ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ Ψηφιακής Τεχνολογίας
ΤΜΗΜΑ Πληροφορικής και Τηλεματικής

Ανάπτυξη εφαρμογής επεξεργασίας δυαδικών αρχείων

Πτυχιακή εργασία

Κατσιφώλης Βασίλης

[Θέση Εικόνας: προαιρετικά]

Αθήνα, 2021

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ Ψηφιακής Τεχνολογίας
ΤΜΗΜΑ Πληροφορικής και Τηλεματικής

Τριμελής Εξεταστική Επιτροπή

Επιβλέπων

Κωνσταντίνος Τσερπές

Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Μέλη

Ανάργυρος Τσαδήμας

Ε.ΔΙ.Π, Τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Γεώργιος Κουσιουρής

Επίκουρος Καθηγητής, Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστημιό

Ο Κατσιφώλης Βασίλης δηλώνω υπεύθυνα ότι:

1. Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2. Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

Ευχαριστίες

Κείμενο

Περιεχόμενα

Περίληψη

Η συγκεκριμένη πτυχιακή εργασία κλήθηκε από την μια μεριά να παρουσιάσει, να μελετήσει και να συγκρίνει τους διάφορους hex editor που κυκλοφορούν με απώτερο σκοπό να υλοποιήσει ένα νέο hex editor βασισμένο σε λειτουργίες των εν λόγω προγραμμάτων. Από την άλλη μεριά επισκοπεί να παρέχει μια εισαγωγή στην έννοια του reverse engineering *RE* όπως και να μελετήσει μια τεχνική η οποία ονομάζεται *Binary Code Reuse detection* μέσω τριών επιστημονικών άρθρων. Η γλώσσα προγραμματισμού που επιλέχθηκε είναι η *c* για την ταχύτητα που προσφέρει αλλά και την ευκολότερη διεπαφή με το τερματικό για το οποίο θα αναπτυχθεί ο editor. Ορισμένα από τα βασικά χαρακτηριστικά του είναι η τροποποίηση ξεχωριστών μεμονωμένων byte, λειτουργία αντικατάστασης, αναζήτηση διεύθυνσης, αναγνώριση αρχείων (από την κεφαλίδα).

Λέξεις κλειδιά: [hex editor, multi-platform, μεγάλα αρχεία, reverse engineering, ανίχνευση επαναχρησιμοποίησης κώδικα]

Abstract

This dissertation was invited on the one hand to present, study and compare the various hex editors that are circulating with the ultimate goal of implementing a new hex editor based on the functions of these programs. On the other hand, it intends to give an introduction to the concept of reverse engineering *RE* as well as to study a technique called *Binary Code Reuse detection* utilizing three scientific papers. The programming language selected is *c* for the speed it provides and easy interface with the terminal for which the editor will be developed. Some of its key features are modification of individual bytes, replacement function, memory address offset search, file recognition (from the header)

Keywords: [hex editor, multi-platform, big files, reverse engineering, binary code reuse detection]

Κατάλογος σχημάτων

Κατάλογος πινάκων

Συντομογραφίες

RE	Reverse Engineering
BCRD	Binary Code Reuse Detection

1 Εισαγωγή

1.1 Το Πρόβλημα

Τα προγράμματα επεξεργασίας δυαδικών αρχείων (τα λεγόμενα προγράμματα επεξεργασίας *HEX editors*) είναι προγράμματα που προορίζονται για επεξεργασία αρχείων που δεν ερμηνεύονται παρά ως μπλοκ δεδομένων. Οι hex editors επί της ουσίας μπορούν να επεξεργαστούν - διαβάσουν όλα τα είδη αρχείων είτε αποτελούν εκτελέσιμα είτε όχι. Για παράδειγμα αρχεία πολυμέσων όπως png, jpg, gif, mp4, mkv, mp3, ogg και εκτελέσιμα αρχεία όπως .exe των *Windows*, elf των *Linux*, apk των *Android*.

Ενδέχεται να υπάρχουν αρκετοί λόγοι για την επεξεργασία τους από έναν *hex editor* όπως η μορφή αρχείου να είναι άγνωστη, η κεφαλίδα αρχείου να είναι κατεστραμμένη συνεπώς το αρχείο να είναι αδύνατο να ανοιχτεί ή ακόμα και να υπάρχει εξειδικευμένο λογισμικό για τη δεδομένη μορφή. Ένα άλλο παράδειγμα είναι η ανάκτηση διαγραμμένων αρχείων από τον σκληρό δίσκο. Η τεχνική της εύρεσης των κομματιών-μπλοκ για την `συναρμολόγηση` ενός διεγραμμένου αρχείου ονομάζεται *file carving*. Επίσης τα byte στην αρχή και στο τέλος ενός αρχείου (κεφαλίδα - header) διατίθονται για συγκεκριμένες πληροφορίες και μεταδεδομένα. Αυτό είναι σημαντικό για τους ανθρώπους στον κλάδο του *digital forensic* επειδή κακόβουλοι χρήστες θα αλλάξουν την επέκταση (και συνεπώς την κεφαλίδα) ενός αρχείου για να `καμουφλάρουν` αυτό το αρχείο από την μια μορφή που είναι σε κάποια άλλη.

Κάποιες από τις πιο περίπλοκες λειτουργίες περιλαμβάνουν την δυνατότητα κρυπτογράφησης και αποκρυπτογράφησης, υπολογισμού αθροίσματος ελέγχου *(checksum)*, κωδικοποίησης και αποκωδικοποίησης, και συμπίεσης και αποσυμπίεσης μπλοκ δεδομένων σε ένα αρχείο. Επί του παρόντος, υπάρχει ένας μεγάλος αριθμός προγραμμάτων που είναι σε θέση να κάνει επεξεργασία αυτών των αρχείων. Κάποια από τα εμπορίκα έχουν την δυνάτοτητα περίπλοκων λειτουργίων όπως είναι το Winhex το οποίο διαθέτει πρόγραμμα επεξεργασίας RAM, παρέχοντας πρόσβαση σε φυσική μνήμη και εικονική μνήμη άλλων διεργασιών. Όπως διαθέτει επίσης πρόγραμμα επεξεργασίας δίσκων για σκληρούς δίσκους, δισκέτες, CD-ROM DVD, ZIP, Smart Media, Compact Flash. Παράλληλα, τα προγράμματα ελευθέρου και ανοιχτού κώδικα όπως και τα εμπορικά διαθέτουν με την σειρά τους πληθώρα λειτουργιών. Όπως για παράδειγμα scripting με κάποια εξωτερική γλώσσα προγραμματισμού, *inline disassembly* και υποστήριξη ιδιαίτερα μεγάλων αρχείων.

Επίσης τα προγράμματα αυτά αποτελούν απαραίτητο εργαλείο για σενάρια reverse engineering. Η πρακτική της αντίστροφης μηχανίκευσης λογισμικού (software reverse engineering) αποτελεί σημαντική πρόκληση ιδιαίτερα στην μελέτη παρωχημένων (legacy) προγραμμάτων δίχως ο πηγαίος κώδικας να είναι διαθέσιμος και στην αντιμετώπιση πιθανών κινδύνων ασφάλειας ενάντια σε ιούς.

Εφαρμόζοντας reverse engineering στα επικείμενα προγράμματα έχει παρατηρηθεί πως είναι αρκετά απαιτητικό και χρονοβόρο. Δυσκολία επίσης συναντάται σε μοτίβα επαναχρησιμοποίησης εκτελέσιμου κώδικα.

