργαζομένων της και των πελατών της. Η παραβίαση των συστημάτων και η διαρροή πληροφοριών, λοιπόν, μπορεί να έχει πολύ μεγάλες επιπτώσεις για τους πολίτες.

Η σημασία της ασφάλειας στα υπολογιστικά συστήματα γίνεται πλήρως αντιληπτή αν πάρουμε υπόψη τον βαθμό στον οποίο βασίζονται σε ηλεκτρονικές υπηρεσίες για την εύρυθμη λειτουργία τους τα σύγχρονα κράτη. Διότι τα ηλεκτρονικά συστήματα ενός κράτους αποθηκεύουν πληροφορίες όπως τους αριθμούς φορολογικών μητρώων όλων των πολιτών, κρατικά μυστικά σχετικά με την εξωτερική πολιτική, πληροφορίες σχετικές με την δράση των μυστικών υπηρεσιών και στρατιωτικά μυστικά. Δηλαδή, πληροφορίες τεράστιας αξίας για ένα κράτος και αποτελούν, αντίστοιχα, ένα πολύ μεγάλο στόχο για άλλες κυβερνήσεις ή εγκληματικές οργανώσεις.

Όλα τα παραπάνω μας οδηγούν στο ίδιο συμπέρασμα που έχουν οδηγηθεί εδώ και κάποια χρόνια ήδη αρκετές υπηρεσίες ασφαλείας: ότι ο βαθμός της στήριξης των σύγχρονων κρατικών και ιδιωτικών οργανισμών σε ηλεκτρονικές υπηρεσίες είναι τέτοιος ώστε να έχει αναχθεί σε ζήτημα εθνικής ασφάλειας η σωστή και ασφαλή λειτουργία των

ηλεκτρονικών υπηρεσιών τους και η προστασία των βάσεων δεδομένων τους. Αυτό αποδεικνύεται και από το γεγονός ότι η Ευρωπαϊκή Ένωση, για παράδειγμα, έχει δημιουργήσει ήδη από το 2004 την Υπηρεσία για την Ασφάλεια Δικτύων και Πληροφοριών (European Union Agency for Network and Information Security - ENISA), που σκοπός της είναι να συμβουλεύει τα κράτη μέλη της σε θέματα ασφαλείας υπολογιστικών συστημάτων και οι ΗΠΑ έχουν δημιουργήσει αντίστοιχα την Αμερικάνικη Επιτροπή Ασφάλειας Εθνικών Τηλεπικοινωνιακών και Πληροφοριακών Συστημάτων (American National Security Telecommunications and Information Systems Security Committee - NSTISSC). Και πολλές άλλες κυβερνήσεις ανά τον κόσμο πλέον έχουν κοινοποιήσει και ανανεώνουν τακτικά την στρατηγική τους για την ασφάλεια των Αν αναλογιστεί κανείς και το γεγονός ότι αυτές οι υπηρεσίες φιλοξενούν στις βάσεις δεδομένων τους και ένα τεράστιο αριθμό , πολλές φορές ευαίσθητων, προσωπικών δεδομένων των πολιτών όπως ιατρικές πληροφορίες και οικονομικές συναλλαγές, καταλαβαίνει πόση αξία μπορεί να έχουν αυτές οι πληροφορίες για κακόβουλους παράγοντες.

Η σημασία της ασφάλειας υπολογιστικών συστημάτων γίνεται φανερή αν ανατρέξουμε στο πρόσφατο παρελθόν για σημαντικά γεγονότα που έχουν σχέση με επιθέσεις σε συστήματα. Το 2010 ανακαλύφθηκε ένα ιός τύπου worm στα συστήματα οδήγησης των φυγόκεντρων σε μία από τις μονάδες εμπλουτισμού ουρανίου του Ιράν. Οι ειδικοί που εξέτασαν τον ιό αποφάνθηκαν ότι αποτελεί τον πιο πολύπλοκο ιό που έχει ποτέ ανακαλυφθεί. Από τότε έχουν υπάρξει και άλλες επιθέσεις όχι μόνο σε κρατικούς φορείς αλλά και σε ιδιώτες, όπως τον Νοέμβριο του 2014 οπότε και σημειώθηκε μια πολύ μεγάλη επίθεση εναντίον των συστημάτων της Sony και στην συνέχεια πολλά από τα δεδομένα και τα εσωτερικά email της εταιρείας δόθηκαν στην δημοσιότητα, προκαλώντας ένα μεγάλο πλήγμα στην εικόνα της Sony. Για την επίθεση στα συστήματα του Ιράν πολλοί ειδικοί υπέδειξαν ως υπαίτιες τις στρατιωτικές υπηρεσίες των ΗΠΑ και του Ισραήλ ενώ για την επίθεση στην Sony κατηγορήθηκε η κυβέρνηση της Βόρειας Κορέας ότι την οργάνωσε. Και στις δύο περιπτώσεις οι κατηγορούμενες κυβερνήσεις αρνήθηκαν οποιαδήποτε γνώση. Δεν είναι αναγκαίο όμως ότι όλες οι επιθέσεις υποκινούνται από κυβερνήσεις, όπως διαπίστωσαν πολλοί σταρ του Hollywood καθ' όλη την διάρκεια του 2014 όταν είδαν προσωπικές τους φωτογραφίες και βίντεο να κοινοποιούνται στο διαδίκτυο από μία άγνωστη ομάδα χάκερ που εκμεταλλεύθηκε κενά ασφαλείας στο λογισμικό iCloud της Apple και υπέκλεψε τους κωδικούς πρόσβασης των χρηστών.

Το συμπέρασμα που προκύπτει από όλα τα παραπάνω είναι ότι στον σύγχρονο κόσμο, η ασφάλεια υπολογιστών είναι ένα θέμα το οποίο μπορεί να κρίνει την επιβίωση μίας εταιρείας ή ακόμα και ενός ολόκληρου κράτους και μπορεί να έχει πολύ μεγάλες επιπτώσεις στην ζωή των πολιτών. Είναι αναγκαίο, λοιπόν, για τους ανθρώπους που συντηρούν υπολογιστικά συστήματα να τηρούν αμυντική στάση και να φροντίζουν

συνέχεια να ενισχύουν τα μέτρα ασφαλείας τους. Υπάρχουν αρκετοί άνθρωποι οι οποίοι είναι εξειδικευμένοι στον τομέα της ασφάλειας υπολογιστών και εργάζονται για εταιρείες και κρατικούς οργανισμούς είτε ως υπάλληλοι είτε ως σύμβουλοι. Πλέον, όμως, έχουν αρχίσει και κάνουν την εμφάνιση τους κάποια εργαλεία τα οποία αυτοματοποιούν πολλές εργασίες που μέχρι πριν από λίγο καιρό μπορούσαν να εφαρμοστούν μόνο από ειδικούς στον τομέα και δίνουν την δυνατότητα σε μη εξειδικευμένους προγραμματιστές να ανακαλύψουν κάποια κενά ασφαλείας. Ένα από αυτά τα εργαλεία είναι και το Zed Attack Proxy το οποίο είναι ένα από τα πιο παλιά και ώριμα εργαλεία στον χώρο του.

1.2 Ορισμός της Ασφάλειας Υπολογιστικών Συστημάτων.

Είναι μάλλον δύσκολο να βρούμε ένα επίσημο ορισμό της ασφάλειας υπολογιστικών συστημάτων, αλλά ένας επαρκής προέρχεται από το βιβλίο UNIX System Security Tools [3], όπου η ασφάλεια υπολογιστών ορίζεται ως η συνεχόμενη και πλεονάζουσα εφαρμογή προστασίας για την εμπιστευτικότητα και την ακεραιότητα των πληροφοριών και των πόρων του συστήματος έτσι ώστε ένας μη εξουσιοδοτημένος χρήστης θα πρέπει να δαπανήσει ένα μη αποδεκτό ποσό χρόνου ή χρήματος ή να πάρει πάρα πολύ μεγάλο ρίσκο προκειμένου να υπερνικήσει τις άμυνες του συστήματος, με τελικό σκοπό να υπάρχει αρκετή εμπιστοσύνη στο σύστημα ώστε να του εμπιστευθούν οι χρήστες ευαίσθητες πληροφορίες.

Για να επιτευχθούν όλοι οι παραπάνω στόχοι έχουν δημιουργηθεί τεχνικές για την κατασκευή και την σχεδίαση των υπολογιστικών συστημάτων που προλαμβάνουν πολλά κενά ασφαλείας ενώ χρησιμοποιείται μία ευρεία γκάμα εργαλείων όπως τα firewalls, intrusion detection systems και λογισμικό anti virus για να ενισχυθεί το σύστημα απέναντι σε επιθέσεις. Αλλά ακόμα και αν ένα σύστημα σχεδιαστεί και χτιστεί από την αρχή με τα καλύτερα δυνατά εργαλεία και τεχνικές κανείς δεν μπορεί να είναι σίγουρος ότι είναι ασφαλές. Πολλές φορές μάλιστα η υπερβολική σιγουριά του κατασκευαστή του συστήματος για την ασφάλεια του ίσως να είναι και το πρώτο και μεγαλύτερο κενό ασφαλείας του συστήματος.

Οι οργανισμοί που επενδύουν πολύ στην ασφάλεια, δεν σταματούν σε κανένα σημείο να δοκιμάζουν και να ανανεώνουν την άμυνα των συστημάτων τους. Ο λόγος που το κάνουν αυτό είναι ότι ο “εχθρός” και οι μέθοδοι του δεν είναι στατικοί. Καθώς περνάει ο χρόνος εξελίσσονται τα εργαλεία και οι μέθοδοι προστασίας ενός συστήματος αλλά ταυτόχρονα εξελίσσονται και οι τεχνικές και τα εργαλεία που χρησιμοποιούνται για να

αναγνωρίζονται τυχόν αδυναμίες αλλά και το πώς χρησιμοποιούνται προκειμένου να αυξηθούν οι πιθανότητες επιτυχίας μίας επίθεσης. Επίσης, οποιοσδήποτε έχει αναπτύξει ή διαχειριστεί σύγχρονες υπολογιστικές υποδομές γνωρίζει ότι πλέον όλες σχεδόν αναπτύσσονται χρησιμοποιώντας και συνδυάζοντας μία πληθώρα εργαλείων και frameworks προκειμένου να μειωθεί όσο το δυνατόν περισσότερο ο χρόνος από την σχεδίαση μέχρι την παράδοση του τελικού προϊόντος. Αυτά τα επιμέρους κομμάτια μπορεί από μόνα τους να έχουν κενά ασφαλείας και να αποτελούν την αχίλλεια πτέρνα του συστήματος. Για αυτό είναι απολύτως αναγκαίο να παρακολουθεί κανείς και τα προβλήματα που αναγνωρίζονται στα επιμέρους κομμάτια γιατί ένα οποιοδήποτε σύστημα είναι τόσο ισχυρό όσο το πιο αδύναμο κομμάτι του.

1.3 Ορισμός του Vulnerability Assessment και του Penetration Testing σε διαδικτυακές εφαρμογές.

Το vulnerability assessment ορίζεται ως η διαδικασία με την οποία δοκιμάζεται συστηματικά και ενεργά μία υποδομή εν λειτουργία προκειμένου να διαπιστωθούν τυχόν αδυναμίες που μπορεί να είναι παρούσες. Ένας απολογισμός των αδυναμιών ενός συστήματος περιλαμβάνει την χαρτογράφηση του δικτύου και των συστημάτων που είναι συνδεδεμένα σε αυτό, την αναγνώριση των υπηρεσιών και των εκδόσεων αυτών των υπηρεσιών [1],[2]. Η ανάλυση των κάθε υποσυστήματος ξεχωριστά και ο τρόπος με τον οποίο αλληλεπιδρούν χρησιμοποιείται για την δημιουργία ενός ολοκληρωμένου καταλόγου των αδυναμιών.

Το Penetration Testing αποτελεί μια πιο ευρεία και συστηματική διαδικασία από το vulnerability assessment. Η αναγνώριση των αδυναμιών είναι το πρώτο κομμάτι ενός penetration test. Αφότου ολοκληρωθεί το vulnerability assessment, το επόμενο βήμα είναι η προσπάθεια αξιοποίησης των αδυναμιών που ανακαλύφθηκαν για να εξακριβωθεί η ύπαρξη και η χρησιμότητά τους. Όταν αποκαλυφθούν οι αδυναμίες σε ένα σύστημα οι ειδικοί που ασχολούνται με το testing προσπαθούν να χρησιμοποιήσουν τα κενά ασφαλείας προκειμένου να δημιουργήσουν νέα και πιο μεγάλα κενά με στόχο σταδιακά να αποκτήσουν υψηλότερα επίπεδα εξουσιοδότησης μέσα στο σύστημα και μεγαλύτερη πρόσβαση στους πόρους του ή ακόμα και τον πλήρη έλεγχό του. Στο τέλος της διαδικασίας, είναι δυνατό για τον ειδικό να υπολογίσει τις επιπτώσεις που μπορεί να προκαλέσουν στο σύστημα σε περίπτωση που χρησιμοποιηθούν οι διάφορες αδυναμίες από κακόβουλους παράγοντες και να δημιουργήσει μία αναφορά με τα κενά ασφαλείας και τους τρόπους με τους οποίους μπορούν να εξαλειφθούν αυτά προτού χρησιμοποιηθούν

από κακόβουλους παράγοντες.

Πιο αναλυτικά, οι στόχοι του penetration testing είναι οι εξής:

1. Να απαριθμηθούν τα κενά ασφαλείας του συστήματος.
2. Να διευκρινιστεί η δυνατότητα ή μη χρήσης μίας συγκεκριμένης τεχνικής επίθεσης εναντίον του συστήματος.
3. Να αναγνωριστούν σημαντικές αδυναμίες που μπορούν να προκύψουν από τον συνδυασμό μιας σειράς μικρότερων κενών ασφαλείας με κάποια συγκεκριμένη σειρά.
4. Να γίνει μία ανάλυση του εύρους της ζημιάς που μπορεί να προκληθεί από μία επιτυχημένη επίθεση.
5. Να πιέσει μέσα από τα συμπεράσματα του την ομάδα πίσω από μία εφαρμογή να επενδύσει περισσότερους πόρους στην καταπολέμηση των αδυναμιών που ανακαλύφθηκαν.

Σε σύγκριση με ένα penetration test, το vulnerability assesement ενός συστήματος δεν στοχεύει στο να διεισδύσει μέσα στο σύστημα και δεν χρειάζεται το ίδιο επίπεδο τεχνικών ικανοτήτων. Ταυτόχρονα, θα πρέπει να τονιστεί ότι ίσως είναι αδύνατον να γίνει ένας τόσο ολοκληρωμένος και λεπτομερής απολογισμός ώστε να αναγνωριστούν οι πιο επικίνδυνες αδυναμίες.

Και στις δύο περιπτώσεις όμως, οι ενέργειες του tester είναι εξουσιοδοτημένες από το άτομο ή την εταιρεία στην οποία ανήκει η εφαρμογή και γίνεται σε συνεργασία με τους ανθρώπους που την συντηρούν με σκοπό να χαρτογραφηθούν χωρίς κίνδυνο κενά ασφαλείας και να καλυφθούν προτού έχουν την δυνατότητα να τα χρησιμοποιήσουν κακόβουλοι παράγοντες. Υπάρχει πλέον γίνονται χρησιμοποιώντας ένα συνδυασμό αυτοματοποιημένων εργαλείων αλλά και χειρωνακτικής εργασίας προκειμένου να αποκαλυφθούν σημεία μη εξουσιοδοτημένου εισόδου

Ένα μεγάλο κομμάτι του penetration testing είναι περισσότερο τέχνη παρά επιστήμη. Η αποτελεσματικότητα του penetration testing εξαρτάται από τις ικανότητες και την εμπειρία των ατόμων που κάνουν τα tests. Οι penetration testers χρειάζονται να έχουν καλή αντίληψη των βασικών αρχών τις ασφάλειας υπολογιστικών συστημάτων αλλά, χρειάζονται και μία σχεδόν εγκυκλοπαιδική γνώση των προϊόντων και των συστημάτων, γνώση που μπορεί σε πολλούς να φανεί ότι μπορεί να μην έχει καμία ιδιαίτερη αξία σε

σχέση με το αντικείμενο της εργασίας τους.

Το Penetration Testing είναι ένα πάρα πολύ σημαντικό βήμα στην διαδικασία της ανάπτυξης ενός ασφαλούς συστήματος ή προϊόντος. Ενώ πολλές σύγχρονες επιχειρήσεις ορίζουν το penetration testing σαν την εφαρμογή αυτοματοποιημένων scanner για αδυναμίες στο δίκτυο μίας ιστοσελίδας εν λειτουργία, το πραγματικό penetration testing είναι κάτι πολύ περισσότερο από αυτό. Το πραγματικό Penetration Testing φτάνει στα άκρα του όχι μόνο την λειτουργία αλλά και την υλοποίηση και την σχεδίαση ενός συστήματος ή ενός προϊόντος.

1.4 Ο ρόλος των αυτοματοποιημένων εργαλείων

Η εποχή που το penetration testing μπορούσε να γίνει μόνο χειρωνακτικά από ένα ειδικό έχει παρέλθει. Εδώ και χρόνια έχει εμφανιστεί μία πληθώρα εργαλείων τα οποία βοηθούν με διάφορους τρόπους να πραγματοποιούνται τα penetration tests πιο γρήγορα, με μεγαλύτερη ακρίβεια και με μεγαλύτερο βαθμό αυτοματοποίησης. Ο σημερινός penetration tester έχει την δυνατότητα να επιλέξει ανάμεσα σε εργαλεία γενικής χρήσης, που εξετάζουν μια υπηρεσία ή μια εφαρμογή για ένα μεγάλο εύρος κενών ασφαλείας που στοχεύονται από τις πιο γνωστές τεχνικές επίθεσης, μέχρι εξαιρετικά εξειδικευμένα εργαλεία που στοχεύουν συγκεκριμένους τύπους συστημάτων ή τεχνικών επιθέσεων. Κάποια από αυτά όμως προσφέρουν πολλές δυνατότητες προσαρμογής και επέκτασης, έτσι ώστε στα χέρια ενός εξειδικευμένου ατόμου να μπορούν να εκτελέσουν tests με πολύ μεγάλη ακρίβεια.

Πίσω από την αύξηση του αριθμού των εργαλείων αυτού του είδους κρύβεται η σημαντική αύξηση της ζήτησης τους. Η ζήτηση πηγάζει από το γεγονός ότι πλέον υπάρχει ένας τεράστιος αριθμός επιχειρήσεων οι οποίες προσπαθούν να εξασφαλίσουν όσο το δυνατόν καλύτερο επίπεδο ασφάλειας για της διαδικτυακές υπηρεσίες τους με το ελάχιστο δυνατόν κόστος. Το εργαλεία που θα συζητηθούν στην συνέχεια επιτρέπουν στους χρήστες να δημιουργήσουν και να αυτοματοποιήσουν μία σειρά από tests με μικρό οικονομικό κόστος.

1.5 Μία σύντομη περιγραφή του Zed Attack Proxy.

Το Zed Attack Proxy είναι ένα πλήρως open source, cross platform, penetration testing και vulnerability assesement εργαλείο. Ξεκίνησε σαν ένα διαφορετικό project με το όνομα ParosProxy, το οποίο ήταν ένα εργαλείο που όπως περιγράφει και το όνομα του, είναι ένα proxy για διαδικτυακά πακέτα. Η πρώτη έκδοση άρχισε να αναπτύσσεται το 2004 και ήταν γραμμένη ολόκληρη σε Java. Μετά το 2007 ο κώδικας που υπήρχε χρησιμοποιήθηκε σαν βάση προκειμένου να ξεκινήσει η κατασκευή του Zed Attack Proxy, δεδομένου ότι ένα μεγάλο κομμάτι της λειτουργίας βασίζεται στον έλεγχο των πακέτων που ανταλλάσσονται ανάμεσα στον browser του χρήστη και στην διαδικτυακή εφαρμογή. Από την αρχή, το ZAP αναπτυσσόταν ως project υπό την αιγίδα του OWASP.

Ο pen tester το εγκαθιστά τοπικά και το χρησιμοποιεί σαν proxy ανάμεσα στον browser του και στην υπηρεσία την οποία θέλει να εξετάσει και μπορεί να εκτελέσει αυτοματοποιημένα tests μέσω του γραφικού περιβάλλοντος. Ο σκοπός του ZAP είναι να αποτελεί ένα εργαλείο που να είναι εύκολο και φιλικό στην χρήση έτσι ώστε να μπορεί να χρησιμοποιηθεί από ένα αρχάριο χρήστη, που μπορεί να θέλει να κάνει μια εισαγωγή στον τομέα του penetration testing, αλλά να μπορεί να καλύπτει και τις ανάγκες ενός επαγγελματία pen tester που αποζητάει μια ευρεία γκάμα δυνατοτήτων και ένα μεγάλο βαθμό ελευθερίας προσαρμογής των test στην εκάστοτε περίπτωση. Όλη η λειτουργικότητα που προσφέρεται, παρέχεται με την μορφή plugins, ένας αριθμός των οποίων είναι διαθέσιμος με την εγκατάσταση του εργαλείου, ενώ ταυτόχρονα προσφέρεται και μία αγορά όπου ένας χρήστης μπορεί να βρει περισσότερα plugins. Επειδή είναι open-source όλος ο κώδικας και τα plugins του ZAP, είναι εύκολο για ένα οποιονδήποτε χρήστη με γνώσεις προγραμματισμού να επεκτείνει την λειτουργία του εργαλείου αν δεν βρει αυτό που ψάχνει, κάτι που έχει συμβάλει πολύ στην δημοφιλία του.

Ακολουθεί μία σύντομη παρουσίαση των πιο γνωστών εργαλείων και για automated penetration και vulnerability assesement testing.

1. Ο πιο συνηθισμένος τρόπος για να εκτελεστεί ένα penetration test είναι με ένα εργαλείο το οποίο εγκαθίσταται τοπικά στον υπολογιστή του tester.
 1. Το Metasploit είναι ένα από τα πιο γνωστά εργαλεία για penetration testing, το οποίο είναι βασισμένο στην έννοια του exploit, δηλαδή κώδικα που μπορεί

να προσπεράσει το μέτρα ασφαλείας και να εισέλθει σε ένα σύστημα. Τελικός στόχος είναι ότι μετά την είσοδο του μέσα στο σύστημα, να εκτελέσει κάποιον άλλο κώδικα με αυξημένα διαπιστευτήρια ασφαλείας. Αυτή του η δυνατότητα το κάνει ένα από τα από ολοκληρωμένα και γνωστά frameworks για penetration testing. Έχει μία ευρεία γκάμα περιπτώσεων χρήσης αφού είναι δυνατό να εκτελέσει tests σε διαδικτυακές εφαρμογές, δίκτυα και servers. Έχει γραφικό περιβάλλον αλλά και CLI και αυτή τη στιγμή οι βασικές εκδόσεις του είναι διαθέσιμες επί πληρωμή και υπάρχει μία δωρεάν έκδοση η οποία όμως δεν έχει την δυνατότητα για αυτοματοποιημένα tests και δεν έχει γραφικό περιβάλλον.

2. Το Burp Suite είναι επίσης ένα αρκετά γνωστό cross-platform εργαλείο που ο βασικός κορμός της λειτουργικότητας του προσφέρεται για την ανακάλυψη κενών ασφαλείας σε διαδικτυακές εφαρμογές, ενώ περιλαμβάνει επίσης και το εργαλείο Intruder που δίνει την δυνατότητα στον χρήστη να χρησιμοποιήσει κάποια κενά ασφαλείας για να εκτελέσει κώδικα στον ευάλωτο server . Είναι ένα closed source εργαλείο που διαθέτει όμως ένα framework προκειμένου ένας χρήστης να υλοποιήσει νέα plugins που θα επεκτείνουν την λειτουργικότητα του. Ένα σημαντικό χαρακτηριστικό του είναι ότι έχει δύο εκδόσεις: την δωρεάν που προσφέρει ένα υποσύνολο της λειτουργικότητας και την επί πληρωμή έκδοση που μέσα στην οποία βρίσκεται και το Intruder εργαλείο καθώς και δυνατότητες για αυτοματοποίηση των tests.
3. Το CANVAS της Immunity είναι ένα closed-source penetration testing tool που διαθέτει την μια ευρεία γκάμα από έτοιμα exploits, πάνω από 400 σύμφωνα με την εταιρεία που το παράγει, σχεδιασμένα για να χρησιμοποιηθούν εναντίον διαδικτυακών υπηρεσιών, ασύρματων συστημάτων και δικτύων. Επίσης, είναι το μόνο εργαλείο από αυτά που παρουσιάζονται το οποίο είναι δεν έχει καμία έκδοση που να είναι δωρεάν.
4. Το Sqlmap είναι ένα open-source εργαλείο, το οποίο είναι εξειδικευμένο στην εύρεση και εκμετάλλευση κενών ασφαλείας σε συστήματα σχεσιακών βάσεων δεδομένων. Είναι cross-platform και διαθέτει πλήρη υποστήριξη για μία μεγάλη ομάδα συστημάτων σχεσιακών βάσεων δεδομένων όπως η

MySQL, OracleSQL, MicrosoftSQL, SQLite και αρκετές άλλες. Επειδή απευθύνεται σε μία συγκεκριμένη ομάδα εργαλείων, είναι περιορισμένο και το εύρος των επιθέσεων τις οποίες μπορεί να εξομειώσει. Πιο συγκεκριμένα, μπορεί να χρησιμοποιηθεί για να εξεταστεί η ασφάλεια του συστήματος απέναντι σε επιθέσεις όπως η boolean-based blind query, η time-based blind query, η error-based query και οι stacked queries, ενώ δίνει την δυνατότητα στον χρήστη να αναγνωρίσει τον αλγόριθμο κατακερματισμού των κωδικών και να εκτελέσει μία dictionary-based επίθεση προκειμένου να τους σπάσει.

5. Το SqlNinja, όπως δείχνει και το όνομα του, είναι ένα ακόμα εργαλείο για penetration testing ειδικευμένο σε συστήματα σχεσιακών βάσεων δεδομένων. Αν και στην σελίδα του αναφέρει ότι αντιμετωπίζει προβλήματα σταθερότητας, ο αριθμός των χρηστών δίνει μια διαφορετική εικόνα. Το εργαλείο προσφέρει την δυνατότητα αναγνώρισης του τύπου και της έκδοσης του συστήματος που χρησιμοποιείται, την εξαγωγή δεδομένων από την βάση, την αποστολή εκτελέσιμων αρχείων στον server, διάφορες τεχνικές για έναρξη συνεδρίας shell καθώς και την δυνατότητα να συνδεθεί με το Metasploit που αναφέρθηκε πιο πριν.
6. Το Arachni είναι ένα από τα πιο καινούργια εργαλεία που έχουν εμφανιστεί στον χώρο. Είναι γραμμένο σε Ruby και χρησιμοποιείται για την εύρεση κενών ασφαλείας σε web applications. Διαθέτει ένα web UI και CLI και την δυνατότητα να χρησιμοποιηθεί από πολλάπλους χρήστες ταυτόχρονα και είναι σχεδιασμένο για να τρέχει σαν distributed application. Ένα μοναδικό του χαρακτηριστικό είναι ότι έχει την δυνατότητα να αντιλαμβάνεται δυναμικά τις αλλαγές στην εφαρμογή καθώς εκτελεί τους ελέγχους και να προσαρμόζει δυναμικά την προτεραιότητα των προβλημάτων που αντιλαμβάνεται.
2. Εκτός από τα εργαλεία που χρειάζεται να εγκατασταθούν τοπικά πλέον έχει αναπτυχθεί μια νέα κατηγορία εργαλείων οι οποίες δεν απαιτούν εγκατάσταση αλλά προσφέρονται σαν SaaS και βασίζονται σε υπολογιστικά νέφη. Ακολουθεί μία σύντομη παρουσίαση των πιο γνωστών τέτοιων υπηρεσιών.
1. Η Veracode παρέχει στους πελάτες της μία cloud-based penetration testing

πλατφόρμα. Η πλατφόρμα αυτή προσφέρει δυο διαφορετικά είδη testing:

1. Έχει την δυνατότητα να εκτελέσει κάποια αυτοματοποιημένα tests πάνω σε ένα url μιας διαδικτυακής υπηρεσίας και βάσει αυτών να δημιουργήσει μία αναφορά με τα προβλήματα που βρέθηκαν και για τρόπους επίλυσης τους.
 2. Μπορεί επίσης να εκτελέσει στατική ανάλυση στον πηγαίο κώδικα που ανεβάζει ο χρήστης και η πλατφόρμα μοντελοποιεί τον κώδικα και, σύμφωνα με την εταιρεία πίσω από το την υπηρεσία, μπορεί ανεξάρτητα από την γλώσσα του κώδικα, να ανιχνεύσει αδυναμίες και κενά ασφαλείας.
2. Η BeyondTrust προσφέρει αντίστοιχα το BeyondSaaS που είναι ένα closed-source cloud-based Vulnerability and penetration testing tool που φιλοξενείται στο Microsoft Azure. Ο χρήστης παρέχει το url της υπηρεσίας που θέλει να ελέγξει και τον τύπο των tests που θέλει να εκτελέσει και το σύστημα θα προσπαθήσει να βρει προβλήματα ασφαλείας. Στο τέλος παράγεται μία αναφορά με το τι προβλήματα βρέθηκαν και συμβουλές για το πως μπορούν να διορθωθούν καθώς και μία βαθμολογία σχετική με το πόσο σοβαρά είναι τα προβλήματα που ανακαλύφθηκαν, σύμφωνα με τους ειδικούς της εταιρείας.

1.6 Σύγκριση του ZAP με τα υπόλοιπα εργαλεία.

Σε αυτό το σημείο είναι αναγκαία η σύγκριση του ZAP με τα υπόλοιπα εργαλεία προκειμένου να ξεκαθαριστεί σε ποια σημεία υπερέχει και σε ποια υστερεί η πλατφόρμα και να αποσαφηνιστεί τι πρέπει και πόσο αναγκαίο να προστεθεί στο μέλλον.

Σε σχέση με τον ανταγωνισμό, το ZAP είναι το μόνο open-source εργαλείο με τόσο μεγάλη κοινότητα χρηστών και προγραμματιστών που ασχολούνται ενεργά με την επέκτασή του.

Αντίθετα με τα SQLMap και SQLNinja που είναι ιδιαίτερα εξειδικευμένα στο είδος

των συστημάτων και επιθέσεων τα οποία μπορούν να εκτελέσουν, το ZAP είναι το μόνο δωρεάν εργαλείο που μπορεί να εκτελέσει ένα τόσο μεγάλο αριθμό διαφορετικών tests. Επίσης, είναι το μόνο εργαλείο που μπορεί να εκτελέσει ZEST scripts, μία νέα scripting γλώσσα που αναπτύσσεται από την Mozilla προκειμένου να βοηθήσει τους pen testers να δημιουργούν penetration tests εύκολα και με μεγάλη ακρίβεια και να μπορούν να τα εκτελούν σε οποιοδήποτε εργαλείο υλοποιεί την γλώσσα.

Εκτός από τις εφαρμογές που εγκαθίστανται τοπικά σε ένα υπολογιστή παρουσιάστηκαν και δύο υπηρεσίες τύπου SaaS, η Veracode και η BeyondTrust. Αυτές οι υπηρεσίες έχουν την δυνατότητα να ελέγξουν μόνο εφαρμογές και υπηρεσίες που έχουν κάποιο δημόσιο IP. Αντίθετα το ZAP μπορεί να εγκατασταθεί σε ένα υπολογιστή μέσα σε ένα κλειστό δίκτυο και να ελέγξει εφαρμογές στις οποίες δεν υπάρχει πρόσβαση από το δημόσιο δίκτυο. Αυτό είναι ιδιαίτερα σημαντικό για εφαρμογές οι οποίες είναι υπό εξέλιξη και το penetration testing γίνεται για να διορθωθούν προβλήματα προτού προλάβουν να χρησιμοποιηθούν από κάποιον για να επιτεθεί στην εφαρμογή, όταν αυτές δημοσιευθούν.