1.2 Σκοπός της Εργασίας

Ο σκοπός της συγκεκριμένης πτυχιακής εργασίας είναι σε πρώτος μέρος να μελετήσει, αναλύσει τους hex editor που κυκλοφορούν και να υλοποιήσει ένα hex editor για τερματικό με τις βασικές λειτουργίες επηρεασμένες από την ανάλυση. Σε δεύτερο μέρος να μελετήσει την τεχνικη reverse engineering *binary code reuse detection* με τελικό σκοπό την ενσωμάτωσή της στο εν λόγω πρόγραμμα. Ταυτόχρονα, αυτός ο συντάκτης θα είναι σχεδιασμένος για τις κύριες πλατφόρμες: *MS Windows, Linux, Mac OS*. Ενσωματώνοντας τεχνικές reverse engineering ένας *hex editor* αποτελεί ένα ισχυρό αλλά και χρήσιμο εργαλείο.

Αναλυτικότερα, οι απαιτήσεις οι οποίες προδιαγράψαμε προκειμένου να υλοποιηθούν από την ανάλυση των hex editor (που ακολουθεί παρακάτω) είναι οι εξής:

- **Μετακίνηση**: Ο χρήστης θα έχει την δυνατότητα να μετακινηθεί σε οποιοδήποτε σημείο του αρχείου θέλει.
- **Εύρεση**: Ο χρήστης θα μπορεί να βρει το ή τα σημεία του αρχείου που υπάρχει το μοτίβο με βάση την δοθείσα συμβολοσειρά που θα παρέχει ο ίδιος.
- **Αντικαθιστώ**: Ο χρήστης θα μπορεί να αντικαθιστά bytes του αρχείου με bytes που αυτός επιθυμεί.
- **Αναγνώριση μορφής**: Το πρόγραμμα θα εντοπίζει την κεφαλίδα του δοθέντος αρχείου. Στην περίπτωση σφάλματος εμφανίζεται το ανάλογο μήνυμα στον χρήστη.
- **Υποστήριξη μεγάλων αρχείων**: Το πρόγραμμα θα υποστηρίζει αρχεία μέχρι 1TB σε λογικό χρονικό διάστημα.
- **Go To**: Ο χρήστης θα έχει την δυνατότητα να φτάσει σε οποιδήποτε σημείου του αρχείου μέσα σε ένα πολύ μικρό διάστημα
- **Code Reuse Detection**: Ο χρήστης θα έχει την δυνατότητα δίνοντας δύο εκτελέσιμα αρχεία να μελετήσει τα σημεία τα οποία ο εκτελέσιμος κώδικας είναι όμοιος ανάμεσα στα δύο αρχεία.

Η τεχνική reverse engineering που θα μελετηθεί, θα ανιχνεύει μοτίβα επαναχρησιμοποίησης κώδικα *(binary code reuse detection)*. Η ανίχνευση των μοτίβων μπορεί να εφαρμοστεί σε σενάρια όπως λογοκλοπή λογισμικού, παραβίαση αδειών λογισμίκου ή binary diffing. Οι επιστημονικές δημοσιεύσεις που θα βασιστεί αυτή η μελέτη είναι τρεις και θα αναφερθούν παρακάτω.

1.3 Δομή της Εργασίας

To Be Written ...

1.4 Δυσκολίες και Προκλήσεις

Η συγγραφή ενός hex editor δεν αποτελεί μια trivial υλοποίηση καθώς προϋποθέτει ακρίβεια και μεθοδική πρακτική για κάθε μια από τις λειτουργίες που την απαρτίζει. Συγκεκριμένα, οι δυσκολίες - προκλήσεις που βρέθηκα αντιμέτωπος ξεκινώντας από τις βασικές λειτουργίες ήταν η στοίχιση στο τερματικό, τα σινιάλα *signals* για τα διάφορα callbacks όπως dynamic resizing του τερματικού, για την ενεργοποίηση του *raw mode*...

Σε ό,τι αφορά το κομμάτι της μελέτης - υλοποίησης της τεχνικής *binary code reuse detection* βρέθηκα αντιμέτωπος με απαιτητικές αλγοριθμικές υλοποιήσεις. Ως αποτέλεσμα η υλοποίηση των δύο από των τριών επιστημονικών αναφορών είναι ελλιπής και μερική.

2 Δυαδικά προγράμματα επεξεργασίας κειμένων

Τα δυαδικά προγράμματα επεξεργασίας κειμένων ή αλλιώς hex editors αποτελούν εφαρμογές για την τροποποίηση και την κατανόηση προγραμμάτων τα οποία έχουν μεταφραστεί σε γλώσσα μηχανής. Η ονομασία "hex" (hexadecimal) ή δεκαεξαδίκο είναι σύστημα αρίθμησης με βάση το δεκαέξι. Περιέχει τους βασικούς αριθμούς 0-9 και τα γράμματα A-F. Το συγκεκριμένο σύστημα αρίθμησης χρησιμοποιείται ευρέως καθώς αντιπροσωπεύει σε πολύ καλό βαθμό τις δυαδικά κωδικοποιημένες τιμές *(binary-coded values)*. Υπάρχουν ποικίλες υλοποιήσεις των συγκεκριμένων προγραμμάτων με έμφαση σε διαφορετικά σενάρια.

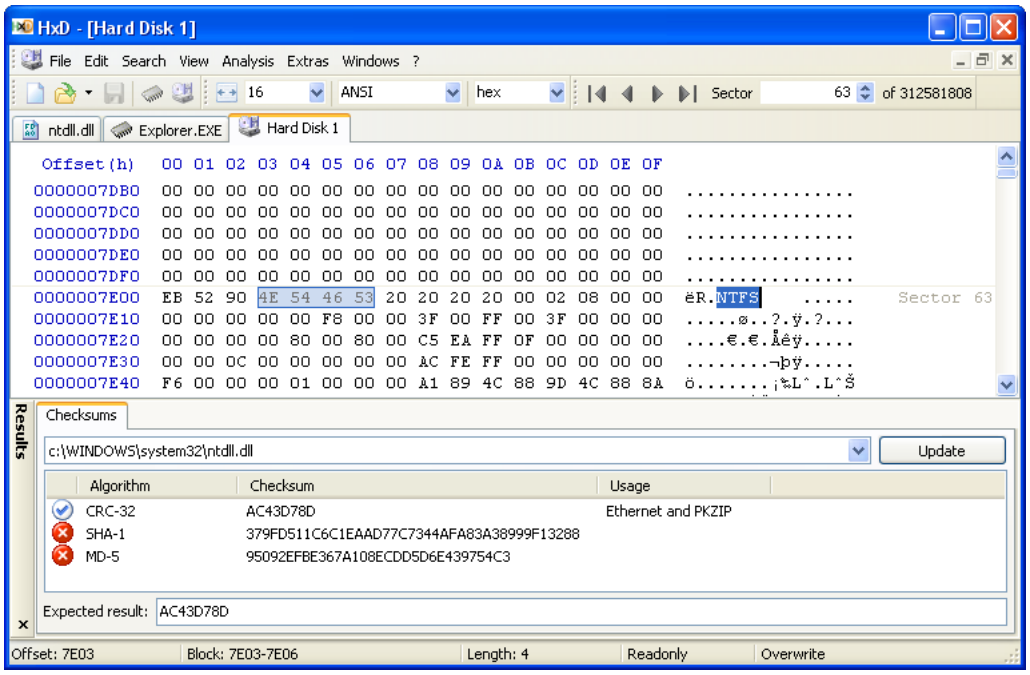
Το παρόν κεφάλαιο θα μελετήσει τις λειτουργίες και τα χαρακτηριστικά των διορθωτών που υπάρχουν στο διαδίκτυο σαν *freeware* ή *free and open source*. Επίσης θα θέσει μια κατευθυντήρια γραμμή με βάση τις απαιτήσεις και τις λειτουργίες στην οποία θα βασιστεί ο καινούργιος διορθωτής hex.

2.1 HxD Editor

Το *HxD Hex Editor* [5] είναι ένας επεξεργαστής που διανέμεται ελεύθερα *(freeware)* και έχει σχεδιαστεί για την πλατφόρμα των *Windows*. Από την γραφική διεπαφή μπορεί να αλλάξει κάποιος κωδικοποίηση μεταξύ πολλών κωδικοποιήσεων 8-bit. Από τις βασικές λειτουργίες, ο επεξεργαστής υποστηρίζει τις κλασσικές λειτουργίες ίδιου είδους προγραμμάτων όπως: αντιγραφή, επικόλληση, εύρεση και αντικατάσταση ή μετάβαση στη διεύθυνση. Μπορεί επίσης να διαχειριστεί μεγάλα σε μέγεθος αρχεία ή επεξεργασία τμημάτων μνήμης ή και δίσκου, συνένωση ή διαχωρισμό αρχείων *splitting - concat* όπως επίσης και στατιστικών μετρήσεων.

Ωστόσο, ο συγκεκριμένος διορθωτής υποστηρίζει σαν λειτουργικό σύστημα μόνο *Windows*. Βέβαια σε περιβάλλον *linux* ή macOS ενδείκνυται η αξιοποίηση βιβλιοθηκών τα οποία λειτουργούν σαν επίπεδα συμβατότητας *compatibility layer* όπως το wine *(Wine Is Not an Emulator)* και το *crossover* ή το bootcamp αντίστοιχα.

Τέλος ο συγκεκριμένος editor διαθέτει πολλές μεταφράσεις σε ξένες γλώσσες και μια από αυτές είναι η ελληνική η οποία δεν είναι ολοκληρωμένη.

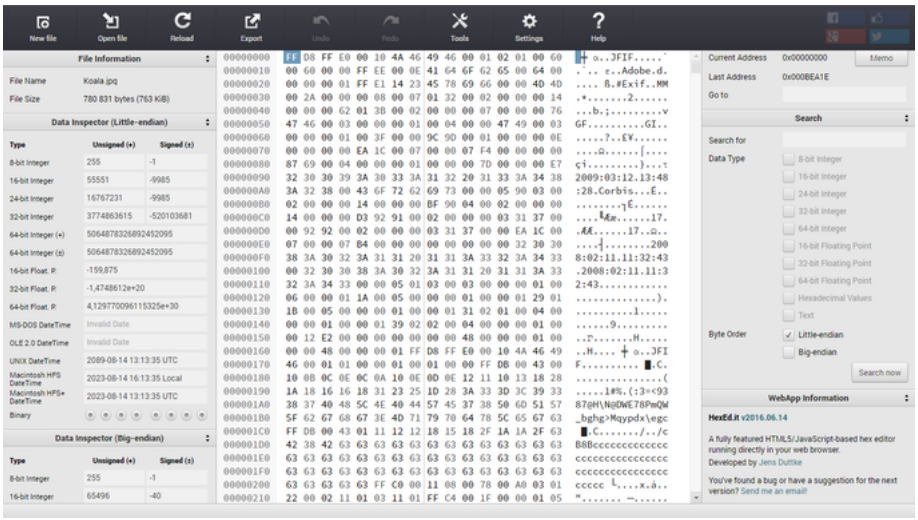


Σχήμα 1: HxD - Freeware Hex Editor and Disk Editor

2.2 Hexed.it

Το *hexed.it* [2] είναι ένας free hex editor ο οποίος λειτουργεί σαν web app στον περιηγητή ιστού. Αυτός ο *editor* προσφέρει λειτουργία χωρίς σύνδεση στο διαδίκτυο καθώς όλο το εκτελέσιμο τρέχει από την μεριά του χρήστη *(client side execution)*.

Επιτελεί και αυτός τις βασικές λειτουργίες ενός hex editor όπως εισαγωγή, αντικατάσταση, ανάζητηση συγκεκριμένης διεύθυνσης *(goto)*, απεριόριστα *undo-redo* οριοθετημένα από τις δυνατότητες του εκάστοτε περιηγητή. Ακόμα είναι ικανό να ανοίγει μεγάλα αρχεία της τάξης των *gigabyte* ακόμα και αν δεν υπάρχει διαθέσιμη μνήμη ram και να αναγνωρίζει την πλειοψηφία από τους τύπους αρχείων που κυκλοφορούν. Επίσης έχει την δυνατότητα της επιθεώρηση (data inspection) σε κάθε μεμονωμένο byte και ο χρήστης να βλέπει την τιμή του σε 8,16,32,64-bit αναπαράσταση.