Το ZAP είναι το μόνο εργαλείο που διαθέτει clients για το RESTful API του γραμμένους σε διάφορες γλώσσες, ενώ διαθέτει και ένα API το οποίο δίνει την δυνατότητα στους χρήστες να επεκτείνουν την λειτουργικότητα του και να προσθέσουν νέα extensions στο εργαλείο. Αυτό το API όμως, είναι και η μεγαλύτερη αδυναμία του αυτή τη στιγμή. Διότι ο σχεδιασμός και η υλοποίηση του API έγιναν με κύριο στόχο την παροχή της δυνατότητας στον προγραμματιστή να μπορεί να έχει πρόσβαση και να επεκτείνει το Swing UI του ZAP. Δεν έχει καμία υποδομή για την διευκόλυνση της υλοποίησης πολυνηματικών extensions, ούτε και κάποιο ενιαίο τρόπο σχετικά με το πώς επικοινωνούν αυτά τα extensions με το γραφικό περιβάλλον. Το κάθε extension έχει την δυνατότητα να προσθέσει νέα κουμπιά, επιλογές στο context menu και panels. Δηλαδή το API είναι φτιαγμένο με τέτοιο τρόπο ώστε να δεσμεύει τον προγραμματιστή να αναπτύξει τη νέα λειτουργικότητα σαν κομμάτι μίας desktop εφαρμογής και όχι ανεξάρτητα από την διεπαφή με την οποία θα επικοινωνήσει ο χρήστης. Αυτός ο σχεδιασμός έχει οδηγήσει το ZAP στο να είναι εγκλωβισμένο στο Swing UI τη στιγμή που για να χρησιμοποιηθεί από επαγγελματίες θα έπρεπε να έχει ένα πλήρες Web UI ώστε να μπορεί να εγκατασταθεί σε ένα server και να προσπελασθεί απομακρυσμένα και εύκολα από τους χρήστες. Όμως το ZAP δεν έχει αυτή τη στιγμή την δυνατότητα να αναγνωρίζει χρήστες. Οπότε, ο συνδυασμός ενός Web UI μαζί με την έλλειψη ελέγχου της πρόσβασης των χρηστών και η δυνατότητα πρόσβασης από το δημόσιο δίκτυο θα σήμαινε πρακτικά ότι ένας οποιοσδήποτε άνθρωπος θα μπορούσε να έχει πλήρη πρόσβαση στο εργαλείο.

Κάθε extension και κάθε νήμα μέσα στο ZAP διαθέτει την δυνατότητα να προσπελάσει οποιοδήποτε κόμβο μίας ιστοσελίδας. Στην τωρινή μορφή του εργαλείου

αυτό δεν είναι πρόβλημα, αλλά σε περίπτωση που παραπάνω από ένας χρήστες έχουν πρόσβαση και εκτελούν ο καθένας πολλές εργασίες ταυτόχρονα, τότε για λόγους ασφαλείας και ακεραιότητας των δεδομένων, θα πρέπει η κάθε εργασία να διαθέτει ένα ξεχωριστό γράφο της ιστοσελίδας πάνω στην οποία πραγματοποιεί το έργο της.

Τέλος, μία νέα λειτουργικότητα που θα μπορούσε να ξεχωρίσει το ZAP από τον ανταγωνισμό, θα ήταν η δυνατότητα να τρέξει το εργαλείο σαν ένα SaaS μέσα σε ένα cluster. Για να υλοποιηθεί θα έπρεπε να έχει ένα WebUI και την δυνατότητα υποστήριξης πολυνηματικής λειτουργίας σε κάθε δαίμονα ZAP που τρέχει μέσα στο cluster. Επίσης, πρέπει να υπάρχει ένας μηχανισμός ώστε οι πελάτες να μπορούν να διατάξουν εύκολα την εκτέλεση κάποιου έργου ανεξάρτητα από τον τρόπο με τον οποίο θα φτάσει η αίτηση στους εργάτες. Η ίδια η μεταφορά, η ανταλλαγή μηνυμάτων και αποτελεσμάτων και ο δίκαιος μοιρασμός του φορτίου μπορεί να αφεθεί σε ένα εξωτερικό εργαλείο. Υπάρχουν αρκετά ώριμα εργαλεία πλέον για αυτό το σκοπό, όπως το Celery που συνδέεται με ουρές μηνυμάτων όπως το RabbitMQ και το ZeroMQ, το Huey, το Retask ή και τα διάφορα task queues που προσφέρουν σαν υπηρεσία πλέον αρκετοί cloud providers όπως η Google Cloud Platform και η Azure.

1.7 Τι προσθέτει αυτή η εργασία

Σκοπός της εργασίας αυτής, είναι να προστεθεί ένας νέος πυρήνας εκτέλεσης εργασιών στο Zed Attack Proxy και να περιγραφούν αναλυτικά τα κίνητρα και οι στόχοι που έδωσαν το έναυσμα για την κατασκευή του. Επίσης, περιγράφεται αναλυτικά η αρχιτεκτονική του ώστε να γίνει κατανοητό πως εκπληρώνονται οι στόχοι της κατασκευής. Τέλος, θα μεταφερθούν και κάποια κομμάτια της τωρινής λειτουργικότητας του εργαλείου πάνω στον νέο πυρήνα, προκειμένου να αποτελέσουν απτή απόδειξη της καλής λειτουργίας του αλλά και ένα παράδειγμα, για το μέλλον, σχετικά με το πώς μπορεί να μεταφερθεί στο μέλλον ολόκληρο το σύνολο λειτουργιών του ZAP πάνω στον πυρήνα.

2. ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΤΟΥ API.

2.1 Απαιτήσεις που πρέπει να καλύπτει το νέο API.

Για να επιτευχθούν οι στόχοι και να καλυφθούν οι αδυναμίες που αναφέρθηκαν στο πρώτο κομμάτι της εργασίας η νέα λειτουργικότητα που θα προστεθεί θα πρέπει να ικανοποιεί τους εξής στόχους:

- πρέπει να διατηρηθεί η δυνατότητα ένας προγραμματιστής να μπορεί να δημιουργήσει εύκολα καινούργια extensions.
- πρέπει τα νέα extensions να εκτελούνται πάντοτε σαν νήματα.
- ο προγραμματιστής δεν πρέπει να χρειάζεται να ασχοληθεί με τον μηχανισμό με τον οποίο θα εκτελεστούν τα νήματα ενός extension.
- πρέπει να δημιουργηθεί ένας πυρήνας που να έχει τον κεντρικό έλεγχο και την διαχείριση των νημάτων.
- πρέπει να υπάρχει ένα ενιαίο API που να βοηθάει τα νήματα να επικοινωνήσουν με το περιβάλλον τους που να είναι πιο αφαιρετικό από το υπάρχον, ώστε να μπορεί να υποστηρίξει ταυτόχρονα πολλές διαφορετικές διεπαφές με τον χρήστη και να μην δεσμεύει το εργαλείο. Δεδομένου του τι προσφέρεται ήδη από το ZAP, μπορούμε να περιορίσουμε το τι εννοούμε επικοινωνία μεταξύ πελάτη και νήματος στα εξής: από την πλευρά του ο πελάτης πρέπει να μπορεί να έχει την δυνατότητα να στείλει σήματα στο ή στα νήματα προκειμένου να ελέγξει την κατάσταση τους, δηλαδή να παύσουν προσωρινά ή εντελώς την εργασίας τους. Επίσης, θα πρέπει να έχει την δυνατότητα να μπορεί να εξετάσει την κατάσταση τους και την εξέλιξη της εργασίας τους. Από την πλευρά τους τα νήματα θα πρέπει να μπορούν να κοινοποιούν μηνύματα προς οποιονδήποτε πελάτη Αυτά τα μηνύματα θα πρέπει να μπορούν να

προσπελασθούν από τον πελάτη σαν ένας απλός Iterator, προκειμένου να μπορούν μετά να χρησιμοποιηθούν από οποιαδήποτε διεπαφή. Τα μηνύματα αυτά δεν είναι απαραίτητο να είναι strings αλλά μπορεί να είναι αντικείμενα οποιασδήποτε κλάσης επιλέξει ο κατασκευαστής ενός extension. Προφανώς, όμως, δεδομένου ότι το ZAP είναι γραμμένο σε Java, όλα τα μηνύματα πρέπει να είναι του ίδιου τύπου ή κάποιου τύπου που να κληρονομεί από τον αρχικό τύπο που θα δηλωθεί.

- όπως φαίνεται και από τον παραπάνω στόχο, ο χρονοπρογραμματιστής πρέπει να έχει την δυνατότητα να χωρίσει μία εργασία σε όσο το δυνατόν περισσότερα νήματα μπορεί. Όμως, ο πελάτης θα πρέπει να μπορεί να επικοινωνήσει σαν να πρόκειται για ένα νήμα που εκτελεί την εργασία.
- πρέπει να είναι τέτοια η κατασκευή του πυρήνα ώστε να μπορεί να δεχθεί καινούργιες αιτήσεις για εργασίες από οπουδήποτε, είτε μέσα από την εφαρμογή είτε από μία διαφορετική εφαρμογή. Ο μηχανισμός που θα χρησιμοποιείται για τις νέες αιτήσεις πρέπει επίσης να προβλέπει κάποιον τρόπο για μελλοντική επέκταση ώστε να μπορεί να αναγνωριστεί από τον πυρήνα ποιος χρήστης δημιούργησε την αίτηση.
- η κατασκευή του API πρέπει να γίνει έχοντας υπόψη ότι στο κοντινό μέλλον έχει προβλεφθεί να προστεθεί η δυνατότητα στο ZAP να αναγνωρίζει χρήστες. Οπότε, πρέπει ειδικά ο πυρήνας να μπορεί εύκολα να αλλάξει ώστε να μπορεί να ελέγχει τις εργασίες και την πρόσβαση σε αυτές βάσει της ταυτότητας του χρήστη.
- όσο το δυνατόν περισσότερες δομές από αυτές που θα δημιουργηθούν να αποτελούνται από interfaces ώστε στο μέλλον να μπορούν να υλοποιηθούν με διαφορετικό τρόπο και να επεκταθούν. Αυτό έχει ιδιαίτερη σημασία διότι η κοινότητα των προγραμματιστών του ZAP έχει θέσει ως στόχο στο κοντινό μέλλον να ανακατασκευασθούν κάποια από τα βασικά κομμάτια του ZAP όπως το spidering και τα sessions ώστε να μειωθούν οι ανάγκες του εργαλείου σε μνήμη και να αποθηκεύονται τα αντικείμενα σε μία βάση δεδομένων.
- κάποια από τα υπάρχοντα extensions προσφέρουν και τα ίδια ένα API για την

επέκταση της λειτουργικότητας τους με την κατασκευή plugins, όπως το ActiveScan και το PassiveScan. Οι δομές που θα δημιουργηθούν δεν πρέπει να παρεμβαίνουν με κάποιο τρόπο στο API στο οποίο στηρίζονται τα υπάρχοντα plugins ή τουλάχιστον να μην αλλάζουν κάποιο κομμάτι με το οποίο έρχονται σε επαφή οι χρήστες.

- όλα τα κομμάτια του API θα πρέπει να είναι κατασκευασμένα με τέτοιο τρόπο ώστε να είναι ασφαλή για χρήση από πολλά νήματα ταυτόχρονα.

Εκτός όμως από την υλοποίηση ενός API που να ικανοποιεί τους παραπάνω στόχους, σημαντικό είναι να δημιουργηθεί και ένα test suite που θα αποδεικνύει στην πράξη ότι τα κινούμενα μέρη του API δουλεύουν σωστά. Τέλος, δεδομένου ότι ο πρώτος στόχος του API είναι να αναπροσαρμοστεί η ήδη υπάρχουσα υποδομή πάνω στις νέες δομές, κρίνεται αναγκαίο να ανακατασκευαστεί κάποιο extension. Με αυτό τον τρόπο θα υπάρξει μία πραγματική επίδειξη των δυνατοτήτων του νέου API, και ο κώδικας θα μπορούσε να χρησιμοποιηθεί σαν οδικός χάρτης για την ανακατασκευή και των υπολοίπων extensions.

2.2 Κεντρική διαχείριση και χρονοπρογραμματισμός των νημάτων με τον GeneralWorker.

Η καρδιά του συστήματος αποτελείται από τον χρονοπρογραμματιστή των threads, που υλοποιείται από την κλάση WorkScheduler. Η κλάση αυτή είναι ένα singleton, το οποίο αρχικοποιείται στο ξεκίνημα της εφαρμογής και το οποίο αναλαμβάνει να εκτελέσει όλες τις αιτήσεις για εργασία.

Για να εκτελεστεί οποιαδήποτε εργασία από το ZAP, πρέπει να δοθεί αντίστοιχη εντολή στο συγκεκριμένο αντικείμενο με την κλήση της μεθόδου submitWork(). Όταν θα υπάρχει η δυνατότητα να ταυτοποιηθούν οι χρήστες και να ελεγχθούν τα δικαιώματά τους, τότε θα υπάρχει η δυνατότητα να επεκταθεί ο WorkScheduler, ώστε σε κάθε κλήση της submitWork() να εξασφαλίζει ότι ο χρήστης έχει τα αναγκαία διαπιστευτήρια για να ζητήσει την εκτέλεση μίας εργασίας. Προς το παρόν όμως, ο WorkScheduler θα εκτελέσει οποιαδήποτε εργασία του ζητηθεί.

Όλα τα threads που δημιουργεί ο χρονοπρογραμματιστής για να εκτελέσουν μία

εργασία, θα εκτελεσθούν στο ίδιο thread pool. Στην τωρινή έκδοση του κώδικα το μέγεθος του thread pool είναι σταθερό στα 20 threads αλλά στο μέλλον το μέγεθος του θα μπορούσε επίσης να φορτώνεται εύκολα από ένα configuration αρχείο. Ο WorkScheduler δεν κάνει καμία υπόθεση σχετικά με τον τρόπο με τον οποίο εκτελείται μία εργασία ή τι χρειάζεται για να εκτελεστεί. Όταν εμφανιστεί μία αίτηση για την εκτέλεση μίας εργασίας, τότε εφόσον υπάρχουν διαθέσιμα threads μέσα στο thread pool, τότε εκτελεί την εργασία. Αλλιώς τοποθετεί τα threads μέσα σε μία ουρά μέχρις ότου εμφανιστεί ένα διαθέσιμο thread. Όταν εμφανιστεί ένα διαθέσιμο thread ο χρονοπρογραμματιστής χρησιμοποιεί ένα αντικείμενο του τύπου MinMaxPriorityQueue από την βιβλιοθήκη Guava, το οποίο με την σειρά του, χρησιμοποιεί ένα αντικείμενο τύπου Comparator προκειμένου να αποφασίσει ποια είναι η εργασία με την μέγιστη προτεραιότητα. Αυτή η επιλογή δίνει την δυνατότητα να αλλάζει ο Comparator που χρησιμοποιείται σύμφωνα με τις επιθυμίες του χρήστη δίνοντας προτεραιότητα σε άλλες παραμέτρους όπως τον τύπο της εργασίας, το ποιος χρήστης έδωσε την εντολή ή με οτιδήποτε άλλο κριθεί αναγκαίο στο μέλλον, προς το παρόν η βασική παράμετρος για να αποφασιστεί η προτεραιότητα μίας εργασίας είναι ο χρόνος κατά τον οποίο έγινε η αίτηση. Το ποιος Comparator χρησιμοποιείται θα μπορούσε εύκολα να γίνει κομμάτι του configuration και να αρχικοποιείται με ένα dependency injection framework.

Η ουρά προτεραιότητας αποθηκεύει αντικείμενα του τύπου WorkUnit τα οποία περιέχουν όλα τα αντικείμενα είναι αναγκαία για να εκτελέσει ο WorkScheduler μία εργασία. Κάθε ένα από τα αντικείμενα WorkUnit κρατάει μία αναφορά σε ένα αντικείμενο του τύπου WorkCommand, ένα αντικείμενο που υλοποιεί το interface MessageRepository, ένα αντικείμενο που υλοποιεί το interface WorkerResultIterable τα οποία χρησιμοποιούνται για την επικοινωνία των νημάτων εκτός του συστήματος με τα νήματα που εκτελούν την εκάστοτε εργασία. Επίσης, αποθηκεύεται μία λίστα με αντικείμενα του τύπου GeneralWorker τα οποία εκτελούν ή θα εκτελέσουν την εργασία και ένα αντικείμενο που υλοποιεί το interface IThreadState και χρησιμοποιείται για τον συγχρονισμό όλων των νημάτων και η ακριβής ώρα κατά την οποία κατατέθηκε το αντικείμενο.