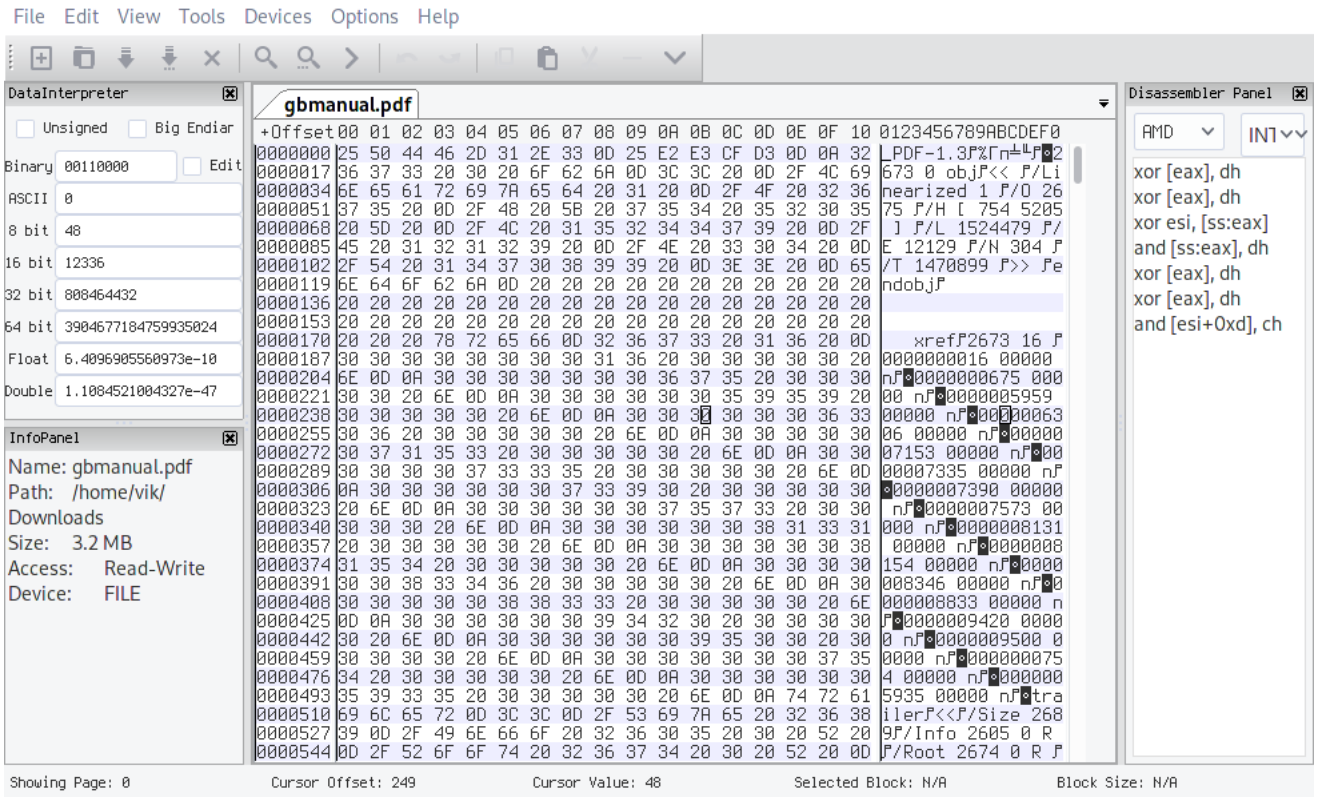


Σχήμα 2: HexEd.it: A full featured HTML5/javascript-based hex editor running directly from your browser

2.3 wxHexEditor

Το wxHexEditor [3] είναι ένα πρόγραμμα επεξεργασίας (hex editor) ανοιχτού κώδικα με μεγάλη υποστήριξη αρχείων. Η γραφική διεπαφή επιτρέπει στον χρήστη να επιλέξει μεταξύ διαφορετικών κωδικοποιήσεων, από ASCII έως UTF παραλλαγές. Ωστόσο, η επεξεργασία κειμένου είναι δυνατή μόνο σε κωδικοποίηση ASCII. Ο συντάκτης περιλαμβάνει έναν μεταγλωττιστή δεδομένων με υποστήριξη για βασικούς τύπους δεδομένων και εναλλαγή endianity. Εκτός από αυτές τις λειτουργίες, προσφέρει επίσης τη δυνατότητα σήμανσης τμημάτων δεδομένων χρησιμοποιώντας ετικέτες,ανάγνωση δεδομένων ως οδηγίες συναρμολόγησης (assembly), ανάγνωση δεδομένων από μνήμη (ram) συγκεκριμένης διαδικασίας.

Ένα άλλο πλεονέκτημα αυτού του προγράμματος επεξεργασίας είναι η διαθεσιμότητα μεταφράσεων σε ξένες γλώσσες. Το wxHexEditor υποστηρίζει πλατφόρμες *MS Windows, Mac OS και Linux*. Παρά τον μεγάλο αριθμό λειτουργιών , ο editor βρίσκεται ακόμη σε έκδοση beta και μπορεί να χρησιμοποιηθεί απλώς αντιμετωπίζει προβλήματα σταθερότητας.

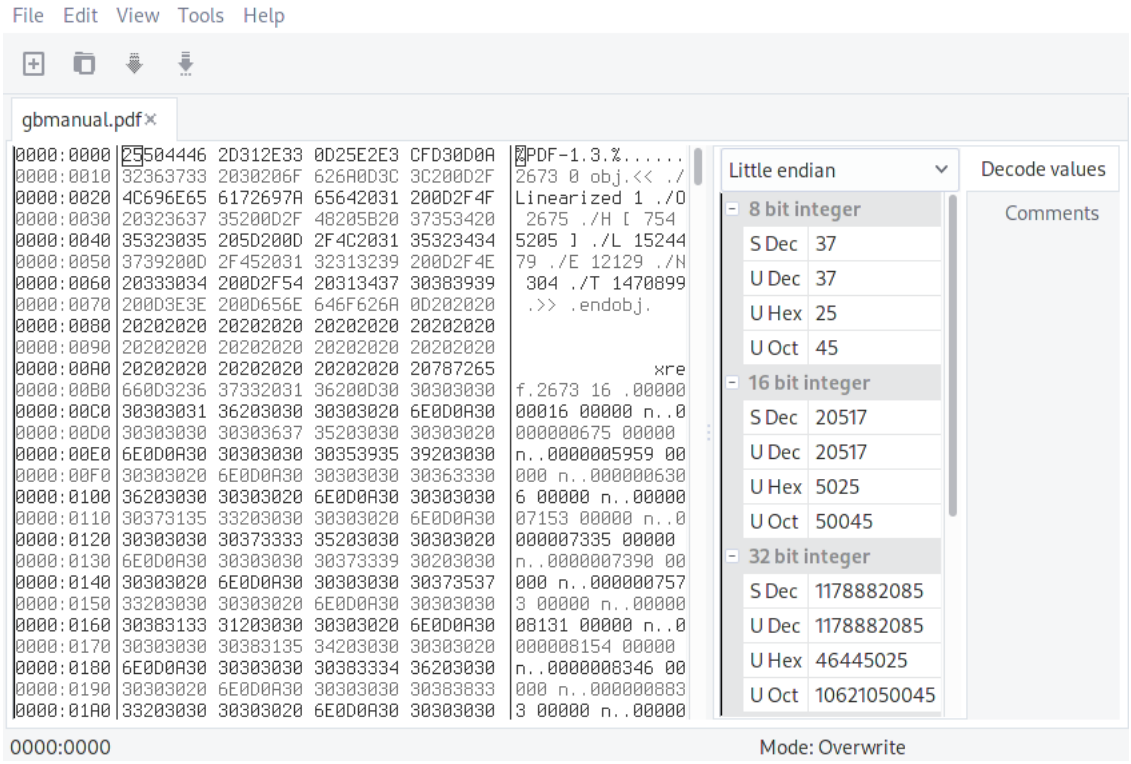


Σχήμα 3: wxHedEditor

2.4 rehex: Reverse Engineer's hex editor

Το *rehex* [1] είναι ένας σύγχρονος hex editor ανοιχτού και ελεύθερου λογισμικού. Η αρχική σελίδα του λογισμικού δίνει την περιγραφή ως *Ένας hex editor πολλαπλών πλατφορμών (Windows, Linux, Mac) για reverse engineering και οτιδήποτε άλλο*.

Ο συγκεκριμένος editor διαθέτει αρκετά παρόμοιες λειτουργίες με την λίστα των editor που έχουμε περιγράψει έως τώρα. Σημαντικό χαρακτηριστικό που τον αναδεικνύει από τους προηγούμενους είναι η δυνατότητα που έχει ο χρήστης για *inline dissasembly* της γλώσσας μηχανής. Διαθέτει και αυτός υποστήριξη για μεγάλα αρχεία της τάξης του 1TB+ όπως επίσης και την δυνατότητα scripting με κάποια εξωτερίκη γλώσσα προγραμματισμού.



Σχήμα 4: rehex: Reverse Engineer's hex editor

2.5 Περίληψη λειτουργιών

Ο παρακάτω πίνακας συγκρίνει τις επιλεγμένες λειτουργίες επεξεργαστή. Κάθε λειτουργία θα σημειώνεται για κάθε hex editor που την υλοποιεί

Λειτουργίες	HxD	HexEd.it	wxHexEditor	Rehex
Αντιγραφή και επικόλληση	×	×	×	×
Βρίσκω και αντικαθιστώ	×	×	×	×
Κάνε πίσω	×	×	×	×
Υποστήριξη μεγάλων αρχείων	×	×	×	×
Μεταγλωττιστής δεδομένων	-	×	×	×
Αναγνώριση μορφής	-	-	×	×
Go to	×	×	×	×
Inline disassembly	×	×	×	×

Πίνακας 1: Λειτουργίες hex editor

Συμπερασματικά, με βάση τους στόχους και τις λειτουργίες που είχαμε ορίσει στις απαιτήσεις του προγράμματος διαπιστώνουμε ότι από τους τέσσερις μόνο οι δυο hex editor υποστηρίζουν την λειτουργικότητα αναγνώρισης μορφής αρχείου.

Η προγραμματιστική υλοποίηση της πτυχιακής βασίστηκε και έγινε με την βοήθεια ενός text editor του kilo [12]. Η συγκεκριμένη επιλογή έγινε αφενός από την αρκετά ολοκληρωμένη τεκμηρίωση του κώδικα (*documentation*) αφετέρου από την επιλογή της γλώσσας προγραμματισμού την **C** που επιλέχθηκε για να γραφτεί ο *text editor*. Επίσης ο συγκεκρίμενος *text editor* στοχεύει το τερματικό το οποίο αποτελεί μια προϋπόθεση για τον *hex editor* της πτυχιακής. Περιέχει αρκετά χρήσιμες *non-trivial* συναρτήσεις που αφορούν την εύκολη αλληλεπίδραση με το περιβαλλόν του τερματικού τις οποίες δανείστηκε η υλοποίηση μου.

Υπάρχει και μια άλλη λειτουργικότητα προς υλοποίηση, η ανίχνευση επαναχρησιμοποίησης κώδικα **binary code reuse detection**. Για την επαναχρησιμοποίηση κώδικα πρέπει να στραφεί η προσοχή σε αλγόριθμους με τα χαρακτηριστίκα **binary diffing**. Συγκεκριμένα, επιλέχθηκαν τρεις επιστημονικές αναφορές που θα περιγραφούν παρακάτω.

2.6 Ανίχνευση επαναχρησιμοποιήσης κώδικα

Ο ορισμός του software reverse engineering αποδίδεται σύμφωνα με το Ινστιτούτο Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών *IEEE* ως ``η διαδικασία ανάλυσης ενός υποκείμενου συστήματος για τον προσδιορισμό των συστατικών του συστήματος και των συσχετισμών τους και για τη δημιουργία αναπαραστάσεων του συστήματος σε άλλη μορφή ή σε υψηλότερο επίπεδο αφαίρεσης"" [6] στο οποίο το "υποκείμενο σύστημα" αποτελεί το τελικό προϊόν της ανάπτυξης κώδικα *software development*.

Στο συγκεκριμένο κεφάλαιο θα γίνει μια επισκόπηση τριών επιστημονικών άρθρων της τεχνικής *binary code reuse detection*. Κάθε μία από τις υλοποιήσεις που προτείνονται στα κείμενα βασίζονται σε διαφορετικές δομές δεδομένων. Για παράδειγμα, στην υλοποίηση του bitshred [7] χρησιμοποιείται ένα *bloom filter* το οποίο είναι μία δομή που απαρτίζεται από συναρτήσεις κατακερματισμού και πίνακες bit. Από την άλλη το binsequence χρησιμοποιεί κατευθυνόμενους γράφους.

2.7 BitShred

Το paper `*BitShred: feature hashing malware for scalable triage and semantic analysis*` [7] προτείνει έναν ελαφρύ και scalable αλγόριθμο ανίχνευσης επαναχρησιμοποίησης κώδικα. Συνοπτικά τα βήματα που ακολουθεί για να επιτελέσει το έργο του είναι:

1. Θρυμματίζει το αρχείο *(*shredding).
2. δημιουργεί αποτυπώματα (*fingerprints*).
3. συγκρίνει τα αποτυπώματα.

Σε αρχικό στάδιο, τεμαχίζει (*shredding*) το αρχείο, αναλύοντας και εντοπίζοντας το εκτελέσιμο κομμάτι του binary. Έπειτα διαχωρίζει τα κομμάτια σε θραύσματα (*shreds*) τα οποία αποτελούν συνεχόμενες ουρές byte μήκους n, που συνήθως αποκαλούνται (*n-gram*).

Για να είναι αποδοτικός ο αλγόριθμος μέχρι και σε ογκώδη προγράμματα, η αποθήκευση των θραυσμάτων γίνεται με την χρήση των *bloom filters*. Ας υποθέσουμε ότι υπάρχει ένα σύνολο δεδομένων S. Ένα bloom filter μπορεί να κρίνει εάν ένα στοιχείο **x** είναι μέλος του **S** με αποδοτικό τρόπο αποθήκευσης. Τα bloom filters δεν έχουν ψευδή αρνητικά (*false negatives*) δηλαδή, οι δοκιμές συμμετοχής δεν επιστρέφουν ποτέ x $\notin$ **S** όταν **x** είναι πραγματικά μέλος του **S**. Στον πυρήνα τους αποτελούνται από μ πίνακες *bit* και ν διαφορετικές συναρτήσεις κατακερματισμού. Αρχικά, όλα τα bit του θέτονται 0. Η πρόσθεση ενός στοιχείου απαιτεί την εφαρμογή **k** hash functions στο στοιχείο και τα bit τα οποία ευρετηριάζονται από τις προκύπτουσες τιμές κατακερματισμού ορίζονται σε 1. Αφού προσθέσουμε όλα τα shreds στον bloom filter ο ίδιος θεωρείται πλέον το αποτύπωμα (fingerprint) του αρχείου.

Στο τελευταίο στάδιο, για να υπολογίσουμε την ομοιότητα μεταξύ των αρχείων χρησιμοποιείται ο δείκτης **Jaccard**. Ο δείκτης ορίζεται ως το μέγεθος τομής δύο δειγμάτων δια το μέγεθος ένωσης δύο δειγμάτων:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Με άλλα λόγια, εάν έχει οριστεί ένα συγκεκριμένο bit του A, τότε το αντίστοιχο χαρακτηριστικό του A είναι 1 αλλιώς είναι 0. Έτσι, ο δείκτης Jaccard μπορεί να υπολογιστεί ως εξής:

$$J(A,B) = \frac{F_{11}}{F_{01} + F_{10} + F_{11}}$$

όπου F_{11}: ο συνολικός αριθμός bit που ορίζονται τόσο από το A όσο και από το B, F_{01}: ο συνολικός αριθμός bit μόνο από το B , F_{10}: ο συνολικός αριθμός bit μόνο από το A.

2.8 BinJuice

Σε αυτό το κεφάλαιο θα μελετήσουμε την επιστημονική αναφορά *`Fast location of similar code fragments using semantic 'juice'`* [9]. Με τον όρο *juice* το συγκεκριμένο paper εννοεί μια γενίκευση της σημασιολογίας (*semantics*) ενός προγράμματος. Ο `χυμός` λαμβάνει υπόψη τις βασικές σχέσεις που δημιουργούνται από ένα κομμάτι κώδικα, ανεξάρτητα από τις επιλογές των καταχωρητών και των σταθερών. Ο `χυμός` στη συνέχεια χρησιμεύει ως πρότυπο του κώδικα που είναι αμετάβλητο έναντι συγκεκριμένων επιλογών από τους μεταγλωττιστές (*compilers*) ή με εργαλεία κωδικοποίησης κώδικα (*code obfuscation tools*).