Για να παρακολουθεί τον αριθμό των διαθέσιμων threads, ο WorkScheduler όταν αρχικοποιεί τα αντικείμενα για μία καινούργια εργασία, προσθέτει στο masterThreadState αντικείμενο, δηλαδή το αντικείμενο που έχει την γενική επίβλεψη της κατάστασης μίας εργασίας μέσα στην ιεραρχία των IThreadState αντικειμένων, κάποια αντικείμενα του τύπου AsyncCallback που θα ενεργοποιηθούν όταν τελειώσει η εργασία και θα φροντίσουν να ανανεώσουν τους μετρητές που κρατάει ο WorkScheduler για τον αριθμό των ενεργών threads και θα ξεκινήσει την επόμενη εργασία που βρίσκεται στην ουρά. Με αυτό τον τρόπο απλοποιείται πολύ ο σχεδιασμός του και απεμπλέκεται εντελώς η εκτέλεση εργασιών από το interface το οποίο χρησιμοποιήθηκε για να δοθεί η εντολή και

απλοποιείται ο τρόπος με τον οποίο διαχειρίζεται ο WorkScheduler τις εργασίες αφού μετά την αρχικοποίηση των αντικειμένων σχετικά με την εκτέλεση της εργασίας, το είδος της δεν παίρνεται ξανά υπόψη (active ή passive scans, spidering, zest script). Προς το παρόν, ο WorkScheduler όταν εκτελεί μία καινούργια εργασία την εκτελεί σε ένα μοναδικό νήμα χρησιμοποιώντας ένα αντικείμενο SingleThreadState, του οποίου η λειτουργία θα εξηγηθεί περαιτέρω σε επόμενη παράγραφο εκτός από τις εργασίες του τύπου ActiveScan οπότε και χρησιμοποιούνται 3 threads και αντίστοιχα 3 SingleThreadStates και ένα MultiThreadState σαν masterThreadState. Στο μέλλον υπάρχει η δυνατότητα να υλοποιηθούν εύκολα πιο πολύπλοκοι αλγόριθμοι οι οποίοι θα μπορούν να διαχωρίζουν το φορτίο μίας εργασίας σε πολλά νήματα, θα βασίζονται σε αντικείμενα του τύπου MultiThreadState για τον συγχρονισμό και θα αυξομειώνουν δυναμικά τον αριθμό των GeneralWorkers που ασχολούνται με την περάτωση της εργασίας.

Η μέθοδος submitWork(), αφού δημιουργήσει όλα τα απαραίτητα αντικείμενα και αποφασίσει αν θα εκτελέσει άμεσα ή όχι, επιστρέφει το αντικείμενο του τύπου WorkerResultIterable, το οποίο θα περιγραφεί σε επόμενη παράγραφο αναλυτικά.

Ο σχεδιασμός που περιγράφηκε αποτελεί το πρώτο βήμα για την μετατροπή του ZAP από ένα απλό desktop application σε ένα daemon που θα μπορεί στο μέλλον να τρέχει και να μπορεί να προσπελαστεί ταυτόχρονα από πολλούς χρήστες και να εκτελέσει διαφορετικές εργασίες ταυτόχρονα και ασύγχρονα.

Αυτό δεν είναι άμεσα δυνατό γιατί το ZAP δεν διαθέτει ακόμα την δυνατότητα να επικοινωνήσει με μία βάση δεδομένων, αλλά θα υλοποιηθεί και θα συμπεριληφθεί στην έκδοση 3.0 που θα γίνει διαθέσιμη μέσα στο 2016. Σε περίπτωση που κάποιο άλλο νήμα επιχειρήσει να επικοινωνήσει με κάποιο από τα νήματα που εκτελούν εργασία.

2.3 Εκτέλεση ενός νήματος GeneralWorker.

Η βάση για την εκτέλεση οποιασδήποτε εργασίας από τον χρονοπρογραμματιστή είναι η κλάση GeneralWorker. Η κλάση αυτή είναι υποκλάση της Runnable και σκοπός της είναι να αποτελέσει την βάση πάνω στην οποία θα χτιστεί όλη η υπόλοιπη λειτουργικότητα. Διαθέτει συναρτήσεις οι οποίες επιτρέπουν σε άλλα αντικείμενα να λάβουν πληροφορίες σχετικά με την κατάσταση της εκτέλεσης της εργασίας του νήματος.

Για να υλοποιηθεί μία καινούργια λειτουργία, ο προγραμματιστής χρειάζεται να δημιουργήσει μία υποκλάση της κλάσης GeneralWorker και να κάνει override την

συνάρτηση `executeWork()`. Όταν ένα νήμα δοθεί στην `executor service` από τον `WorkScheduler` τότε καλείται η μέθοδος `run()` του νήματος που με την σειρά της καλεί την `executeWork()`. Προτού κληθεί η μέθοδος `executeWork()`, ο `GeneralWorker` θα φροντίσει να θέσει όλες τις σωστές τιμές και να αρχικοποιήσει τα κατάλληλα αντικείμενα ώστε όταν ξεκινήσει η `executeWork()` να μην χρειάζεται να ασχοληθεί με τίποτα άλλο εκτός από την εργασία της. Καθώς εκτελεί το έργο του το νήμα, ο προγραμματιστής πρέπει να προβλέψει ανά τακτά χρονικά διαστήματα να χρησιμοποιεί τις μεθόδους `shouldStop()` και `shouldPause()`. Η μέθοδος `shouldStop()` επιστρέφει ένα `boolean` που σηματοδοτεί αν έχει ληφθεί κάποιο σήμα που να ειδοποιεί το νήμα ότι πρέπει να σταματήσει την εργασία του. Η μέθοδος `shouldPause()` φροντίζει και αυτή να εξετάσει αν έχει ληφθεί κάποιο σήμα που να ενημερώνει ότι το νήμα πρέπει να παύσει προσωρινά και αν αυτό έχει συμβεί τότε φροντίζει αυτόματα να σταματήσει την εκτέλεσή του. Εδώ, πρέπει να τονιστεί ότι το περιβάλλον απλώς στέλνει σήματα σχετικά με το ποια είναι η επιθυμητή κατάσταση του νήματος. Η ίδια η κατάσταση του νήματος μπορεί να αλλάξει μόνο αν το ίδιο το νήμα εθελοντικά καλέσει κάποια από τις δύο συναρτήσεις που αναφέρθηκαν πριν. Και στην περίπτωση της `shouldStop()` η συνάρτηση δεν πρόκειται να τερματίσει την εκτέλεση του νήματος, αλλά απλά θα ενημερώσει το νήμα, ώστε να δοθεί πρώτα ο χρόνος να τερματίσει και να καθαρίσει σωστά όποιες δομές χρησιμοποιεί και μετά να τερματίσει. Τέλος, υπάρχει και η συνάρτηση `cancel()`, σε περίπτωση που υπάρξει κάποιο ανυπέρβλητο εμπόδιο για την εκτέλεση του νήματος, τότε μπορεί να χρησιμοποιήσει αυτή την συνάρτηση για να καταχωρίσει όλα τα `exceptions`, που εμφανίστηκαν ώστε να υπάρχει η δυνατότητα για `debugging` από τον χρήστη μετά το τέλος της εργασίας του νήματος.

Σκοπός των συναρτήσεων του `GeneralWorker`, είναι να κάνουν όλες τις απαραίτητες ενέργειες όπως το να θέσουν τις σωστές τιμές για την κατάσταση και να εκτελέσουν όλα τα `callbacks` που έχουν καταχωρηθεί για τις εκάστοτε αλλαγές και να μην επιβαρύνεται ο προγραμματιστής με την υλοποίηση αυτών των ενεργειών. Επίσης, η κλάση `GeneralWorker` είναι κατασκευασμένη με τέτοιο τρόπο ώστε να μην κρατάει η ίδια καμία από τις τιμές που καταχωρούν την κατάσταση του νήματος, ούτε αναλαμβάνει να κρατήσει και να συγχρονίσει τα σήματα που στέλνονται στο νήμα, ούτε να κρατήσει και να εκτελέσει τα `callbacks` και ούτε να αναλάβει με κάποιο τρόπο να αποθηκεύσει με κάποιο τρόπο τα μηνύματα που επιθυμεί να κοινοποιήσει το νήμα στο περιβάλλον. Αυτές όλες οι ευθύνες ανατίθενται σε άλλα αντικείμενα που ενεργούν εκτός του νήματος και θα περιγραφούν στην συνέχεια. Αυτό δίνει το πλεονέκτημα ότι μπορούν εύκολα να επεκταθούν οι δυνατότητες του `GeneralWorker` χωρίς να χρειάζεται να αλλάξει ο κώδικας των υποκλάσεων του `GeneralWorker`.

Σκοπός της υλοποίησης της κλάσης είναι να δίνει μία βάση στον προγραμματιστή για να δημιουργήσει νέες λειτουργίες για το ZAP με όσο το δυνατόν λιγότερες προσθήκες

στην κλάση Thread. Με αυτό τον τρόπο μπορεί εύκολα να εξελίξει τον κώδικα σαν να έγραφε ένα απλό java thread, έχοντας άμεση πρόσβαση στις βασικές δομές που θα του επιτρέψουν να επικοινωνήσει τα αποτελέσματα και την κατάσταση του. Μία σημαντική παράμετρος που πρέπει να παρθεί υπόψη από τους προγραμματιστές είναι ότι ένα νήμα GeneralWorker δεν έχει κανένα έλεγχο σχετικά με το αν θα αναλάβει όλο το φορτίο της εργασίας ή αν θα διαχωριστεί το φορτίο σε πολλά νήματα. Αυτό πρέπει να είναι επιλογή του WorkScheduler για να μπορεί να είναι όσο το δυνατόν πιο ευέλικτος. Επίσης, δεν υπάρχει κάποιος τρόπος για να ανακαλύψει το νήμα πόσα άλλα νήματα εκτελούνται για την ίδια εργασία.

2.4 Συγχρονισμός της λειτουργίας των νημάτων GeneralWorker.

Τα αντικείμενα που υλοποιούν το interface IthreadState χρησιμοποιούνται προκειμένου να μπορούν νήματα εκτός του πυρήνα του ZAP να παρακολουθήσουν την εξέλιξη της λειτουργίας των νημάτων εντός του πυρήνα. Το interface και τα αντικείμενα που το υλοποιούν, είναι σχεδιασμένα με τέτοιο τρόπο ώστε να επιτρέπουν την ασφαλή χρήση τους από πολλαπλά thread ταυτόχρονα προκειμένου να μπορεί να εκτιμηθεί με ακρίβεια, είτε η κατάσταση μίας εργασίας είτε η κατάσταση κάποιου από τα νήματα που την εκτελούν. Αυτή τη στιγμή υπάρχουν δύο αντικείμενα που υλοποιούν το interface IThreadState: η κλάση SingleThreadState και η κλάση MultiThreadState.

Τα αντικείμενα της κλάσης SingleThreadState έχουν σαν στόχο να συγχρονίσουν την επικοινωνία ενός νήματος τύπου GeneralWorker με το περιβάλλον του. Για την συγχρονισμένη αλλαγή των καταστάσεων και των σημάτων του SingleThreadState χρησιμοποιούνται αντικείμενα της κλάσης MonadicAtomicFieldReferenceUpdater. Η κλάση αυτή είναι ένας wrapper γύρω από την κλάση AtomicReferenceFieldUpdater και φροντίζει να ανακατευθύνει αυτόματα όλες τις κλήσεις προς την με ένα αντικείμενο SingleThreadState και έχει σαν σκοπό να απλοποιήσει την δομή και να αυξήσει την αναγνωσιμότητα του κώδικα και να προσφέρει την δυνατότητα ακριβούς logging κατά την κλήση οποιασδήποτε συνάρτησης αλλάζει την κατάσταση του αντικειμένου SingleThreadState. Η κλάση AtomicReferenceFieldUpdater παρέχεται από το JDK και αναλαμβάνει να αλλάξει την κατάσταση ενός πεδίου κάποιου αντικειμένου και παρέχει κάποιες πολύ χρήσιμες συναρτήσεις για τον έλεγχο και την ανανέωση της κατάστασης όπως η getAndSet και η compareAndSet. Σε περιπτώσεις όπου πολλά αντικείμενα προσπαθούν ταυτόχρονα να στείλουν κάποιο σήμα σε ένα αντικείμενο SingleThreadState,

οι συναρτήσεις του AtomicFieldReferenceUpdater μπορούν να βοηθήσουν στην σωστή ανάθεση τιμών στα σήματα χωρίς να είναι αναγκαίες δομές συγχρονισμού όπως σημαφόροι και synchronized blocks που θα είχαν σημαντικό αντίκτυπο στην απόδοση του συστήματος. Κάθε αντικείμενο SingleThreadState διαθέτει δύο booleans για τα σήματα του pause και cancel και τρία Objects, τα startOfWorkLock, pauseLock και endOfWorkLock. Οι booleans χρησιμοποιούνται από την κλάση GeneralWorker προκειμένου να καταλάβει αν πρέπει να παύσει την εκτέλεση της εργασίας του, ενώ τα τρία αντικείμενα χρησιμοποιούνται προκειμένου να συγχρονίσουν την εκτέλεση των callbacks που καταχωρούνται σε ένα αντικείμενο.

Πιο συγκεκριμένα, το interface IThreadState ορίζει τις υπογραφές τριών συναρτήσεων οι οποίες μπορούν να χρησιμοποιηθούν από ένα νήμα για να εκτελέσει οποιαδήποτε ενέργεια επιθυμεί όταν συμβεί κάποιο γεγονός. Η συνάρτηση registerStartOfWorkCallback() αναλαμβάνει να καταχωρίσει ένα callback και να το εκτελέσει όταν ξεκινήσει η εργασία του GeneralWorker και θα επιστρέψει true. Σε περίπτωση που ο GeneralWorker έχει ήδη ξεκινήσει την εργασία του, τότε το callback δεν θα καταχωρηθεί και θα επιστραφεί false από την συνάρτηση. Η συνάρτηση registerPauseCallback() καταχωρεί ένα callback για να εκτελεστεί όταν ξαναρχίσει η εκτέλεση της εργασίας. Αν το νήμα δεν έχει ακόμα παύσει την εκτέλεσή του τότε θα καταχωρηθεί το callback και θα εκτελεστεί είτε αφότου παύσει και ξαναρχίσει την εκτέλεση της εργασίας το νήμα, είτε όταν το αντικείμενο SingleThreadState τεθεί σε μία από τις καταστάσεις οι οποίες σηματοδοτούν το τέλος της εργασίας του νήματος. Η συνάρτηση registerEndOfWorkCallback() καταχωρεί ένα callback και θα το εκτελέσει όταν τερματιστεί η εκτέλεση της εργασίας. Όλα τα callbacks είναι αντικείμενα του τύπου interface AsyncCallback. Το interface ορίζει μόνο μία συνάρτηση, την callback() μέσα στην οποία βρίσκεται όλος ο κώδικας που πρέπει να εκτελεσθεί. Ο λόγος που το AsyncCallback ορίζει μόνο μία συνάρτηση, είναι ώστε όταν μετακινηθεί το ZAP στην έκδοση 8 της Java τότε θα μπορούν να αντικατασταθούν τα AsyncCallbackds με lambdas. Το αντικείμενο SingleThreadState χρησιμοποιεί την κλάση AsyncCallbackWrapper προκειμένου να καταχωρίσει τα AsyncCallbacks και να προσθέσει μία μοναδική ταυτότητα στο αντικείμενο. Αυτή η ταυτότητα χρησιμοποιείται ειδικά στην περίπτωση των callbacks για να ξεχωρίσει το SingleThreadState ποια callbacks πρέπει να εκτελέσει. Ένα από τα πεδία του αντικείμενου είναι το pauseld που είναι τύπου AtomicInteger. Κάθε φορά που ένα νέο αντικείμενο AsyncCallback δίνεται στο SingleThreadState τότε δημιουργείται ένα νέο αντικείμενο τύπου AsyncCallbackWrapper και του δίνεται πρώτα το AsyncCallback αντικείμενο καθώς και την τωρινή τιμή του pauseld. Όταν δοθεί σήμα στο SingleThreadState να ξαναρχίσει την λειτουργία του τότε, όλα τα callbacks που έχουν ταυτότητα ίση ή μικρότερη από την τωρινή τιμή του pauseld θα εκτελεστούν και θα αυξηθεί

η τιμή του `paused`. Με αυτό τον τρόπο εξασφαλίζεται ότι θα εκτελεστούν όλα τα callbacks που έχουν καταχωρηθεί μέχρι και την στιγμή που άλλαξε η κατάσταση του νήματος από `PAUSED` σε `RUNNING`. Στο τέλος, κάθε callback που εκτελείται αφαιρείται από την λίστα των callbacks. Για να εξασφαλιστεί η συγχρονισμένη πρόσβαση στις λίστες των callbacks, όλες οι λίστες είναι του τύπου `SynchronizedList` που επιστρέφεται από την μέθοδο `synchronizedList` του αντικειμένου `Collections`. Αν η κατάσταση αλλάξει σε κάποια που να σηματοδοτεί το τέλος της εκτέλεσης του νήματος, τότε όλα τα callbacks θα εκτελεστούν ανεξαιρέτως.