Η διαδικασία εξαγωγής *binary juice* από ένα εκτελέσιμο αρχείο αποτελείται από τα ακόλουθα βήματα:

- Αποσυναρμολόγηση *(disassembly)* του binary.
- Αποσύνθεση του αποσυναρμολογημένου προγράμματος σε διαδικασίες (*procedures*) και block.
- Υπολογισμός της σημασιολογίας ενός block.
- Υπολογισμός του *juice* ενός block.

Για το πρώτο βήμα, έτσι όπως περιγράφεται στην αναφορά χρησιμοποιούνται εργαλεία όπως το IDA PRO [10] και το objdump [11]. Το disassembly που παράγεται από το τα προαναφερθέντα εργαλεία *(και άλλα)* δεν εγγυώνται ολοκληρωμένες λύσεις. Ως εκ τούτου το ποσοστό πληρότητας της σημασιολογίας και των υπολογισμών του χυμού βασίζονται σε ακριβείς λύσεις από τα δύο πρώτα βήματα.

Το αποτέλεσμα της αποσύνθεσης του δεύτερου βήματος πρέπει να αντιπροσωπεύει ένα γράφημα ροής ελέγχου (*Control Flow Graph*). Ένας κόμβος αυτού του γραφήματος είναι ένα μπλοκ: μια ακολουθία από instructions έτσι ώστε εάν η εκτέλεση ξεκινά από το πρώτο , ο έλεγχος θα περάσει μέχρι και το τελευταίο instruction και θα τερματίσει. Η σημασιολογία (*ή χυμός*) μιας διαδικασίας αποτελείται από ένα ισομορφικό γράφημα των οποίων οι κόμβοι αντιπροσωπεύουν τη σημασιολογία (ή χυμό) του αντίστοιχου κόμβου στο CFG. Όπως είναι εύλογο, η ουσία του αλγορίθμου έγκειται στον υπολογισμό της σημασιολογίας-χυμού των μεμονωμένων block.

Το τρίτο βήμα, αφορά τον υπολογισμό του χυμού των μπλοκ και των διαδικασιών με την χρήση ενός συμβολικού διερμηνέα (*symbolic interpreter*) με τον όρισμο:

$$\text{Interpret:seq(Instruction) x State} \rightarrow \text{State}$$

$$\text{where State} = \text{LValue} \rightarrow \text{RValue}$$

Κάθε πράξη από μία εντολή assembly κωδικοποιείται , όπως η ADD, και εκτελείται σε συμβολικές τιμές. Όποτε λοιπόν οι τελεστές της εντολής είναι γνωστό ότι αποτελούν τύπο *Int*, ο υπολογισμός εκτελείται αμέσως από τον διερμηνέα, οδηγώντας έτσι σε μια συγκεκριμένη τιμή. Ωστόσο εαν ένας από τους δύο τελεστές δεν είναι *Int* τότε η πράξη παγώνει στην μορφή $\mathrm{r}_1\ op\ r_2$.

Μαζί με τον διερμηνέα, ορίζεται και μια συνάρτηση η οποία εκτελεί αλγεβρική απλοποίηση ενός *RValue*:

Simplify: RValue →RValue

Συγκεκριμένα, η συνάρτηση αυτή εκτελεί έναν επιμεριστικό, προσεταιριστικο, αντιμεταθετικο μετασχηματισμό από μια συμβολική παράσταση σε μορφή αθροίσματος γινομένων (*sum-of-product*). Για παράδειγμα, οι παραστάσεις:

(def(eax) + 2) + def(ebx)

(def(eax) + def(ebx)) + 2

(2 + def(ebx)) + def(eax)

όλες παίρνουν την μορφή $2 + (def(eax) + def(ebx))$.

Η επιμεριστική ιδιότητα χρησιμοποιείται για να αναπαράγει μια έκφραση έτσι ωστε να διαδώσει τις πράξεις υψηλότερης προτεραιότητας πιο βαθιά στην παράσταση. Έτσι, η έκφραση $(def(eax)+2)\times def(eax))$ μετατρέπεται σε $(def(eax)\times def(eax))+(2\times def(eax))$. Ο αλγεβρικός απλοποιητής περιλαμβάνει επίσης κανόνες ταυτότητας και μηδενικά διαφόρων αριθμητικών και λογικών τελεστών. Αυτές οι ταυτότητες και τα μηδενικά χρησιμοποιούνται επίσης για την απλοποίηση των εκφράσεων, όπως η μείωση μιας έκφραση τής μορφής $(def(eax) - def(eax))\times def(ebx)$ στον ακέραιο 0.

Όπως αναφέρθηκε προηγουμένως, ο χυμός (juice) είναι μια γενίκευση της σημασιολογίας με περιορισμούς τύπου και αλγεβρικούς. Ενώ η σημασιολογία αποτελείται από βασικούς όρους, ο χυμός μπορεί να περιέχει λογικές μεταβλητές. Η γενίκευση της σημασιολογίας σε χυμό μπορεί να πραγματοποιηθεί αντικαθιστώντας συνεχώς τα ονόματα των καταχωρητών με λογικές μεταβλητές. Η αντικατάσταση βασίζεται στο ότι δύο εμφανίσεις του ίδιου ονόματος καταχωρητή αντικαθίστανται πάντα από την ίδια μεταβλητή.

Το πρόβλημα που παραμένει σε αυτό το σημείο είναι με ποιο τρόπο θα επιτευχθεί η δημιουργία των αλγεβρικών περιορισμών μεταξύ των λογικώ μεταβλητών. Για παράδειγμα ο περιορισμός $N2 = N1 \times N3$ σε μια παράσταση που δίνεται από το paper:

$$A = N1$$
$$B = def(B) \times N1 + N2$$
$$\text{where} \quad N2 = N1 \times N3$$
$$\text{and} \quad type(A) = type(B) = reg32$$

Η βασική ιδέα που προτείνεται είναι να αυξηθεί ο συμβολικός διερμηνέας για να παρακολουθεί τις απλοποιήσεις που εκτελεί. Για παράδειγμά, ο όρος 20 στην έκφραση $def(ebx) \times 5 + 20$ προκύπτει από την άμεση απλοποίηση της έκφρασης 5×4, η οποία με τη σειρά της προκύπτει από την επιμεριστική ιδιότητα πολλαπλασιασμού. Σε αυτό το παράδειγμα, ο διερμηνέας θα επισημάνει την σημασιολογία με την ταυτολογία $20 = 5 \times 4$. Στην συνέχεια, όταν εξάγεται ο `χυμός`, γίνεται μια γενίκευση των επισημάνσεων μαζί με την σημασιολογία δηλαδή ο όρος 20 αντικαθίσταται από το N2 και ο 5 από το N1 τόσο στην επισήμανση όσο και στην σημασιολογία αποφέροντας τον περιορισμό `N2 = N1 ×N3`.

Μια πιθανή υλοποίηση για να αποφασίσει κάποιος αποδοτικά εαν δυο κομμάτια κώδικα έχουν τον ίδιο `χυμό` είναι η ονομασία των μεταβλητών με την σειρά που πραγματοποιούνται οι αντικαταστάσεις με σκοπό να χρησιμοποιηθεί η προκύπτουσα σειρά για την σύγκριση. Έτσι οι όροι του `χυμού` μπορούν να καταταχθούν χρησιμοποιώντας γραμμική διάταξη. Εαν δυό τέτοιοι κατατεταγμένοι όροι ταιριάζουν τότε τα αντίστοιχα κομμάτια κώδικα θα είναι όμοια.

2.9 BinSequence

Το paper *`BinSequence: Fast, Accurate and Scalable Binary Code Reuse Detection `* [4] προτείνει μια δομή fuzzy matching η οποία στο κατώτερο επίπεδό της, συγκρίνει assembly blocks. Μια σύντομη περιγραφή των βημάτων που χρησιμοποιεί η συγκεκριμένη υλοποίηση ακολουθεί παρακάτω:

- Σε αρχικό στάδιο μια συλλογή από binary προγράμματα, αποσυναρμολογείται *(disassembly)* σε αποθετήρια συναρτήσεων γλώσσας μηχανής (assembly).
- Έπειτα χρησιμοποιείται μια τεχνική φιλτραρίσματος των συναρτήσεων στην οποία η έξοδος εύλογα, απαρτίζεται από ένα σύνολο υποψηφίων *(candidate set)*.
- Από το σύνολο εφαρμόζεται μια σύγκριση μία προς μία ως προς την συνάρτηση στόχο *(target function)* που επιθυμούμε να ταιριάξουμε με τα ακόλουθα βήματα.
    1. Παράγεται το μεγαλύτερο μονοπάτι *(longest path algorithm)* για την συνάρτηση στόχο.
    2. Αμέσως μετά εξερευνούμε την συνάρτηση αναφοράς από το αποθετήριο συναρτήσεων που ανήκουν στο σύνολο υποψηφίων για να βρούμε την αντιστοιχούσα διαδρομή.
    3. Βελτιώνεται η διαδικασία αυτή με την χρήση του αλγορίθμου *(neighbourhood exploration)* τόσο στην επικείμενη όσο και στην αναφερόμενη συνάρτηση.
- Η έξοδος αποτελείται από το *σκορ ομοιότητας* των δύο συναρτήσεων και την απεικόνιση των βασικών μπλοκ (εντολών) των συναρτήσεων.
- Αφού η διαδικασία έχει πραγματοποιηθεί για όλες τις συναρτήσεις, έχουμε στην διάθεσή μας τελικά μια βαθμολογική ιεραρχία των συναρτήσεων αντιστοίχισης.

Αρχικά, παρατηρούμε ότι στο πρώτο στάδιο όπως και στο προηγούμενο paper γίνεται η χρήση εργαλείων όπως το IDA Pro για την διαδικασία του disassembly των δοθέντων binary αρχείων και την εξαγωγή των γράφων ροής ελέγχου της κάθε συνάρτησης. Είναι σημαντικό να γίνει μια κανονικοποίηση (normalization) των κάθε εντολών assembly καθώς ο μεταφραστής (compiler) όσον αφορά τους καταχωρητές, τις θέσεις μνήμης, και τα μνημονικά εντολών (mnemonics) έχει πολλές επιλογές ως προς την δημιουργία τους. Η διαδικασία της κανονικοποίησης λαμβάνει υπόψη τους εξής περιορισμούς:

- Κανονικοποιούμε μόνο τους τελεστές και όχι την μνημονική τής εντολής.
- Χωρίζουμε τους τελεστές σε τρεις κατηγορίες: καταχωρητές (registers), memory references, immediate values.
    - Κανονικοποιήουμε περεταίρω τις immediate values σε διευθύνσεις μνήμης και σταθερές τιμές.

Στην συνέχεια, θα κοιτάξουμε με ποιον τρόπο επιτυγχάνεται η σύγκριση των assembly εντολών και η απόδοση ενός matching score. Μεταξύ δύο κανονικοποιημένων εντολών assembly εαν έχουν διαφορετικά mnemonic τότε το matching score τους θα είναι 0 ανεξάρτητα από τους τελεστές τους. Εαν οι αντίστοιχοι τελεστές είναι όμοιοι και μετά από την κανονικοποίηση τότε προστίθεται επίπλεον score. Εαν οι τελέστες αποτελούν σταθερές τιμές τότε συγκρίνονται και αυτές για την ομοιοτητά τους και προστίθεται αντίστοιχα το score. Με τους πειραματισμούς που κάναν κατά την διάρκεια συγγραφής του paper αποφασίσανε να δώσουν score 1, 2, 3 σε όμοιο τελεστή, mnemonic, σταθερά αντίστοιχα.

2.10 Περίληψη λειτουργιών

Στον παρακάτω πίνακα θα συγκρίνουμε τις λειτουργίες των 3 paper που περιγράφτηκαν στα προηγούμενα κεφάλαια και στην συνέχεια θα αποφασιστεί ποια υλοποίηση θα ενσωματωθεί στον hex editor.

Λειτουργίες	Bitshred	Binjuice	Binsequence
Υλοποίηση με γράφους	-	×	×
Υλοποίηση με bitarrays	×	-	-
Χρήση Hash functions	×	-	-
Βαθμός δυσκολίας υλοποίησης	Ήπια	Μέτρια	Μέτρια
Βαθμός δυσκολίας ενσωμάτωσης στον editor	Ήπια	Δύσκολη	Δύσκολη

Πίνακας 2: Τεχνικές υλοποιήσεων *Binary Code Reuse Detection*

Με βάση τον πίνακα και για τις γενικές απαιτήσεις θα υλοποιηθεί το paper του Bitshred και θα γίνει μια προσπάθεια ενσώματωσης στον editor. Οι λεπτομέρειες ακολουθούν παρακάτω.

3 Υλοποίηση

3.1 Υλοποίηση του hex editor

Σε αυτό το κεφάλαιο θα γίνει μια επισκόπηση της υλοποίησης του hex editor. Σε αρχικό στάδιο, αποφάσισα να υλοποιήσω τον hex editor σε περιβάλλον τερματικού και με την βοήθεια του *kilo* editor ο σκοπός επιτεύχθηκε ευκολότερα.

Ξεκίνησα ορίζοντας το βασικό struct του προγραμμάτος και το struct του ανοιχτού αρχείου:

```
struct E {
        char*        flname;          /* Filename */
        char*        data;            /* Data from file */
        char         status_msg[256]; /* Buffer for custom strings */
        char         search_str[20];  /* Search string buffer */
        long         data_len;        /* buffer length */
        int          size[2];         /* Size of the terminal */
        int          cx,cy;           /* cursor x, y */
        int          oct_offset;      /* octet offset */
        int          ln;              /* current line cursor */
        int          grouping;        /* grouping of data */
        int          dirty;           /* is it modified */
        enum e_mode  mode;            /* Editing mode of the editor */
};

struct buffer {
        unsigned char *data; /* Raw data */
        long          len;   /* Length of the buffer */
        long          cap;   /* How big is the initialized buffer */
};
```

Η πρώτη πρόκληση που ήρθε αντιμέτωπος ο editor ήταν με ποιο τρόπο ο χρήστης θα έκανε navigate μέσα στο ανοιχτό αρχείο. Δανείστηκα την συνάρτηση *enableRawMode* η οποία μέσω flags και signals επιτρέπει στον χρήστη να μετακινείται στο αρχείο χώρις οι χαρακτήρες που πληκτρολογεί να εμφανίζονται στην οθόνη του τερματικού. Έπειτα θα πρέπει να υπάρχει στο πρόγραμμα μια συνάρτηση η οποία διαβάζει ένα χαρακτήρα από το πληκτρολόγιο και εκτελεί την αντίστοιχη λειτουργία. Η συγκεκριμένη συνάρτηση *editorReadKey* παίρνει ως όρισμα τον *file descriptor* του ανοιχτού stream το οποίο είναι το *standard input* και διαβάζει τους χαρακτήρες. Η συνοδευτική της συνάρτηση είναι η *editorProcessKeypress* η οποία διαβάζει τους χαρακτήρες από την *editorReadKey* και εκτελεί την εκάστοτε λειτουργία.