Για κάθε σήμα που έρχεται και κάθε αλλαγή της κατάστασης της εκτέλεσης του νήματος, το αντικείμενο `SingleThreadState` κρατάει μία καταγραφή του συμβάντος χρησιμοποιώντας ένα αντικείμενο της κλάσης `EventRecord`. Με αυτό τον τρόπο μπορεί να γίνει εύκολα debugging και να εξαχθούν στατιστικά σχετικά με την λειτουργία του νήματος. Τα αντικείμενα του τύπου `EventRecord` αποθηκεύονται σε μία λίστα του τύπου `ConcurrentLinkedDeque` και η αποθήκευση γίνεται μέσα στον κώδικα που αλλάζει τις τιμές των αντίστοιχων μεταβλητών και πάντα μέσα σε `synchronized blocks` ώστε να εξασφαλιστεί ότι η σειρά των καταγραφών αντιστοιχεί στην χρονική σειρά των γεγονότων.

Η κατάσταση ενός νήματος μπορεί να αλλάξει μόνο από κλήσεις συναρτήσεων της κλάσης `GeneralWorker` και μπορεί να έχει τις εξής τιμές: `NOT_RUNNING`, `RUNNING`, `PAUSED`, `FINISHED`, `ABORTED` και `CANCELLED`. Ο λόγος για την ύπαρξη τριών διαφορετικών καταστάσεων που σηματοδοτούν το τέλος της εκτέλεσης της εργασίας έχει σαν σκοπό να κάνει πιο σαφή τον λόγο που η εργασία τελείωσε. Σε περίπτωση που η κατάσταση έχει την τιμή `FINISHED` τότε η λειτουργία του νήματος προχώρησε χωρίς πρόβλημα.

Η αλλαγή των καταστάσεων γίνεται βάσει των σημάτων που δέχεται το αντικείμενο `SingleThreadState`. Δεν υλοποιείται κάποιος μηχανισμός με τον οποίο γίνεται έλεγχος στο αν το αντικείμενο που έστειλε το σήμα έχει αυτό το δικαίωμα και δεν έχει υπάρξει πρόβλεψη για κάποιον τέτοιο μηχανισμό στο μέλλον. Το αντικείμενο `SingleThreadState` δεν είναι κατασκευασμένο για να έρχεται σε άμεση επαφή με κλάσεις που ανήκουν στον πελάτη αλλά μόνο από κώδικα που του ίδιου του πυρήνα. Και δεδομένου ότι όλα αυτά τα αντικείμενα δίνονται κατόπιν αιτήσεων στον `WorkScheduler`, θεωρείται αρμοδιότητα του `WorkScheduler` να εξασφαλίσει ότι το μόνο αντικείμενα με κατάλληλα δικαιώματα μπορούν να στείλουν σήμα στο `SingleThreadState`. Για αυτό το `SingleThreadState` δεν δίνει περισσότερη ή λιγότερη βαρύτητα στα σήματα που λαμβάνει, αλλά επεξεργάζονται με χρονική σειρά προτεραιότητας και για αυτό η κατάσταση του `SingleThreadState` ανά πάσα στιγμή καθορίζεται από το τελευταίο σήμα που έλαβε.

Εκτός από την κλάση `SingleThreadState`, το interface `IThreadState` υλοποιεί και η

κλάση `MultiThreadState`. Η συγκεκριμένη κλάση έχει σαν σκοπό να προσφέρει περισσότερη ευελιξία στον `WorkScheduler` ώστε να μπορεί δυναμικά να διαχωρίσει το φορτίο σε πολλά νήματα. Η κλάση αυτή χρησιμοποιεί μία λίστα μέσα στην οποία αποθηκεύει άλλα αντικείμενα που υλοποιούν το interface `IThreadState` και τα οποία θα αναφέρονται ως εσωτερικά `IThreadStates` στο εξής. Αυτά τα αντικείμενα μπορούν να προστεθούν ανά πάσα στιγμή. Κάθε κλήση σε μία συνάρτηση του `MultiThreadState` προσπελαύνει όλα τα αντικείμενα της λίστας με τα `IThreadStates` και εκτελεί την ίδια συνάρτηση. Στο τέλος, μαζεύει όλα τα αποτελέσματα και προσπαθεί να εξάγει ένα αποτέλεσμα που να δίνει μία εικόνα της γενικής κατάστασης της εκτέλεσης όλων των νημάτων.

Όταν δεν έχει δοθεί κανένα άλλο `IThreadState` αντικείμενο στο `MultiThreadState` και ζητηθεί η κατάσταση του αντικειμένου τότε επιστρέφει την κατάσταση `NOT_RUNNING`. Όταν δοθούν κάποια αντικείμενα `IThreadState`, τότε η κατάσταση του `MultiThreadState` καθορίζεται με τον εξής αλγόριθμο:

- Αν όλα τα αντικείμενα έχουν την ίδια κατάσταση τότε επιστρέφεται αυτή.
- Αν όλα τα αντικείμενα έχουν τερματίσει αλλά ένα ή παραπάνω έχουν την κατάσταση `CANCELLED`, τότε η κατάσταση του αντικειμένου θα είναι `CANCELLED` προκειμένου να ειδοποιήσει τον χρήστη ότι υπήρξε κάποιο πρόβλημα κατά την διάρκεια της εκτέλεσης της εργασίας.
- Αν έστω και ένα αντικείμενο έχει την κατάσταση `RUNNING` τότε η κατάσταση είναι `RUNNING`.
- Αν όλα τα αντικείμενα έχουν σταματήσει να εκτελούν κάποια εργασία αλλά υπάρχει έστω και ένα που να είναι `PAUSED` τότε η κατάσταση του αντικειμένου είναι `PAUSED`.

Για τις συναρτήσεις `isCancelSignalSet()` και `isPauseSignalSet()` το αντικείμενο θα καλέσει τις αντίστοιχες συναρτήσεις σε όλα τα εσωτερικά `IThreadState` αντικείμενα που έχουν κατάσταση `RUNNING` και θα επιστρέψει το αποτέλεσμα του λογικού `and` στα αποτελέσματα που επιστράφηκε από τα αντικείμενα. Αντίστοιχα, για να θέσει τα `pause` και `cancel` σήματα, καλεί τις συναρτήσεις `tryPause()`, `tryCancel()` και `tryContinue` σε κάθε εσωτερικό `IThreadState`, όπως και η κλήση της μεθόδου `getThrowables()` που συλλέγει όλα τα αντικείμενα που επιστρέφουν οι συναρτήσεις και τα μαζεύει σε ένα `Collection` και τα επιστρέφει. Οι συναρτήσεις `registerStartOfWorkCallback()` και `registerEndOfWorkCallback()` ελέγχουν πρώτα την κατάσταση των εσωτερικών `IThreadStates` και αν έχουν την σωστή κατάσταση τότε προσθέτουν τα `callbacks` στην αντίστοιχη ουρά και καταχωρούν ένα άλλο `callback` στα εσωτερικά `IThreadStates` τα οποία

θα ενεργοποιηθούν όταν αλλάξει αντίστοιχα η κατάσταση του callback και με την σειρά τους θα εκτελέσουν όλα τα callbacks που έχουν καταχωρηθεί στο αντικείμενο `MultiThreadState` που βρίσκεται ψηλότερα από όλα στην ιεραρχία. Εκτός από τις ουρές χρησιμοποιούνται και δύο αντικείμενα του τύπου `AtomicBoolean` για να βοηθήσουν στο σωστό συγχρονισμό της εκτέλεσης των callbacks. Στην σπάνια περίπτωση που ανάμεσα στον έλεγχο των καταστάσεων και την καταχώριση των callbacks στο εσωτερικά `IThreadStates`, έχει ενεργοποιηθεί το γεγονός το οποίο αναμένει το αντικείμενο `MultiThreadState` να εκτελεστεί τότε χρησιμοποιούνται αυτοί οι δύο booleans για να ειδοποιήσουν το αντικείμενο προκειμένου να εκτελέσει άμεσα το callback. Για την περίπτωση του `registerPauseCallbacks` χρησιμοποιείται αντί για έναν `AtomicBoolean`, ένας `AtomicInteger` η τιμή του οποίου κατά την καταχώριση ενός callback αποθηκεύεται και μέσα στο `AsyncCallbackWrapper` αντικείμενο που δίνεται στα χαμηλότερα `IThreadStates` και όταν ενεργοποιηθεί το callback θα εκτελεστούν όλα τα callbacks με ταυτότητα ίση ή μικρότερη του id του αντικειμένου.

Όπως γίνεται αντιληπτό, η χρησιμότητα της κλάσης `MultiThreadState` είναι ότι μπορεί να χρησιμοποιηθεί από τον `WorkScheduler` ώστε να δημιουργήσει δυναμικά μία δενδρική δομή από `MultiThreadStates` η οποία στα φύλλα έχει αντικείμενα `SingleThreadState`. Αυτό δίνει την ελευθερία στο μέλλον να δημιουργηθούν αρκετά πολύπλοκοι αλγόριθμοι χρονοπρογραμματισμού, χωρίς να χρειαστεί καμία αλλαγή στο `MultiThreadState`.

2.5 Μηχανισμός αίτησης ενός νήματος για χρονοπρογραμματισμό και εκτέλεση μίας εργασίας.

Προκειμένου να δώσει εντολή ένα νήμα για την έναρξη μίας εργασίας από τον χρονοπρογραμματιστή χρησιμοποιείται ένας συνδυασμός των `Builder` και `Command` και `Factory patterns`.

Όταν ένα νήμα επιθυμεί να ξεκινήσει κάποια εργασία τότε πρέπει να στείλει στον `WorkScheduler` ένα αντικείμενο που να είναι υποκλάση της `WorkCommand`. Για κάθε διαφορετική εργασία που μπορεί να εκτελέσει ο πυρήνας, υπάρχει και μία αντίστοιχη υποκλάση της `WorkCommand`. Σκοπός της κλάσης αυτής είναι να είναι εύκολο για τον `WorkScheduler` να πάρει μία αίτηση για μία εργασία χωρίς να χρειάζεται εξειδικευμένο κώδικα για να προγραμματίσει και να εκτελέσει το συγκεκριμένο είδος εργασίας και να μπορεί να υπάρχει αρκετή ευελιξία στο μέλλον ώστε να μπορούν να προστεθούν νέες

δυνατότητες με τον ελάχιστο δυνατό κώδικα. Επίσης, τα αντικείμενα του τύπου `WorkCommand`, είναι φτιαγμένα έτσι ώστε να είναι `immutable` μετά την δημιουργία τους. Αυτό σημαίνει ότι αφότου αρχικοποιηθεί ένα τέτοιο αντικείμενο, οι τιμές των πεδίων του δεν μπορούν να αλλάξουν. Με αυτό τον τρόπο ο `WorkScheduler` μπορεί να είναι σίγουρος ότι δε θα αλλάξει κάτι αφότου κατατεθεί ένα αντικείμενο `WorkCommand` και μπορεί να προγραμματίσει την εκτέλεση του με ακρίβεια.

Όπως αναφέρθηκε πριν, τα αντικείμενα που κληρονομούν από την κλάση `WorkCommand` και χρησιμοποιούνται για να κατατεθούν αιτήσεις για νέες εργασίες είναι `immutable`. Για να δημιουργηθούν αυτά τα αντικείμενα υπάρχουν δύο τρόποι: να δίνονται όλες οι τιμές των πεδίων στον `constructor` της κλάσης είτε να χρησιμοποιηθεί το `Builder pattern`. Η πρώτη μέθοδος έχει τα μειονεκτήματα ότι κάνει τον `constructor` της κλάσης πάρα πολύ μεγάλο και περίπλοκο και αντίστοιχα κάνει περίπλοκο και τον κώδικα που αρχικοποιεί το αντικείμενο. Το `Builder pattern` βοηθάει σε αυτές τις περιπτώσεις με το να αποσυνδέει την ανάθεση και τιμών από την ίδια την κατασκευή του αντικειμένου. Πιο συγκεκριμένα, πρώτα χρησιμοποιείται το `singleton CommandBuilderFactory`, προκειμένου να δημιουργήσει και να επιστρέψει ένα αντικείμενο κάποιου τύπου που είναι υποκλάση της `CommandBuilder`. Προς το παρόν το μόνο που κάνει η `CommandBuilderFactory` είναι να δημιουργεί τα αντικείμενα. Ο σκοπός της είναι ότι αν στο μέλλον κάποια από αυτά τα αντικείμενα χρειάζονται κάποια ιδιαίτερη διαδικασία για την δημιουργία τους, τότε μπορεί ο αναγκαίος κώδικας να ενσωματωθεί στην αντίστοιχη μέθοδο. Κατόπιν, τα αναγκαία ορίσματα δίνονται στο αντικείμενο και καλώντας την συνάρτηση `build()` δημιουργείται και επιστρέφεται ένα νέο αντικείμενο μίας υποκλάσης της `WorkCommand`. Αυτός ο σχεδιασμός διευκολύνει ιδιαίτερα την περίπτωση όπου κάποιο νήμα θέλει να δημιουργήσει αιτήσεις για πολλές εργασίες οι οποίες διαφέρουν λίγο στο ορίσματα τους, διότι μπορεί κάθε φορά να καλεί την μέθοδο `build()` και στην συνέχεια να αλλάζει τα αναγκαία ορίσματα και να την ξανακαλεί. Επίσης, επειδή η κλάση και οι υποκλάσεις της `WorkCommand` είναι σχεδιασμένη σαν `Monad`, τα ορίσματα μπορούν να της ανατεθούν με `chained method calls`, κάτι που κάνει τον κώδικα πιο σαφή και ευανάγνωστο προσθέτοντας όμως μία μικρή επιβάρυνση στην δημιουργία της ίδιας της κλάσης `WorkCommand`.

Κατά την δημιουργία του, σε κάθε αντικείμενο της κλάσης `WorkCommand` ανατίθεται και μία μοναδική ταυτότητα. Αυτή η ταυτότητα μπορεί να χρησιμοποιηθεί στην συνέχεια από ένα νήμα για να ζητήσει να δει την κατάσταση του αντικειμένου, αφότου αυτό έχει κατατεθεί προς εκτέλεση. Σε επόμενη έκδοση του εργαλείου, αυτές οι ταυτότητες μπορούν να χρησιμοποιούνται για να παρακολουθεί ο πυρήνας ποιες και πόσες εργασίες έχει διατάξει ο κάθε χρήστης.

2.6 Επικοινωνία των αποτελεσμάτων και της κατάστασης ενός νήματος GeneralWorker με το περιβάλλον.