Η μετακίνηση και το editing του editor έχουν υλοποιηθεί με βάση τη φιλοσοφία του **vim** η οποία διαχωρίζει το editing mode από το insert και το replace και χρησιμοποιεί χαρακτήρες για τις διάφορες λειτουργίες του. Οι λειτουργίες που περιγράφτηκαν στο πρώτο κεφάλαιο έχουν υλοποιηθεί ως εξής: Για την μετακίνηση χρησιμοποιήθηκαν τα *keybindings* hjkl για αριστέρα, κάτω, πάνω, δεξιά όπως και w ή b για την μετακίνηση 2 bytes την φορά. Για την εύρεση ο χρήστης εισάγει τον χαρακτήρα / και πληκτρογεί το search string του. Για την αντικατάσταση ο χρήστης εισάγει τον χαρακτήρα r (replace) και μπαίνει σε κατατάσταση (mode) replace επιτρέποντας την εισαγωγή χαρακτήρων διαγράφοντας τους ήδη υπάρχοντες. Για την αναγνώριση μορφής γίνεται εντοπισμός της κεφαλίδας του αρχείου και στην συνέχεια αντιστοιχίζεται με κάποια γνωστή μορφή χρησιμοποιώντας τον *magin number* της κεφαλίδας. Για το go to χρήστης θα χρησιμοποιεί τον ίδιο χαρακτήρα με την εύρεση αλλά με το πρόθεμα 0x.

...

3.2 Υλοποίηση του Bitshred

Η γλώσσα προγραμματισμού η οποία χρησιμοποιήθηκε για την υλοποίηση του Bitshred είναι η *python*. Η ευκολία χρήσης της συγκεκριμένης γλώσσας προγραμματισμού και η πληθώρα βιβλιοθηκών που υπάρχουν είναι η αιτία που χρησιμοποιήθηκε.

Αρχικά, για τον διαχωρισμό του εκτελέσιμου κώδικα σε binary μορφή χρησιμοποιούμε την βασική βιβλιοθήκη της python και το πρόγραμμα *readelf* από τα binutils για να ανοίξουμε το αρχείο και να βρούμε την τα κομμάτια (sections) του εκτελέσιμου κώδικα.

```
f = open(file, "rb")
x_seg = os.popen("readelf -SW " + str(file) + " | grep AX", "r")
```

Στην συνέχεια αποκτώντας τα κομμάτια του binary τα χωρίζουμε σε chunks:

```
for off, size in segment.items():
    f.seek(int(off, 16))
    chunk = f.read(int(size, 16))
    exec_str += chunk.hex()
```

Έπειτα κάθε ένα από τα chunks τα εισάγουμε στον bloomfilter περνώντας τα από την hash function η οποία θα πρεπεί να είναι γρήγορη και αποδοτική. Για αυτό τον λόγο χρησιμοποιούμε την *MurmurHash3* η οποία είναι κατάλληλη για το σενάριο μας δηλαδή για *hash-based lookups*.

```
bloom = BloomFilter(BLOOMSIZE, HASHCOUNT)
for vv in shreds:
    bloom.add(vv)
```

Τα δύο ορίσματα που παίρνει η BloomFilter είναι το μέγεθος του υποκείμενου bitarray *BLOOMSIZE* και οι φορές που θα κατακερματιστεί το shred *HASHCOUNT*.

Για να βρούμε τις αποδοτικές τιμές των ορισμάτων πρέπει να λάβουμε υπόψη τα εξής:

Πιθανότητα Λανθασμένης Θετικότητας: m είναι το μέγεθος του πίνακα bit, k είναι ο αριθμός των συναρτήσεων κατακερματισμού και n είναι ο αριθμός των αναμενόμενων στοιχείων που θα εισαχθούν στο φίλτρο, τότε η πιθανότητα ψευδώς θετικού p μπορεί να υπολογιστεί ως:

$$P = \left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)$$

Μέγεθος της σειράς Bit: n είναι γνωστός και η επιθυμητή ψευδής θετική πιθανότητα είναι p, τότε το μέγεθος της διάταξης bit m μπορεί να υπολογιστεί ως:

$$m = -\frac{n\ lnP}{(ln2)^2}$$

Βέλτιστος αριθμός συναρτήσεων κατακερματισμού: Ο αριθμός συναρτήσεων κατακερματισμού k πρέπει να είναι θετικός ακέραιος. Εάν το m είναι το μέγεθος bitarray και το n είναι ο αριθμός των προς εισαγωγή στοιχείων, τότε το k μπορεί να υπολογιστεί ως:

$$k = \frac{m}{n} ln2$$

Αφού βρήκαμε τις βέλτιστες τιμές για το μέγεθος του bitarray και τον αριθμό συναρτήσεων κατακερματισμού προχωράμε στο επόμενο στάδιο δηλαδή στον υπολογισμό του δείκτη *jaccard*.

Από το paper παίρνουμε τον τύπο του *jaccard index*:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \Rightarrow J(A,B) = \frac{F_{11}}{F_{01} + F_{10} + F_{11}}$$

Και τον μετατρέπουμε σε κώδικα:

```
def calc_jaccard(A, B):
  and_score = 0
  or_score = 0
  for a, b in zip(A, B):
      and_score += a & b  # F11
      or_score += a | b  # F01 + F10 + F11

  jaccard = and_score / or_score
```

Και έτσι με αυτό τον τρόπο καταλήγουμε σε ένα αποτέλεσμα το οποίο εκφράζει το ποσοστό ομοιότητας του εκτελέσιμου κομματιού των δύο προγραμμάτων.

4 Συμπεράσματα

Αναφορές

[1] Daniel Collins. https://github.com/solemnwarning/rehex.

[2] Jens Duttke. https://www.hexed.it/.

[3] erdem ua. https://sourceforge.net/projects/wxhexeditor/.

[4] He Huang, Amr M. Youssef, and Mourad Debbabi. BinSequence: Fast, Accurate and Scalable Binary Code Reuse Detection. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 155--166, Abu Dhabi United Arab Emirates, April 2017. ACM.

[5] Maël Hörz. https://www.mh-nexus.de/en/hxd/.

[6] Ati Jain, Swapnil Soner, and Anand Gadwal. Reverse engineering: Journey from code to design. In *2011 3rd International Conference on Electronics Computer Technology*, volume 5, pages 102--106, 2011.

[7] Jiyong Jang, David Brumley, and Shobha Venkataraman. BitShred. In *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*. ACM Press, 2011.

[8] Alexander Kowarik and Matthias Templ. Imputation with the R package VIM. *Journal of Statistical Software*, 74(7):1--16, 2016.

[9] Arun Lakhotia, Mila Dalla Preda, and Roberto Giacobazzi. Fast location of similar code fragments using semantic 'juice'. In *Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop on - PPREW '13*. ACM Press, 2013.

[10] IDA Pro. https://www.hex-rays.com/ida-pro/.

[11] The GNU Project. https://www.gnu.org/software/binutils/.

[12] Salvatore Sanfilippo. https://github.com/antirez/kilo.

[13] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*, 2016.