Ένα σημαντικό κομμάτι μίας οποιασδήποτε πολυνηματικής εφαρμογής είναι ο μηχανισμός που επιτρέπει σε ένα νήμα να επικοινωνήσει πληροφορίες σχετικά με την κατάσταση του και τα αποτελέσματα του στο περιβάλλον. Στην περίπτωση του ZAP θα πρέπει να είναι δυνατόν μέσω ενός μηχανισμού ο πυρήνας αλλά και ο χρήστης να μπορεί ανά πάσα στιγμή να δει την κατάσταση της εκτέλεσης μίας εργασίας καθώς και τα αποτελέσματα της εκτέλεσης της εργασίας, αν αυτά έχουν παραχθεί. Επίσης, δεδομένου ότι κάποιες από τις εργασίες αναμένονται να πάρουν αρκετό χρόνο, χρειάζεται να υπάρχει ένας δείκτης που να δείχνει το ποσοστό ολοκλήρωσης της εργασίας. Εκτός από το ποσοστό ολοκλήρωσης, πρέπει ο χρήστης να μπορεί να δει το μερικό αποτέλεσμα που έχει παραχθεί μέχρι εκείνη την στιγμή και να υπάρχει η δυνατότητα να περιμένει ή όχι μέχρι να παραχθούν περισσότερα μηνύματα. Δεδομένου ότι δεν γνωρίζουμε ποιες θα είναι οι μελλοντικές εργασίες που θα πρέπει να διεκπεραιώνει ο πυρήνας, θα πρέπει η δομή να είναι ευέλικτη ώστε να επιτρέπει το αποτέλεσμα να είναι οποιαδήποτε τύπου. Τέλος, θα πρέπει να υπάρχει η δυνατότητα να υποστηρίζονται διαφορετικοί τρόποι αποθήκευσης των αποτελεσμάτων.

Για να επιτευχθούν όλοι οι παραπάνω στόχοι έχουν δημιουργηθεί 3 interfaces και οι 4 κλάσεις που υλοποιούν τα interfaces: οι WorkerResultIterable, WorkerResultIterableImpl, WorkerResultIterator, WorkerResultIteratorImpl, MessageRepositoryIterator, MessageRepository και InMemoryMessageRepository.

Ένας πελάτης που επιθυμεί να επικοινωνήσει με το νήμα ή τα νήματα που εκτελούν μία εργασία θα χρησιμοποιήσει ένα αντικείμενο που υλοποιεί το interface WorkerResultIterable. Σκοπός του αντικειμένου αυτού είναι να αποτελέσει μία επέκταση του interface Future που παρέχεται από το JDK και να προσφέρει ένα απλό και ομογενές API για σχεδόν οτιδήποτε μπορεί να χρειαστεί ο πελάτης. Ένα αντικείμενο που υλοποιεί αυτό το interface επιστρέφεται από τον WorkScheduler όταν καλείται η συνάρτηση submitWork() ή όταν κάποιο νήμα ξανακάνει αίτηση για να επικοινωνήσει με το νήμα με την κλήση της getIterableForWorkId(). Με αυτό το αντικείμενο μπορεί ο πελάτης να δει την κατάσταση της εργασίας με τις μεθόδους isPaused, isRunning, isCancelled και isDone, οι οποίες επιστρέφουν booleans. Επίσης, διαθέτει και τις μεθόδους continueWork, cancel και pause με τις οποίες μπορεί ο πελάτης να επηρεάσει την εκτέλεση της εργασίας. Και αυτές οι συναρτήσεις επιστρέφουν booleans, με τους οποίους δείχνουν στον πελάτη αν ήταν επιτυχής ή όχι η προσπάθεια να σταλεί το αντίστοιχο σήμα στο νήμα. Όλες αυτές οι

συναρτήσεις στέλνουν και δέχονται σήματα από μία κλάση που υλοποιεί το interface `IThreadState` που έχει περιγραφεί σε προηγούμενη παράγραφο. Επίσης, όπως έχει περιγραφεί σε προηγούμενες παραγράφους, το αντικείμενο που υλοποιεί το interface `IThreadState` δέχεται σήματα από το περιβάλλον σχετικά με το ποια είναι η επιθυμητή κατάσταση αλλά είναι επιλογή του ίδιου του νήματος που εκτελεί την εργασία το αν θα τα πάρει υπόψη του. Αυτό σημαίνει ότι αν ένας πελάτης χρησιμοποιήσει την μέθοδο `pause()` του `WorkerResultIterable` και η μέθοδος επιστρέψει την τιμή `true`, τότε απλώς το `pauseSignal` του `IThreadState` έχει πάρει την τιμή `true` και όχι ότι έχει διακόψει προσωρινά την λειτουργία του. Το νήμα `GeneralWorker` από μόνο του θα αποφασίσει αν και πότε θα παύσει την λειτουργία του.

Σε συνδυασμό με τις μεθόδους για τον έλεγχο και την αλλαγή της κατάστασης της εκτέλεσης μίας εργασίας, προσφέρονται και μέθοδοι για την αναμονή ενός πελάτη για αλλαγές στην κατάσταση μίας εργασίας. Συγκεκριμένα υπάρχουν οι μέθοδοι `waitUntilStarted`, `waitUntilWorkResumes` και `waitUntilCompleteOrFailed` που χρησιμοποιούνται για να σταματήσουν ένα πελάτη μέχρι είτε να συμβεί η αλλαγή στην κατάσταση που δείχνει και το όνομα της εκάστοτε συνάρτησης, είτε να περάσει το χρονικό διάστημα που έθεσε σαν όρισμα σε αυτές τις συναρτήσεις ο πελάτης. Αν το χρονικό διάστημα έχει την τιμή `-1` τότε οι συναρτήσεις αυτές θα σταματήσουν τον πελάτη επ' αόριστον, μέχρι είτε να συμβεί η εκάστοτε αλλαγή στην κατάσταση της εργασίας είτε μέχρι να σταματήσει εντελώς η εκτέλεση της εργασίας. Επίσης, προσφέρονται και οι μέθοδοι `registerStartOfWorkCallback()`, `registerPauseCallback()` και `registerOfWorkCallback()`. Οι μέθοδοι `waitUntilStarted`, `waitUntilWorkResumes` και `waitUntilCompleteOrFailed` εσωτερικά χρησιμοποιούν αυτές τις συναρτήσεις προκειμένου να καταχωρήσουν τα αντίστοιχα callbacks, αλλά δίνεται η δυνατότητα στον πελάτη να αποκτήσει πρόσβαση και στις κανονικές συναρτήσεις.

Η κλάση `WorkerResultIterable` κληρονομεί δύο άλλα interfaces: το `Future` και το `Iterable`. Από το interface `Future` κληρονομεί την μέθοδο `get()` εκτός από τις `isDone()`, `isCancelled()`, `isPaused()` που έχουν αναφερθεί ήδη. Η μέθοδος `get()` σύμφωνα με τον ορισμό του `Future` σταματάει την εκτέλεση του νήματος που την καλεί μέχρι να παραχθεί το αποτέλεσμα είτε μέχρι να περάσει ένα συγκεκριμένα χρονικό διάστημα. Όταν τελειώσει η εκτέλεση της εργασίας τότε η μέθοδος επιστρέφει το αντικείμενο `MessageRepository` που κρατάει όλα τα αποτελέσματα της εργασίας. Όμως, υπάρχει η δυνατότητα για ένα νήμα να μπορεί να δει το αποτέλεσμα μίας εργασίας με ένα `Iterator`. Για να το κάνει αυτό μπορεί να χρησιμοποιήσει την μέθοδο `iterator` που κληρονομείται από το interface `Iterable`, η οποία ανάλογα με την κατάσταση της εκτέλεσης της εργασίας φροντίζει να επιστρέψει είτε ένα αντικείμενο της κλάσης `MessageRepositoryIterator` είτε ένα αντικείμενο της κλάσης `WorkerResultBlockingIterator`. Η κλάση `MessageRepositoryIterator` είναι μία απλή

υλοποίηση του interface `Iterator` για την πρόσβαση στα μηνύματα ενός αντικειμένου του τύπου `MessageRepository`. Το συγκεκριμένο αντικείμενο φροντίζει απλά να επιστρέψει τα μηνύματα του `MessageRepository` με την σειρά που είναι αποθηκευμένα. Δεν έχει καμία εγγύηση για την σωστή λειτουργία του σε περίπτωση που ο `GeneralWorker` δεν έχει τελειώσει την εργασία του και για αυτό θα επιστραφεί μόνο αν κληθεί η μέθοδος `iterator` του `WorkerResultIterable` μετά το τέλος της εργασίας του `GeneralWorker`. Σε περίπτωση που ο `GeneralWorker` δεν έχει τελειώσει την εργασία του επιστρέφεται ένα αντικείμενο του τύπου `WorkerResultBlockingIterator`. Το συγκεκριμένο αντικείμενο είναι μία πιο περίπλοκη υλοποίηση του `Iterator`, προκειμένου να μπορεί να δώσει πρόσβαση, στο νήμα που το χρησιμοποιεί, στα μηνύματα του `MessageRepository` καθώς παράγονται. Αν τα μηνύματα έχουν τελειώσει χωρίς να έχει τελειώσει και ο `GeneralWorker` τότε το νήμα που χρησιμοποιεί τον `Iterator` θα σταματήσει μέχρι να υπάρξει καινούργιο μήνυμα είτε να σταματήσει την εργασία του ο `GeneralWorker`.

Το interface `MessageRepository` είναι υπεύθυνο για την αποθήκευση των αποτελεσμάτων που παράγει μία εργασία. Ο σκοπός των αντικειμένων που υλοποιούν αυτό το interface είναι να μπορούν να αποθηκεύσουν τα αποτελέσματα που παράγονται από ένα `GeneralWorker` και μετά να δώσει πρόσβαση σε αυτά τα αποτελέσματα με μεθόδους που μοιάζουν με τις μεθόδους που διαθέτει η κλάση `LinkedList`. Ο λόγος για τον οποίο το `MessageRepository` είναι ένα interface και όχι μία κλάση, είναι προκειμένου να υπάρχει ευελιξία στο μέλλον σχετικά με τον τρόπο με τον οποίο αποθηκεύονται τα μηνύματα. Αν στο μέλλον υπάρξει η ανάγκη να αποθηκεύονται τα μηνύματα σε μία βάση δεδομένων, σε αρχεία `yaml` ή σε μία ουρά μηνυμάτων όπως το `RabbitMQ`, τότε μπορεί να δημιουργηθεί ένα αντίστοιχο αντικείμενο και με την βοήθεια ενός `dependency injection framework` να χρησιμοποιηθεί δυναμικά, ανάλογα με τις ανάγκες του εκάστοτε χρήστη. Προς το παρόν, η υπάρχουσα υλοποίηση του `MessageRepository` είναι η κλάση `InMemoryMessageRepository`. Τα αντικείμενα αυτού του τύπου κρατάνε όλα τα μηνύματα στην μνήμη. Αυτός ο σχεδιασμός είναι απλός και προφανώς θα δημιουργήσει πρόβλημα διότι αν αφεθεί να εκτελέσει πολλές εργασίες το ZAP τότε υπάρχει περίπτωση να καταναλωθεί όλη η διαθέσιμη μνήμη δεδομένου ότι δεν υπάρχει κάποιος μηχανισμός για να επιλέγει και να καθαρίσει κάποια από τα αποτελέσματα που δεν είναι πλέον αναγκαία. Αλλά δεδομένου ότι ο σχεδιασμός του ZAP, μέχρι και τις αλλαγές που περιγράφονται σε αυτό το έγγραφο, κράταγε επίσης όλα τα μηνύματα στην μνήμη, η αλλαγή στο `MessageRepository` δεν πρόκειται να επιβαρύνει την λειτουργία του. Αλλά σε βάθος χρόνου, θα βοηθήσει στην μετάβαση του ZAP σε μία έκδοση που θα χρησιμοποιεί βάσεις δεδομένων για να μειώσει τις ανάγκες της εφαρμογής σε μνήμη.

2.7 Εκτέλεση κάποιου είδους εργασίας πάνω σε ένα κόμβο της ιστοσελίδας.

Το ZAP στην βάση του είναι ένα proxy tool. Αυτό σημαίνει ότι για κάθε αίτηση που δρομολογείται μέσα από το εργαλείο, καταγράφει τα δεδομένα που διέρχονται και δημιουργεί μέσα στην μνήμη του ένα χάρτη της ιστοσελίδας. Η δομή δεδομένων που χρησιμοποιείται δημιουργεί ένα γράφο όπου κάθε κόμβος αποτελείται από την απάντηση του server σε μία αίτηση. Προς το παρόν, κάθε εργασία που μπορεί να εκτελέσει το ZAP, εκτελείται πάνω σε ένα κόμβο αυτού του γράφου. Η τωρινή δομή του ZAP είναι τέτοια ώστε να υπάρχει άμεση πρόσβαση σε οποιοδήποτε κόμβο του γράφου. Δεδομένου ότι το εργαλείο μπορεί ανά πάσα στιγμή να έχει στην μνήμη του γράφους από πολλές ιστοσελίδες. Επίσης, η τωρινή μορφή δεν διαθέτει κανένα επίπεδο αφαίρεσης σχετικά με το πως είναι αποθηκευμένος ο γράφος. Για να λυθούν αυτά τα προβλήματα δημιουργήθηκε η κλάση DFSNodeIterator. Τα αντικείμενα αυτής της κλάσης παρέχονται από όλα τα αντικείμενα του τύπου WorkCommand τα οποία έχουν κάποια εργασία που απαιτεί πρόσβαση στους κόμβους μίας ιστοσελίδας. Όταν κάποιος πελάτης θέλει να εκτελέσει μία εργασία πάνω σε ένα εύρος κόμβων, τότε μαζί με τα υπόλοιπα ορίσματα περνάει και κάποια ορίσματα που υποδεικνύουν ποιοι κόμβοι πρέπει να εξεταστούν. Με βάση αυτά τα ορίσματα παράγεται το αντικείμενο του DFSNodeIterator που θα χρησιμοποιηθεί από τον αντίστοιχο GeneralWorker προκειμένου να επεξεργασθεί τους αντίστοιχους κόμβους. Όπως δηλώνει και το όνομα της κλάσης, το αντικείμενο υλοποιεί της μεθόδους του interface Iterator. Χρησιμοποιώντας τα ορίσματα του χρήστη, ο DFSNodeIterator κάνει ένα depth first search μέσα στον γράφο και επιστρέφει κάθε φορά τον επόμενο κόμβο που ικανοποιεί τα κριτήρια αναζήτησης. Τα κριτήρια αυτά μπορεί να είναι από το αντικείμενο ενός κόμβου από όπου θα αρχίσει η αναζήτηση μέχρι ένα regular expression με το οποίο θα αναγνωριστούν κάποιοι από τους κόμβους.

Η κλάση DFSNodeIterator θα επιτρέψει στο μέλλον μεγαλύτερη ευελιξία σχετικά με τον τρόπο με τον οποίο διαχειρίζεται το ZAP τον γράφο των ιστοσελίδων που χαρτογραφεί. Προς το παρόν όλοι οι γράφοι αποθηκεύονται στην μνήμη, αλλά υπάρχει ήδη σχεδιασμός προκειμένου να αλλάξει αυτό και να αποθηκεύονται όλοι οι κόμβοι σε μία βάση δεδομένων σε μελλοντική έκδοση. Ο λόγος που ο συγκεκριμένος iterator κάνει ένα depth first search μέσα στον γράφο για να βρει τον επόμενο κόμβο που θα επιστρέψει, είναι προκειμένου να εξασφαλιστεί ότι χρησιμοποιεί την ελάχιστη δυνατή μνήμη. Στην τωρινή μορφή του κώδικα, επειδή ο γράφος είναι ολόκληρος στην μνήμη, αυτή η δομή δεν προσφέρει κάποιο πλεονέκτημα, αλλά σε περίπτωση που ο γράφος είναι αποθηκευμένος σε κάποιο αρχείο ή σε μία βάση δεδομένων, θα βοηθήσει στην μείωση της ανάγκης για μνήμη. Επίσης, αν η εκάστοτε υλοποίηση του iterator είναι φτιαγμένη με τέτοιο τρόπο ώστε να μπορεί να χρησιμοποιηθεί αξιόπιστα από παραπάνω από ένα μήνα, τότε μπορεί να εύκολα να

χρησιμοποιηθεί για να συγχρονίσει την λειτουργία σε παραπάνω από ένα νήμα GeneralWorker. Δηλαδή, αν για παράδειγμα, δοθεί το ίδιο αντικείμενο WorkCommand σε τρία νήματα GeneralWorker και όλα χρησιμοποιήσουν τον ίδιο DFSNodeIterator για να πάρουν τους κόμβους του γράφου, τότε θα έχουμε εξασφαλίσει μία σχεδόν ίση κατανομή του φορτίου της επεξεργασίας ανάμεσα στα νήματα. Προς το παρόν ο GeneralWorker δεν μπορεί να υποστηρίξει αυτή την λειτουργικότητα γιατί είναι αρκετά απλοϊκός και επειδή χρειάζεται να υλοποιηθεί νέα λειτουργικότητα που να επιτρέπει πολλά αντικείμενα του τύπου ThreadState να συνομιλήσουν σαν ένα αντικείμενο με ένα MessageRepository και ένα WorkerResultIterable.

3. ΧΡΗΣΗ

3.1 Χρήση της κλάσης GeneralWorkerConcurrencyTester για την δημιουργία tests για το αντικείμενο ThreadState και των logs του.

Για να εξακριβωθεί ότι όλα τα επιμέρους τμήματα που περιγράφηκαν δουλεύουν με σωστό τρόπο, δημιουργήθηκε ένα test suite. Είναι σχετικά δύσκολο και χωρίς ιδιαίτερο όφελος το να κατασκευαστούν unit tests για κάθε κομμάτι ξεχωριστά, διότι όλες οι επιμέρους κλάσεις βασίζονται η μία στην άλλη για την σωστή λειτουργία τους. Για αυτό το λόγο η πρώτη υποκλάση του GeneralWorker που κατασκευάστηκε ήταν η κλάση GeneralWorkerConcurrencyTester. Η συγκεκριμένη κλάση έχει σαν σκοπό να δοκιμάσει το κατά πόσον το σύστημα μπορεί να λειτουργήσει σωστά στην περίπτωση που υπάρχουν πολλοί πελάτες ταυτόχρονα που στέλνουν τυχαία σήματα σε ένα αντικείμενο της κλάσης.

Πιο συγκεκριμένα, για την εκτέλεση των tests έχει δημιουργηθεί η κλάση GeneralWorkerTest με τη μέθοδο testWithExecutorService. Μέσα στην κλάση GeneralWorkerTest υπάρχουν άλλες δύο κλάσεις: οι RandomSignalTester και RandomWaiter. Και οι δύο παίρνουν ως όρισμα στον constructor ένα αντικείμενο της κλάσης ThreadState και είναι υποκλάσεις της κλάσης Thread. Εκτελώντας την συνάρτηση testWithExeccutorService, δημιουργείται ένα threadPool και μέσα του αφήνονται να εκτελεστούν 11 threads. Τα 5 είναι αντικείμενα του τύπου RandomWaiter, τα επόμενα 5 είναι του τύπου RansomSignalTester και ένα αντικείμενο της κλάσης GeneralConcurrencyTester. Τα 5 αντικείμενα της κλάσης RandomWaiter χρησιμοποιούν μία γεννήτρια ψευδοτυχαίων αριθμών μέσα σε ένα βρόχο προκειμένου να αποφασίσουν σε κάθε επανάληψη αν απλά θα παύσουν την λειτουργία τους για ένα δευτερόλεπτο, αν θα περιμένουν μέχρι την λήξη της εργασίας του GeneralConcurrencyTester με την μέθοδο waitForEndOfWork() της κλάσης ThreadState ή αν θα περιμένουν για την επόμενη αλλαγή κατάστασης με την μέθοδο waitForNextStateChange() της κλάσης ThreadState. Σε όλες τις περιπτώσεις το νήμα τερματίζει τον βρόχο μόλις εντοπίσει ότι το αντικείμενο GeneralWorkerConcurrencyTester έχει τερματίσει. Αντίστοιχα, το αντικείμενα της κλάσης RandomSignalTester χρησιμοποιούν και αυτά μία γεννήτρια ψευδοτυχαίων αριθμών μέσα σε ένα βρόχο για να αποφασίσουν τι είδους σήμα θα στείλουν στο αντικείμενο GeneralWorkerConcurrencyTester, δηλαδή, αν θα στείλουν σήμα στο νήμα να κάνει παύση, αν θα στείλουν σήμα να ακυρώσει την εκτέλεση του, αν θα στείλουν σήμα να

αρχίσει ξανά την εκτέλεση της εργασίας του ή αν θα περιμένει απλά για 1 δευτερόλεπτο. Προκειμένου να στείλει σήμα τα αντικείμενα `RandomSignalTester` δεν ελέγχουν την κατάσταση του νήματος `GeneralConcurrentTester`, αλλά στέλνουν μηνύματα τυχαία, διότι είναι ευθύνη του `ThreadState` αντικειμένου να φροντίσει να διαχειριστεί σωστά τα διάφορα σήματα που δέχεται και να αποφασίσει ποια είναι η σωστή τιμή. Σε κάθε περίπτωση και τα `RandomSignalTester` threads και τα `RandomWaiter` threads φροντίζουν αν εκτύπωνουν με ακρίβεια τι είδους ενέργεια πρόκειται να εκτελέσουν, με αποτέλεσμα τα logs της εφαρμογής να μπορούν να χρησιμοποιηθούν εύκολα για τον εντοπισμό τυχόν σφαλμάτων.

3.2 Δημιουργία του `DownloadWorker` και διεύρυνση του test suite.

Η πρώτη λειτουργία που ανακατασκευάστηκε ώστε να χρησιμοποιεί το νέο API είναι η κλάση `DownloadWorker`. Το ZAP έχει ανάγκη πολλές φορές να κατεβάσει φακέλους προκειμένου να εκτελέσει κάποια εργασία. Ένα παράδειγμα αυτής της συμπεριφοράς είναι όταν το ZAP χρειάζεται να κατεβάσει ένα καινούργιο jar αρχείο προκειμένου να ανανεώσει κάποια από τις λειτουργίες του ή να προσθέσει κάποια καινούργια στις ήδη υπάρχουσες. Όμως, ο `DownloadWorker` είναι χρήσιμος και διότι επιτρέπει την εύκολη διεύρυνση των υπαρχόντων tests ώστε να δοκιμαστεί η λειτουργία των διαφόρων μεθόδων για την επισκόπηση των μηνυμάτων ενός `GeneralWorker`. Ο `DownloadWorker` φροντίζει να παράξει ένα μήνυμα για κάθε νέο πακέτο των 1024 bytes που παραλαμβάνει. Έτσι, για ένα αρχείο μερικών megabyte, είναι σίγουρο ότι σε περίπτωση σωστής λειτουργίας, θα παραχθούν κατά τη διάρκεια του downloading. Οπότε μπορούμε εύκολα να εξακριβώσουμε την σωστή λειτουργία όλων των επιμέρους αντικειμένων που μεσολαβούν ανάμεσα στον πελάτη και στον `GeneralWorker`, δηλαδή την υλοποίηση του `MessageRepository`, την υλοποίηση του `WorkerResultIterator` και του `WorkerResultIterable`. Με το να ξεκινήσει να κάνει αίτηση για εργασία και αμέσως να ξεκινήσει να διαβάζει τα μηνύματα που παράγονται δοκιμάζοντας με αυτό τον τρόπο την σωστή λειτουργία του `BlockingIterator`. Αφότου τελειώσει την εργασία του ο `DownloadWorker`, τότε το test θα ξαναδιαβάσει όλα τα μηνύματα χρησιμοποιώντας αυτή την φορά τον απλό `WorkerResultIterator`. Αν τα μηνύματα συμπίπτουν με τα μηνύματα που διαβάστηκαν την πρώτη φορά τότε μπορούμε να είμαστε απόλυτα σίγουροι ότι η κλάση `BlockingIterator` εκτέλεσε σωστά την λειτουργία της.

3.3 Χρήση του GeneralWorker για την ανακατασκευή του ActiveScan extension.

Όπως αναφέρθηκε και σε προηγούμενη παράγραφο, το υπάρχον API του ZAP δεν παρέχει καμία υποδομή πάνω στην οποία να βασιστεί ο προγραμματιστής προκειμένου να προσθέσει κάποια νέα λειτουργία που να μπορεί να εκτελεστεί ασύγχρονα. Αυτό σημαίνει ότι ο προγραμματιστής που κατασκευάζει ένα νέο extension για το ZAP θα πρέπει να αποφασίσει αν θέλει να χρησιμοποιήσει νήματα ή όχι και σε περίπτωση που θέλει να χρησιμοποιήσει νήματα, ποιος θα είναι ο αριθμός τους και πότε θα εκτελεστούν. Σε περίπτωση που θέλει να εκτελέσει κάποιο κομμάτι της λειτουργίας του με νήματα θα πρέπει να φροντίσει από μόνος του να φτιάξει κάποιο μηχανισμό που να εκτελεί τα νήματα, κάποιο μηχανισμό επικοινωνίας με τα νήματα για τον έλεγχο της κατάστασής τους και την ανάκτηση των αποτελεσμάτων τους. Όλα αυτά όμως σημαίνουν ότι δεν υπάρχει κάποιος τρόπος να υπάρχει κάποιος κεντρικός έλεγχος του αριθμού και του χρονοπρογραμματισμού των νημάτων. Καλό παράδειγμα αυτού του προβλήματος είναι το ActiveScan extension.

Το ActiveScan extension είναι το μόνο extension που είναι κατασκευασμένο ώστε να εκτελεί την λειτουργία του με πολλαπλά νήματα. Για να το πετύχει αυτό χρησιμοποιεί την κλάση HostProcess η οποία με την σειρά της δημιουργεί ένα αριθμό από άλλα HostProcess αντικείμενα. Ο αριθμός τους εξαρτάται από το πόσους κόμβους θα δώσει ο χρήστης σαν όρισμα μέσω του γραφικού περιβάλλοντος για να ξεκινήσει τα tests το extension. Πιο συγκεκριμένα, όταν ο χρήστης δίνει την εντολή για να ξεκινήσει ένα active scan τότε δημιουργείται ένα αντικείμενο της κλάσης ActiveScan με μία λίστα από κόμβους από τους οποίους, για τον καθένα από τους οποίους και τα παιδιά τους θα εκτελεστούν μία σειρά από tests. Η ίδια η κλάση ActiveScan είναι υποκλάση της κλάσης Scanner που με τη σειρά της είναι υποκλάση της Runnable. Το αντικείμενο της κλάσης ActiveScan που δημιουργείται αφήνεται να τρέξει ξεχωριστά σαν ένα νήμα. Για κάθε κόμβο που δίνεται στο ActiveScan αντικείμενο για να ξεκινήσει τα tests δημιουργείται και ένα αντικείμενο του τύπου HostProcess το οποίο είναι επίσης ένα ξεχωριστό thread. Το κάθε αντικείμενο HostProcess αναλαμβάνει να εξερευνήσει τα παιδιά του αρχικού κόμβου που του δόθηκε και σε κάθε ένα να εκτελέσει όλα τα tests που ορίζονται στο αντικείμενο ScanPolicy. Για κάθε κόμβο που πρέπει να ελεγχθεί δημιουργείται ένα ξεχωριστό thread που τρέχει κάθε διαφορετικό plugin του ActiveScan. Αυτό σημαίνει ότι για παράδειγμα, για τρεις αρχικούς κόμβους θα δημιουργηθούν 4 αντικείμενα HostProcess και τα τρία από αυτά θα δημιουργήσουν n threads, όπου n είναι ο αριθμός των plugins που πρέπει να εκτελεστούν για κάθε κόμβο. Επίσης, κάθε HostProcess για να τρέξει τα διάφορα νήματα δημιουργεί ένα ξεχωριστό thread pool μέσα στο οποίο θα τρέξουν τα αντίστοιχα plugins. Αυτό σημαίνει ότι δημιουργείται μία ιεραρχία από HostProcesses, όπου γονιός είναι το πρώτο

HostProcess που δημιουργήθηκε και παιδιά είναι όλα τα υπόλοιπα. Τα μηνύματα που παράγονται από τα plugins στέλνονται πρώτα στο HostProcess που τα ξεκίνησε και στην συνέχεια μέσω callbacks στέλνονται στο αρχικό HostProcess και τέλος στο ActiveScan αντικείμενο. Σε περίπτωση που ο χρήστης επιθυμεί να σταματήσει πρόωρα την εκτέλεση των test τότε θα δώσει το αντίστοιχο σήμα στο ActiveScan το οποίο με την σειρά του θα προωθήσει το σήμα προς το αρχικό HostProcess αντικείμενο και στην συνέχεια προς την υπόλοιπη ιεραρχία των HostProcess αντικειμένων, τα οποία με την σειρά τους θα διακόψουν την εκτέλεση των νημάτων των plugins.

Η ανακατασκευή του ActiveScan χρησιμοποιώντας το νέο API ξεκίνησε με την δημιουργία του ActiveScanCommand αντικειμένου και του ActiveScanCommandBuilder. Το αντικείμενο ActiveScanCommandBuilder χρησιμοποιείται από τον πελάτη προκειμένου να συγκεντρώσει όλα τα απαραίτητα ορίσματα για την εκτέλεση του test. Όταν ο πελάτης έχει τελειώσει με την απόδοση τιμών στα πεδία του αντικειμένου, χρησιμοποιεί την μέθοδο build() προκειμένου να παράξει ένα νέο αντικείμενο ActiveScanCommand. Κάποια από τα βασικά ορίσματα τα οποία χρειάζεται για να εκτελεστεί σωστά η εργασία, είναι αυτά τα οποία υποδεικνύουν ποιοι κόμβοι της ιστοσελίδας θα πρέπει να εξεταστούν καθώς και τα ποια plugins θα χρησιμοποιηθούν. Ο χρήστης χρησιμοποιεί την μέθοδο addStartNode ή την addStartNodes για να προσθέσει ένα κόμβο ή ένα αντικείμενο του interface Collection με κόμβους από τους οποίους θα ξεκινήσουν τα tests. Υπάρχουν δύο συναρτήσεις addStartNode, η μία παίρνει ως όρισμα ένα αντικείμενο του τύπου SiteNode και η άλλη ένα αντικείμενο του τύπου StructuralNode. Τα αντικείμενα του τύπου SiteNode είναι τα αντικείμενα που αντιπροσωπεύουν στην μνήμη τους κόμβους της ιστοσελίδας. Το interface StructuralNode είναι ένα καινούργιο interface που έχει προστεθεί στις τελευταίες εκδόσεις του ZAP και θα χρησιμοποιηθεί στο μέλλον για να παρέχει ένα επίπεδο αφαίρεσης πάνω στο οποίο θα βασιστούν μελλοντικές κλάσεις κόμβων ιστοσελίδων τα δεδομένα των οποίων θα είναι αποθηκευμένα σε βάσεις δεδομένων. Εκτός από τους κόμβους, ο χρήστης μπορεί να δώσει σαν όρισμα και μία λίστα από urls, ώστε οι κόμβοι των οποίων τα url είναι ίσα με κάποιο από αυτά της λίστας δε θα ελεγχθούν. Αν δεν δώσει ο χρήστης κάποιους start nodes τότε μπορεί να δώσει ένα αντικείμενο του τύπου Target από το οποίο θα εξαχθούν είτε μία λίστα με τα startNodes είτε το context. Αν δεν δοθούν ούτε αρχικοί κόμβοι ούτε context τότε ο constructor του ActiveScanCommand θα παράξει ένα NullPointerException με αντίστοιχο μήνυμα. Τέλος, ο χρήστης μπορεί να ορίσει ένα μέγιστο βάθος για την έρευνα μέσα στον γράφο.

Ο χρήστης μπορεί να καθορίσει με ακρίβεια εκτός από το ποιοι κόμβοι θα ελεγχθούν και το ποια plugins θα εκτελεστούν χρησιμοποιώντας ένα αντικείμενο του τύπου ScanPolicy. Αν δεν δοθεί ένα αντικείμενο του τύπου ScanPolicy τότε κάθε φορά που θα καλείται η μέθοδος getScanPolicy() του ActiveScanCommand τότε θα επιστρέφεται ένα

αντικείμενο `ScanPolicy` με την default πολιτική. Αν όμως ο χρήστης επιθυμεί να χρησιμοποιηθούν κάποια διαφορετικά plugins τότε μπορεί να δημιουργήσει ένα αντικείμενο `ScanPolicy` με όρισμα στον constructor ένα αρχείο xml που περιγράφει ποια είναι τα επιθυμητά plugins. Σε αυτή την περίπτωση κάθε φορά που καλείται η μέθοδος `getScanPolicy()` τότε επιστρέφεται ένας κλώνος του αντικειμένου `ScanPolicy` προκειμένου να αποφευχθούν τυχόν προβλήματα, διότι το αντικείμενο `ScanPolicy` δεν είναι κατασκευασμένο για να χρησιμοποιείται από παραπάνω από ένα thread.

Η ίδια η εκτέλεση των active scans γίνεται από την κλάση `ActiveScanner`, η οποία είναι υποκλάση της `GeneralWorker` και υλοποιεί την μέθοδο `executeWork()` της υπερκλάσης της. Η μέθοδος `executeWork()` ξεκινά αρχικοποιώντας τα αντικείμενα `HttpServer` και `Analyser` που είναι αναγκαία για την λειτουργία των tests και στην συνέχεια παίρνει ένα αντικείμενο του τύπου `DFSNodeIterator` από την `ActiveScanCommand` που της έχει δοθεί σαν όρισμα και ξεκινάει να παίρνει κόμβους από εκεί. Για κάθε κόμβο που της ανατίθεται χρησιμοποιεί το αντικείμενο `PluginFactory` που επιστρέφει το αντικείμενο `ScanPolicy` για να πάρει όλα τα plugins που πρέπει να εκτελέσουν τα tests. Κάποιες φορές τα plugins μπορεί να μην έχουν φορτωθεί, για αυτό υπάρχει πρόβλεψη να περιμένει το νήμα κάνοντας διαλείμματα ανά 100ms για να ελέγξει αν έχει φορτωθεί το επόμενο plugin από το `pluginFactory`. Για κάθε νέο plugin που λαμβάνει από το `pluginFactory` εκτελεί το test στον κόμβο μέχρι να μην υπάρχουν άλλα plugins. Στο τέλος καλείται η μέθοδος `reset` του `pluginFactory` για να ξαναρχίσει η διαδικασία με τον επόμενο κόμβο που θα ληφθεί από τον iterator. Τα ίδια τα plugins δεν τρέχουν σε ξεχωριστά threads αλλά στο ίδιο thread με τον `ActiveScanner`.

Στην κανονική τους λειτουργία τα plugins για τα active scans χρησιμοποιούν κάποιες μεθόδους της κλάσης `HostProcess` προκειμένου να πάρουν κάποια σημαντικά για την λειτουργία τους αντικείμενα ή για να ειδοποιήσουν το `HostProcess` αντικείμενο που τα εκτελεί για κάποιο μήνυμα. Αυτές οι κλήσεις μεθόδων δε θα μπορούσαν να γίνουν αν τα plugins εκτελούνται από ένα νήμα της κλάσης `ActiveScanner`. Δεδομένου όμως ότι όλα τα plugins βασίζονται στις δύο abstract κλάσης `AbstractHostPlugin` και `AbstractParamPlugin` δημιουργήθηκαν τρία interfaces: τα `HasNotifications`, `HasKb` και `HasSenderAndAnalyser`. Με αυτά τα τρία interfaces μπορούν να διαχωριστούν εντελώς τα plugins από τα `HostProcesses` και να μπορούν να συνεργαστούν με οποιοδήποτε συνδυασμό αντικειμένων που υλοποιούν τα interfaces. Στην περίπτωση μας, τα interfaces προστέθηκαν στην υπογραφή της κλάσης `HostProcess` και υλοποιήθηκαν από την κλάση `ActiveScanner` ώστε να επιτρέψουν στην κλάση να εκτελέσει όλα τα plugins που εκτελεί και ένα `HostProcess` χωρίς περαιτέρω αλλαγές.

3.4 Δημιουργία ενός CLI για την επίδειξη του νέου ActiveScanner.

Για την ολοκληρωμένη παρουσίαση των δυνατοτήτων του νέου πυρήνα του ZAP είναι φανερό ότι το γραφικό του περιβάλλον είναι ανεπαρκές, διότι θα έπρεπε να επανακατασκευαστεί ένα μεγάλο κομμάτι προκειμένου να επιτρέψει την εκτέλεση πολλών εργασιών ταυτόχρονα και την παρακολούθηση των αποτελεσμάτων τους. Εκτός αυτού, ένα γραφικό περιβάλλον δεν έχει πλέον ιδιαίτερη αξία για ένα penetration tester που θέλει να χρησιμοποιήσει την εφαρμογή στα πλαίσια της εργασίας του. Το ZAP με την μορφή που θα πάρει στις επόμενες εκδόσεις, θα είναι ένα εργαλείο που θα εγκαθίσταται σε ένα server και ο χρήστης του θα έχει πρόσβαση σε αυτό απομακρυσμένα είτε μέσω ενός command line client είτε μέσω ενός web interface. Για τα πλαίσια της εργασίας επιλέχθηκε να δημιουργηθεί ένας command line client παρόμοιος με αυτόν που χρησιμοποιεί ο Jenkins, ένα από τα πιο γνωστά και ευρέως χρησιμοποιούμενα εργαλεία γραμμένα σε Java, με τρόπο λειτουργίας και χρήσης αρκετά παρόμοιο με αυτό του ZAP.

Ακόμα και μετά από όλες τις προσθήκες που έχουν γίνει στο ZAP, δεν έχει φτάσει ακόμα το εργαλείο σε μία μορφή που τέτοιου είδους προσθήκες να είναι ιδιαίτερα εύκολες. Για αυτό το λόγο ο command line client που δημιουργήθηκε δεν υποστηρίζει εντολές για όλη την λειτουργικότητα, αλλά μόνο για την αίτηση του χρήστη για την εκτέλεση ενός active scan. Από την μεριά του ZAP δημιουργήθηκε η κλάση ActiveScanRequestListener, η οποία αναλαμβάνει να δημιουργήσει ένα thread το οποίο θα ακούει για αιτήσεις στο port 33333 και να τις εξυπηρετεί. Ταυτόχρονα, έχει δημιουργηθεί και ένα jar αρχείο, με το οποίο ένας χρήστης μπορεί να δώσει εντολή να εκτελεστεί ένα active scan. Για να εκτελέσει ο χρήστης ένα active scan πρέπει πρώτα να φροντίσει να περιηγηθεί στους κόμβους της σελίδας που θέλει να εκτελέσει τα tests, είτε να χρησιμοποιήσει το spidering για να εξερευνήσει όλους τους κόμβους. Στην συνέχεια χρησιμοποιεί jar αρχείο περνώντας ως όρισμα το url του ZAP, την εντολή scan και το context path ή το url του ανώτερου κόμβου του δέντρου που θέλει να εκτελεστεί το active scan. Αφότου ο client συνδεθεί με το ZAP, ένα thread της κλάσης ActiveScanRequestHandler αναλαμβάνει να ξεκινήσει το scan και χρησιμοποιώντας τις κλάσεις που έχουν παρουσιαστεί σε προηγούμενες παραγράφους, μεταφέρει μέσω ενός socket τα αποτελέσματα και τα μηνύματα που παράγει το scan.

Για την δοκιμή του client χρησιμοποιήθηκε η εφαρμογή bodgeit, ένα διαδικτυακό κατάστημα το οποίο είναι κατασκευασμένο με τέτοιο τρόπο ώστε να έχει κάποιες συγκεκριμένα προβλήματα ασφαλείας προκειμένου να βοηθήσει άτομα που έχουν ξεκινήσει να ασχολούνται με τον τομέα της ασφάλειας υπολογιστών και είναι μία εφαρμογή

που απαιτεί μόνο ένα server container προκειμένου να εκτελεσθεί. Αφότου εκκινήσουμε την εφαρμογή, μέσω του browser περιηγούμεστε στην αρχική σελίδα του καταστήματος στο url <http://localhost:8000/bodgeit> και στην συνέχεια μέσω του ui του ZAP δίνουμε την εντολή να γίνει spider στην σελίδα προκειμένου να εξερευνηθούν όλοι οι προσβάσιμοι κόμβοι της. Στην συνέχεια εκτελούμε την εξής εντολή: `java -jar zap_cli.jar -s localhost -p 33333 -n bodgeit`. Με αυτή την εντολή θα συνδεθεί το πρόγραμμα στο ZAP και θα δώσει εντολή να εκτελεστούν active scans σε όλους τους κόμβους του bodgeit και θα επιστραφεί η πρόοδος της διαδικασίας. Παρακάτω παρατίθεται το αποτέλεσμα μίας εκτέλεσης της εντολής:

```
Sending command Scan bodgeit
Found the node with name bodgeit that you were looking for!
Scanning of node http://localhost:8000/bodgeit/score.jsp is started
Scanning of node http://localhost:8000/bodgeit/search.jsp?q=ZAP is started
Scanning of node http://localhost:8000/bodgeit is started
Examining of node http://localhost:8000/bodgeit/score.jsp with plugin Path Traversal is started
Examining of node http://localhost:8000/bodgeit/search.jsp?q=ZAP with plugin Path Traversal is started
HttpMessage for node http://localhost:8000/bodgeit request body: response body:
<html><head><title>Apache Tomcat/7.0.62 - Error report</title><style><!--H1 {fon...
Examining of node http://localhost:8000/bodgeit with plugin Path Traversal is started
Examining of node http://localhost:8000/bodgeit/score.jsp with plugin Path Traversal is finished
Examining of node http://localhost:8000/bodgeit with plugin Path Traversal is finished
Examining of node http://localhost:8000/bodgeit/score.jsp with plugin Remote File Inclusion is started
Examining of node http://localhost:8000/bodgeit with plugin Remote File Inclusion is started
Examining of node http://localhost:8000/bodgeit/score.jsp with plugin Remote File Inclusion is finished
Examining of node http://localhost:8000/bodgeit with plugin Remote File Inclusion is finished
Examining of node http://localhost:8000/bodgeit/score.jsp with plugin Server Side Include is started
.....
Alert found for node http://localhost:8000/bodgeit/search.jsp?q=ZAP with plugin Cross Site Scripting
(Reflected) : Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code
into a user's browser instance. A browser instance can be a standard web browser client, or a browser object
embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The
code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or
any other browser-supported technology.
```

When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.

There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.

Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.

```
(uri=http://localhost:8000/bodgeit/search.jsp?q=%3C%2Ffont%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Cfont%3E)
```

Solution: Phase: Architecture and Design

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Phase: Architecture and Design

For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

.....
Examining of node http://localhost:8000/bodgeit/search.jsp?q=ZAP with plugin Script active scan rules is finished

Scanning of node http://localhost:8000/bodgeit/search.jsp?q=ZAP is finished

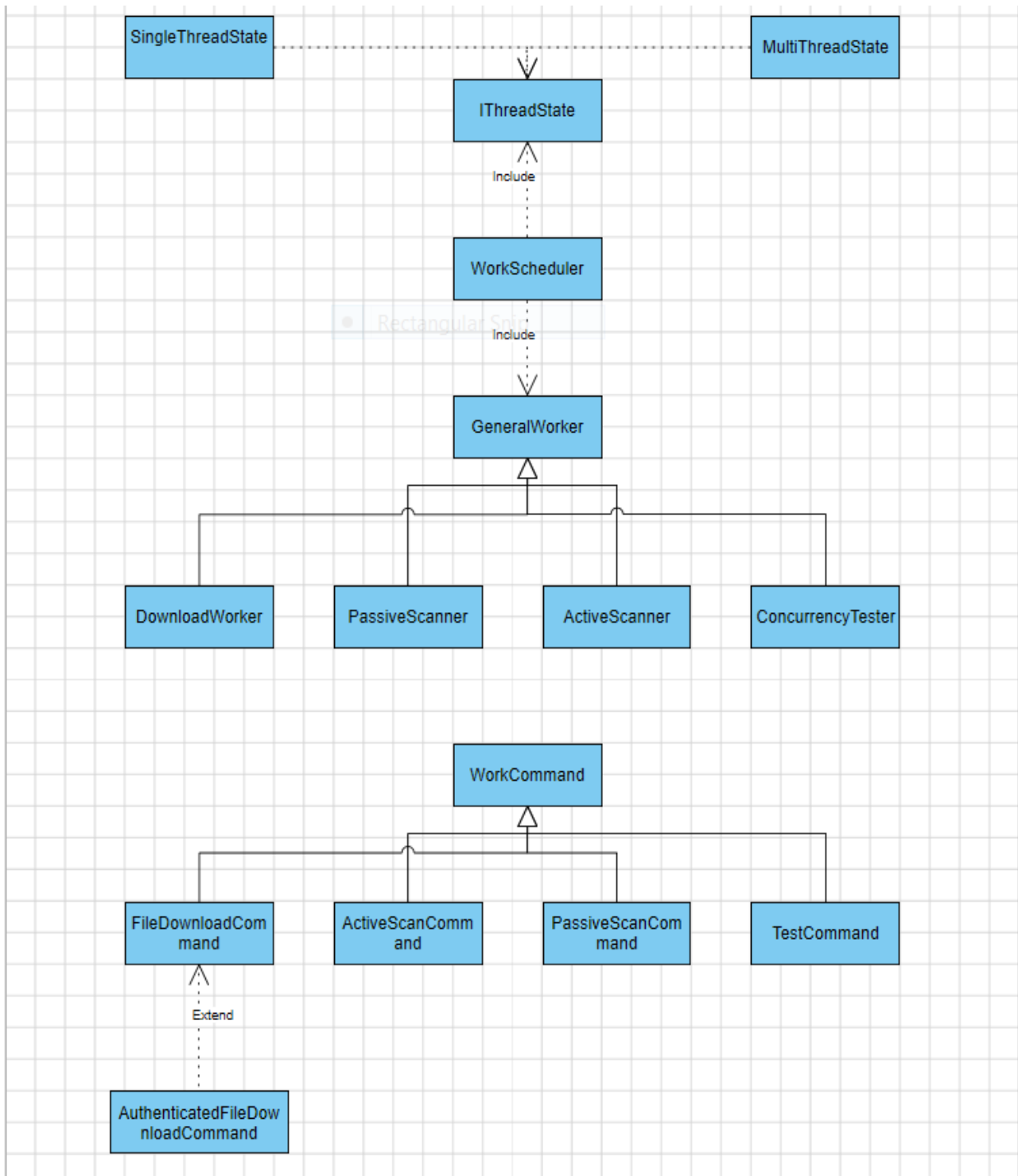
Current state is: FINISHED

The end

Στο παραπάνω τμήμα παρατίθεται ένα κομμάτι της εξόδου της εντολή με το κομμάτι στο οποίο αναφέρεται ότι βρέθηκε ένα πρόβλημα ασφαλείας καθώς και ένα κομμάτι της επεξήγησης του που οφείλεται το πρόβλημα και πώς μπορεί να λυθεί.

4. ΣΧΕΔΙΑΓΡΑΜΜΑ ΚΛΑΣΕΩΝ

Παρακάτω παρουσιάζεται ένα σχεδιάγραμμα με τις κυριότερες κλάσεις του API καθώς και τις σχέσεις τους:



ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	Ελληνικός Όρος
Penetration Tests	Παράλληλος προγραμματισμός
Vulnerability Assessment	Εκτίμηση Αδυναμιών
Open-source	Κώδικας Ανοικτού Λογισμικού
Closed-source	Κώδικας Κλειστού Λογισμικού
Thread	Νήμα Επεξεργασίας
Servers	Εξυπηρετητής
Cross-platform	Ανεξάρτητο πλατφόρμας
Framework	Εργαλειοθήκη
Tool	Εργαλείο
Error-based Query	Επερώτηση βασισμένη σε λάθος
Stacked Queries	Στοιβαγμένες επερωτήσεις
Time-based Blind Query	Επερώτηση βασισμένη στο χρόνο
Boolean-based Blind Query	Επερώτηση βασισμένη σε άλγεβρα Boolean
Web application	Διαδικτυακές Εφαρμογή
Distributed application	Διαμοιρασμένη Εφαρμογή
Cloud-based	Βασιζόμενη σε υπολογιστικό νέφος
Desktop	Υπολογιστής γραφείου
Extension	Επέκταση
Cluster	Σύμπλεγμα
Thread Pool	Δεξαμενή νημάτων επεξεργασίας
Configuration	Ρύθμιση
Application	Εφαρμογή

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

Pen Tester	Penetration Tester
RESTful	Compute Unified Device Architecture
OWASP	Open Web Alliance Security Project
ZAP	Zed Attack Proxy
API	Application Programming Interface
CLI	Command Line Client
UI	User Interface
GUI	Graphical User Interface
SaaS	Software as a Service
Web UI	Web User Interface

ΑΝΑΦΟΡΕΣ

- [1] J.P. McDermott, Attack Net Penetration Testing
- [2] Konstantinos Xynos, Iain Sutherland, Huw Read, Emlyn Everitt and Andrew J C Blyth, Penetration Testing and Vulnerability Assessments: A Professional Approach 4
- [3] Seth T. Ross, Unix System Security Tools